**Q1 - SCENARIO**

A car rental company called FastCarz has a .net Web Application and Web API which are recently migrated from on-premise system to Azure cloud using Azure Web App Service

and Web API Service.

The on-premises system had 3 environments Dev, QA and Prod.

The code repository was maintained in TFS and moved to Azure GIT now. The TFS has daily builds which triggers every night which build the solution and copy the build package to drop folder.

deployments were done to the respective environment manually. The customer is planning to setup Azure DevOps service for below requirements:

*1) The build should trigger as soon as anyone in the dev team checks in code to master branch.*

*2) There will be test projects which will create and maintained in the solution along the Web and API. The trigger should build all the 3 projects - Web, API and test.*

*The build should not be successful if any test fails.*

*3) The deployment of code and artifacts should be automated to Dev environment.*

*4) Upon successful deployment to the Dev environment, deployment should be easily promoted to QA and Prod through automated process.*

*5) The deployments to QA and Prod should be enabled with Approvals from approvers only.*

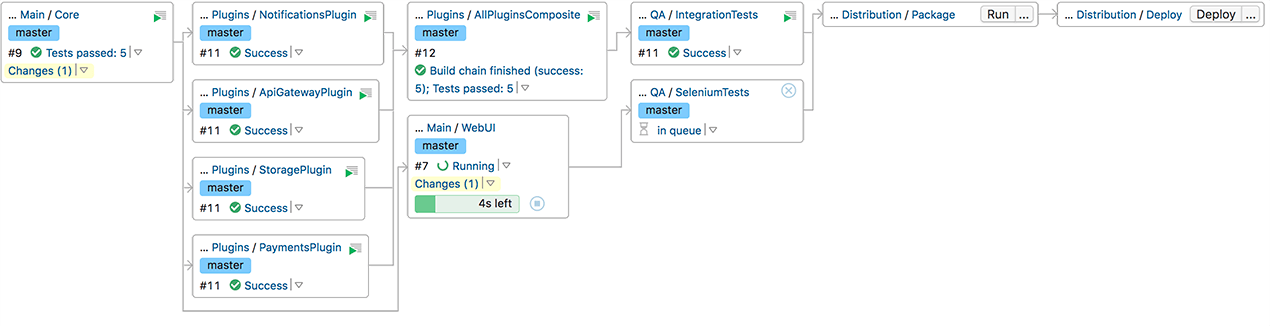
Explain how each of the above the requirements will be met using Azure DevOps configuration.

Explain the steps with configuration details.

Solution :

As Said we have 3 environments Dev, QA, Prod We can maintain these 3 environments either in docker or on Cloud servers.

As we know it is almost an end for TFS as all are migrating to git, We are maintaining our source code in Git I am sure as it is a .net cod we should use MS Build to build the code Below is the sample automated pipeline.



The pipeline code which we used is like below.

// Set Job Properties

import groovy.json.JsonOutput

import groovy.json.JsonSlurper

properties([

buildDiscarder(logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '', numToKeepStr: '100')),

parameters([

string(defaultValue: '', name: 'ENVIRONMENT\_NAME', trim: true, description: '''Here you add the environment you want to generate the daily build, Example: int01.red.ref.eu-west-1'''),

string(defaultValue: '', name: 'RELEASE\_VERSION', trim: true, description: '''You can add your release version, example: v9.2.64'''),

string(defaultValue: 'sportsbook', name: 'PRODUCT', trim: true, description: '''You can add your product, example: sportsbook'''),

booleanParam(defaultValue: false, name: 'LIVE\_VERSION', description: '''Set this if your release is going to be live''')

])

])

if (params.ENVIRONMENT\_NAME == "") {

echo "Please name your environment!"

return

}

if (params.RELEASE\_VERSION == "") {

echo "Please name your release version!"

return

}

if (params.PRODUCT == "") {

echo "Please name your product!"

return

}

def jsonSlurper = new JsonSlurper()

// for example: int01.red.ref.eu-west-1

def ENVIRONMENT = params.ENVIRONMENT\_NAME.tokenize('.')

def env = params.ENVIRONMENT\_NAME

def env\_name = ENVIRONMENT[0]

def env\_color = ENVIRONMENT[1]

def env\_operator = ENVIRONMENT[2]

def env\_region = ENVIRONMENT[3]

def product = params.PRODUCT

def release\_version = params.RELEASE\_VERSION

def lock\_id = "daily-build"

def server = "dev03.openbet"

def customer\_repo\_folder = "${env}\_release"

def package\_list\_file = "${product}\_${release\_version}.yml"

def remote\_env\_path = "/shared/prj/daily\_build/support\_${env\_operator}/${env}"

def remote\_product\_path = "/shared/prj/daily\_build/product-${product}"

def remote\_env\_release\_path = "${remote\_env\_path}/${release\_version}"

currentBuild.displayName = "${env}-${release\_version}"

echo "LockID: " + lock\_id

lock(resource: lock\_id) {

node('sourcepack') {

stage('Local: Prepare') {

sh "mkdir -p ${customer\_repo\_folder}"

sh "ssh ${server} 'mkdir -p ${remote\_env\_release\_path}'"

