How To Use DevOps Utilities

--- Build Customization Docker Image

Precondition

1. Please make sure you have following guide to setup the DevOps Utilities Environment

See : <https://github.com/IBM/wc-devops-utilities/raw/master/doc/SetupDevOpsSystem.docx>

1. Plan your environment:

DevOps Utilities use Namespace/Tenant/Env/EnvType pattern to manage all resources, so before you start trigger build, you must have clear target

For this guide, we assume:

Namespace: default

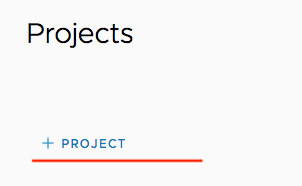
( if you want to deploy on other namespace, you need to create namespace on Kubernetes first )

Tenant: demo

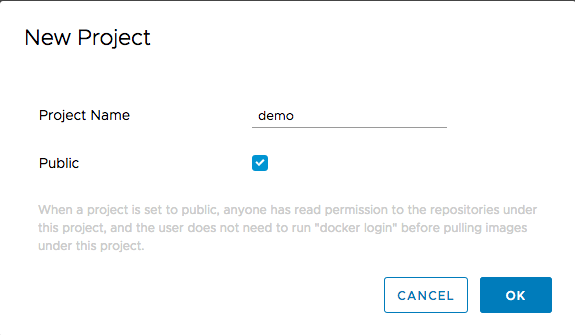
Env: qa

EnvType: auth

1. Created target project for target tenant on Harbor ( docker repository )
2. Log on harbor
3. Click Add Project Button



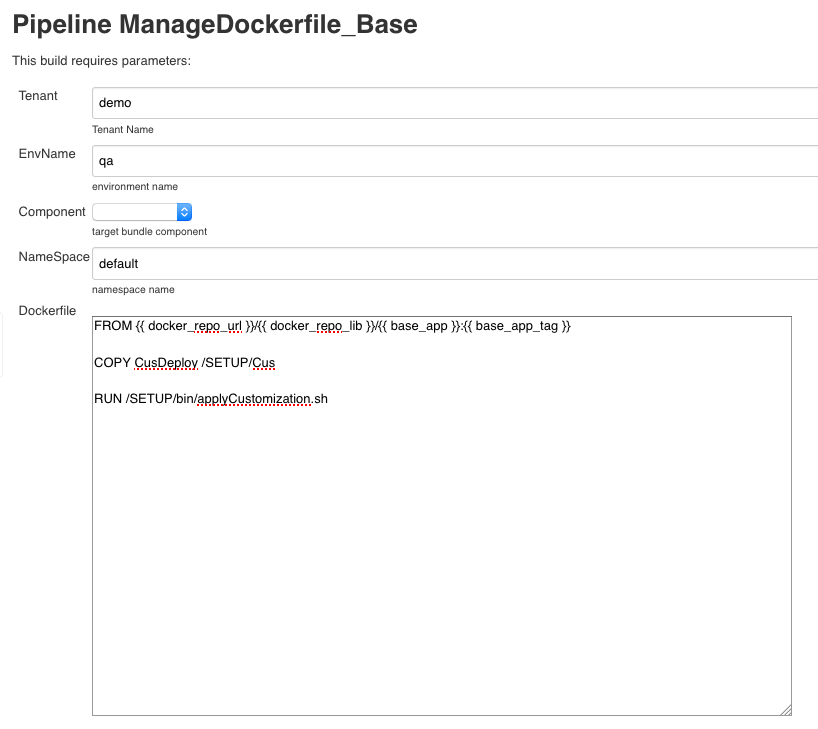
1. Input Project name “demo” and select “public” and save



High Level Design

Detail Steps

1. Logon DevOps Utilities Controller
2. Create Dockerfile For Target Env
3. Open Jenkins Job “ManageDockerfile\_Base”
4. Click “Built with Parameters” button
5. Input Configuration As Below and Click Save Button



