**Lay down a plan for CI/CD. As there will be tests pertaining to each microservice,**

Stage 1: Build, test, package and push microservices to ECR

Stage 2: Microservice service definition

Stage 3: Microservice deployment

#!/bin/bash -e

detect\_changed\_services() {

echo "----------------------------------------------"

echo "detecting changed folders for this commit"

# get a list of all the changed folders only

changed\_folders=`git diff --name-only $SHIPPABLE\_COMMIT\_RANGE | grep / | awk 'BEGIN {FS="/"} {print $1}' | uniq`

echo "changed folders "$changed\_folders

changed\_services=()

for folder in $changed\_folders

do

if [ "$folder" == '\_global' ]; then

echo "common folder changed, building and publishing all microservices"

changed\_services=`find . -maxdepth 1 -type d -not -name '\_global' -not -name 'shippable' -not -name '.git' -not -path '.' | sed 's|./||'`

echo "list of microservice "$changed\_services

break

else

echo "Adding $folder to list of services to build"

changed\_services+=("$folder")

fi

done

# Iterate on each service and run the packaging script

for service in $changed\_services

do

echo "-------------------Running packaging for $service---------------------"

# copy the common code to the service so that it can be packaged in the docker image

cp -r ./\_global $service

pushd "$service"

# move the build script to the root of the service

mv ./\_global/package-service.sh ./.

./package-service.sh "$service"

popd

done

}

detect\_changed\_services

**2. Define ECS cluster .**

resources:

- name: ecs\_cluster

type: cluster

integration: dr-aws

pointer:

sourceName : "mono-repo" #name of the cluster to which we are deploying

region: "us-east-1"

**Stage 3: Microservice deployment**

jobs:

- name: app\_deploy

type: deploy

method: replace

steps:

- IN: micro\_api\_def

- IN: micro\_www\_def

- IN: ecs\_cluster

**Prepare a deployment strategy that can be adapted for Staging, UAT and Production.**

The standard solution to the challenges of a 3-tier model is to push changes through it faster. That led to two twin concepts:

Continuous Integration (CI), which roughly translates to “get code integrated into Testing as fast as possible.”

Continuous Deployment (CD), which roughly translates to “get code through the pipeline and deployed to production as fast as possible.”

CI/CD, as it became known, tries to address the challenges of the 3-tier model with automation in order to get code through the process faster. That could entail a number of different options.

Automatically running code-level tests on new code before it even gets deployed anywhere.

Automatically running some QA or UAT tests in Testing, cutting down on the need for slow and over-busy humans.

Automatically advancing code through the pipeline when certain criteria are met (such as tests passing).