}

stage("Package-List: Generate") {

command = [

"sb-product package-list",

"-r ${env\_operator}",

"-v ${release\_version}",

"-p ${product}",

"-o yml",

"| tee ${customer\_repo\_folder}/${package\_list\_file}"

].join(" ")

command = $/eval "${command}"/$

content = sh(script: "${command}", returnStdout: true).trim()

}

stage('Daily-Build: Downloading') {

args = [

"/sourcepack/daily-build.py",

"-a",

"--rules /sourcepack/package-list-rules.yaml",

"--output\_dir ${customer\_repo\_folder}",

"${customer\_repo\_folder}/${package\_list\_file}"

]

sh args.join(" ")

}

stage('Daily-Build: Details') {

depth = 2

local\_components = sh(script: "find ${customer\_repo\_folder} -maxdepth ${depth} -mindepth ${depth} -type d", returnStdout: true).tokenize("\n")

sh "ls -l ${customer\_repo\_folder}"

for (local\_component in local\_components) {

sh "du -sh --summarize ${local\_component}"

}

sh "du -sh --summarize ${customer\_repo\_folder}"

}

stage('Daily Build: Push') {

sh "chmod -R 755 ${customer\_repo\_folder}"

for (local\_component in local\_components) {

local\_component\_list = local\_component.tokenize("/")

version = local\_component\_list[-1]

component = local\_component\_list[-2]

local\_path = "${local\_component}"

remote\_path = "${remote\_product\_path}/${component}/"

sh "ssh ${server} 'mkdir -p ${remote\_path}'"

sh "time rsync -az --delete-after ${local\_path} ${server}:${remote\_path}"

}

}

stage('Remote: Sym Links') {

for (local\_component in local\_components) {

local\_component = local\_component.tokenize("/")

version = local\_component[-1]

component = local\_component[-2]

link\_path = "${remote\_env\_release\_path}/${component}\_${version}"

target\_path = "${remote\_product\_path}/${component}/${version}"

remote\_cmd = "ln -sfn ${target\_path} ${link\_path}"

sh "ssh ${server} '${remote\_cmd}'"

}

if (params.LIVE\_VERSION == true) {

link\_path = "${remote\_env\_path}/live"

target\_path = "${remote\_env\_release\_path}"

remote\_cmd = "ln -sfn ${target\_path} ${link\_path}"

sh "ssh ${server} '${remote\_cmd}'"

}

}

stage('Remote: Validate') {

for (local\_component in local\_components) {

local\_component = local\_component.tokenize("/")

version = local\_component[-1]

component = local\_component[-2]

ls\_path = "${remote\_env\_release\_path}/${component}\_${version}"

remote\_cmd = "ls ${ls\_path} | wc -l"

sh(script: "ssh ${server} '${remote\_cmd}'")

}

}

stage('Remote: Folder Structure') {

remote\_cmd = "tree -d -L 2 ${remote\_env\_release\_path}"

sh(script: "ssh ${server} '${remote\_cmd}'")

}

stage('Local: Cleanup') {

deleteDir()

}

}

}

The above code will build the code and placed in Jenkins by referring some of the python files which I did not include as those contains some of our organization info.

We are using vault to store the passwords we can get log in using the below commands,

vault login -method=ldap username=\*\*\*\*\* password=xxxxxx

We use python scripts to get these vault passwords to get authenticated.

Below are the some of playbooks we used to provision new environments.

- hosts: \*\*\*\*\*\*

tasks:

- name: check for file

local\_action:

module: stat

path: "{{ playbook\_dir }}/envinfo/env\_info.yml"

register: check\_envinfo

run\_once: True

- debug:

msg: "{{ playbook\_dir }}/envinfo/env\_info.yml"

- include\_role:

name: envinfo

when: check\_envinfo.stat.exists

Above are the examples which we are using in our organization.

As we know the poll scm or git web hooks will trigger the build immediately after checkin.

And we can create a deployment after successful build to dev environment through pipeline as a code or shell or python.

Once changes verified we can give a Jenkins job access to dev and QA teams to trigger deployments to QA or some other lower environments.

We can integrate Jenkins with jira when ever the approver approves it will trigger deployment as well.

Even we know in pipeline any job will stopped it fails anywhere and the deployment also void.

May be I can explain bit more if we joined in a conversation if the provided info is not Enough.

Regards,

Hari