Sample Of Dockerfile template:

FROM {{ docker\_repo\_url }}/{{ docker\_repo\_lib }}/{{ base\_app }}:{{ base\_app\_tag }}

COPY CusDeploy /SETUP/Cus

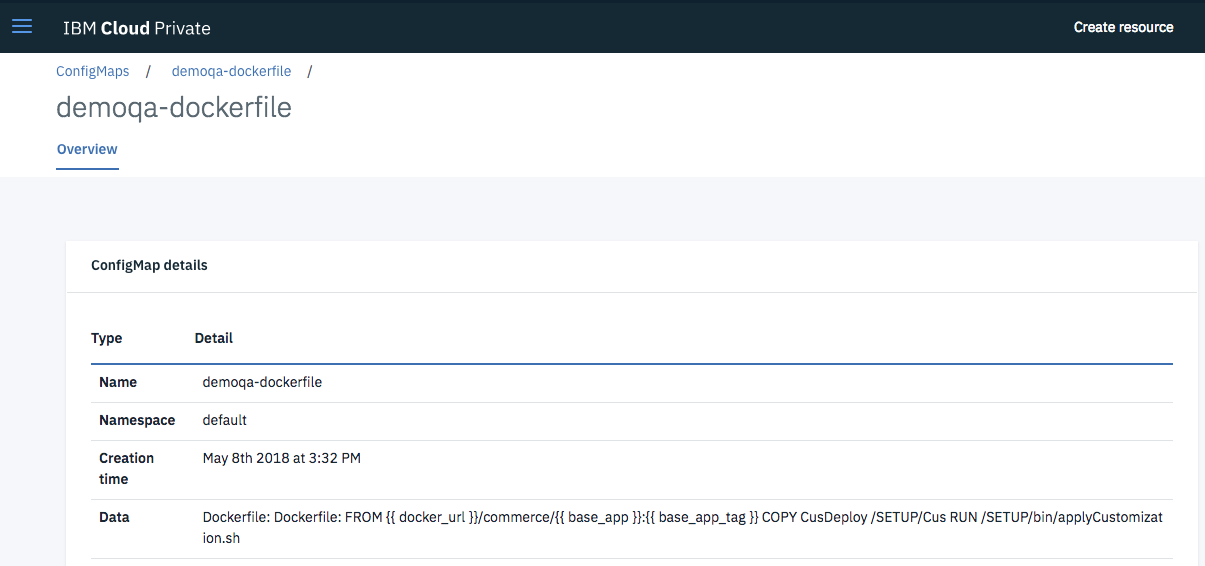
RUN /SETUP/bin/applyCustomization.sh

|  |  |
| --- | --- |
| Filed Name | Description |
| Tenant | Target tenant for build this custom Docker Image |
| EnvName ( Optional ) | Target env for build this custom Docker Image. If you not input it, This Dockerfile will use for all environment under this Tenant |
| NameSpace | Target namespace for this docker image |
| Component ( Optional ) | Target Component for this environment. This is option field. If you  not specify target component, The Dockerfile will be the common Dockerfile for all component of this environment |
| Dockerfile | This is the Dockerfile template which will used to build customization  You can keep the base image from with template mode, It will be  replaced when you trigger build. If you don’t know how to trigger it,  please keep as it |

Note: The sample template is the mini requirement. If user want to add more steps, they can add more Docker build command in it.

When this job finish. The special Dockerfile will be stored as ConfigMap on Kubernetes. You can find it this configmap object under target namespace with naming pattern.

1. Check Dockerfile on ConfigMap ( Optional )



You also can check the Dockerfile configmap by run command:

>kubectl get configmap -n default

>kubectl edit configmap <configmap\_name> -n default

1. Upload Customize Assets To Nexus

This guide assume you already finished customize source code and use WCB tool

package it, then you can using Gradle to upload this package to Nexus

On Nexus, DevOps tool chain need organize the package under group

1. Store the assets in release repository
2. Assets must under group : commerce.<tenant>
3. ArtifactId must be of them ( crs-app / ts-app / search-app / ts-web )

Here is the sample:

1. Create build gradle file with below

apply plugin: "maven"

apply plugin: "maven-publish"

publishing{

repositories {

maven {

url "http://<Nexus\_IP\_Address>:8081/nexus/content/repositories/releases"

credentials {

username ‘<nexus user name>’

password '<nexus user password>'

}

}

}

publications {

zip(MavenPublication) {

groupId 'commerce.<tenant\_name>'

artifactId '<component\_name>'

version '<valida\_version>'

artifact '<path\_of\_customization\_package>/xxx.zip'

}

}

}

Please correct all <xxxx> to appropriate value

For example, if upload crs-app custom package with

version 201812031221 for demo tenant, we can config the gradle scripts like:

…….

publications {

zip(MavenPublication) {

groupId 'commerce.demo'

artifactId 'crs-app'

version '201812031221 '

artifact '<path\_of\_customization\_package>/xxx.zip'

}

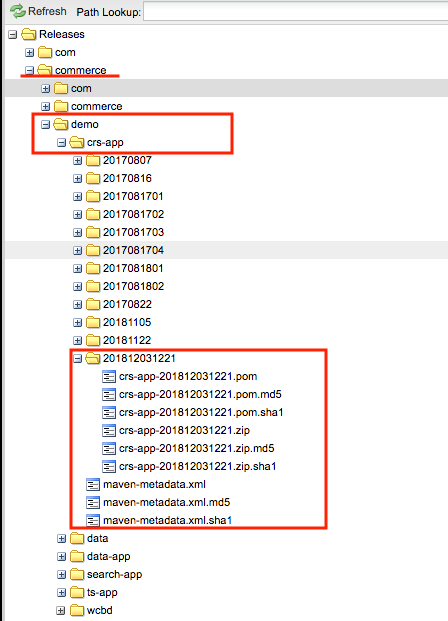
…………….

1. Run Gradle command to publish package to Nexus

#> gradle publish

Run this command on the same directory with build.gradle

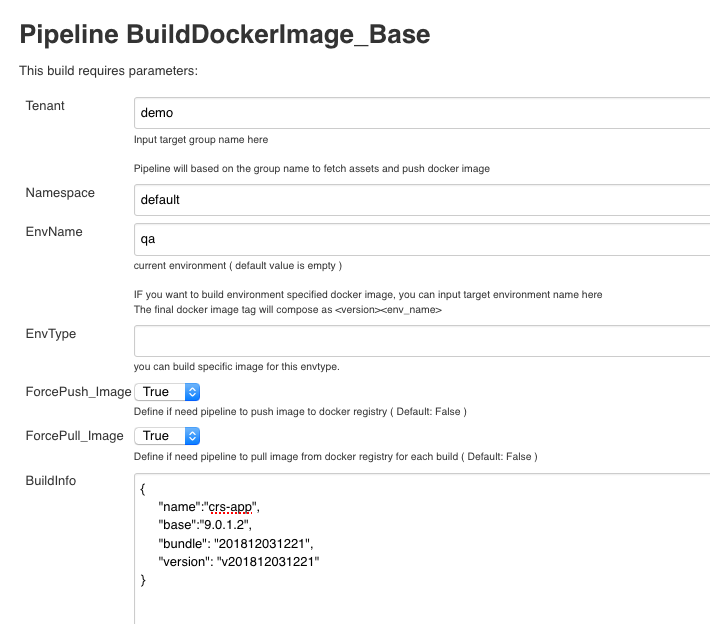
1. Optional: When gradle job finish, you can logon Nexus to check



1. Trigger Build Custom Docker Image

This guide assume will trigger build store customization docker image with custom code for demo tenant / qa environment in default namespace with custom assets version 201812031221 with Commerce OOTB docker image 9.0.1.2

1. Open Jenkins job “BuildDockerImage\_Base”
2. Fill in configuration as below:



|  |  |
| --- | --- |
| **Filed Name** | **Description** |
| Tenant | Target tenant for build |
| Namespace | Target namespace for target environment |
| EnvName | Target envname for this build, when finish  build docker, the envname will be added in  docker image tag if this filed is not empty |
| EnvType | Target envtype for this build, when finish  build docker, the envtype will be added in  docker image tag if this field is not empty |
| ForcePush\_Image | If you don’t want to push the final build  docker to docker repository, you can change  it to false |
| ForcePull\_Image | Please keep this filed as True |
| BuildInfo | This filed used to descript what component  need to build with what bundle version and  with what base OOTB docker image.  As design, it support to build multiple  component at once, but please make sure  you have extend the default docker size,  otherwise, the docker size will not enough |

BuildInfo Mapping Description

{

"name":"crs-app", ---- This is the target component name

"base":"9.0.1.2", ---- This is the target base docker image tag

"bundle": "201812031221", ---- This is customization package version. Here you

also can put a http url to point to target bundle

package

"version": "v201812031221" ---- This is the custom docker image tag

}

When you trigger build, the backend script will based on the input on this view:

1. Use namespace/tenant/env/component to find the target Dockerfile on Kubernetes
2. Use pre-defined nexus url with target tenant name / component name / bundler version to find target customization package
3. The backend scripts will use the Dockerfile and target package to build customized docker image and tag it with tenant\_name/component\_name:version+envname
4. The backend scripts will push the new custom docker image to pre-define docker-repository under tenant project

Next

With this new customized docker image, you can use “DeployWCSCloud\_Base” to upgrade existed environment

Assume you already have did initial deploy for demo/qa/auth in default namespace. You want to upgrade store with customized version

1. Open DeployWCSCloud\_Base job and Click “Build with Parameters”
2. Input Tenant / EnvName / EnvType / Namespace with demo, qa , auth, default
3. Let current filed lose focus （ for example, click any place of current view ）

Backend script will based on input the out fetch the customized helm chart value.yaml for last time deploy

1. Change “DeployAction” to “update”
2. Change docker image tag for Crsapp in “HelmChart\_Values”

………………

Crsapp:

Name: crs-app

Image: crs-app

Tag: v201812031221qa

Replica: 1

EnvParameters: {}

NodeSelector: {}

………………

**Tips : --- I have made this template change on git**

1. When you do the initial deploy, we suggest you change the default value of Common.ImageRepo from <dockerRepoHost\_value>/commerce/ to <dockerRepoHost\_value>/
2. Add commerce/ as prefix for each component image name. For example:

Crsapp:

Name: crs-app

Image: : commerce/crs-app

Tag9.0.0.4

Replica: 1

EnvParameters: {}

NodeSelector: {}

By this way, when you do the initial deploy, you can use the OOTB Commerce V9 docker image, when you do upgrade, you can just change target image name to make it point to target tenant

Crsapp:

Name: crs-app

Image: : demo/crs-app

Tag: customized-tag

Replica: 1

EnvParameters: {}

NodeSelector: {}

1. When you finish initial deploy, we recommend you use upgrade to scale out search slave / store / ts-web / xc-app / ts-app replica higher then 1. Because in Kubernetes default scale out strategy, it will start new version then, remove old version, if replica number is 1, the old version will be remove immediately. Which will cause your service offline for a while.
2. Correct target HelmChart ( this Helm Chart must exist in pre-defined Helm Reposiroty Server )
3. Optional: set “EnalbeTLS” to true if helm enabled TLS and correct “helm\_ca” / “helm\_cert” / “helm\_key”. This can be done by administrator to set those correct value as default
4. Click “Build” button to trigger upgrade