

Integración continua Front

1) Preparar ambientes en front.

Se crean 4 ambientes (environment) dentro de la MFSicof (Local, Dev, QA, Prod). Dentro de ellos se ejecutan los comandos correspondientes de ambiente:

Dev: npm run build:dev QA: npm run build:qa Prod: npm run build:prod Local es el definido por defecto si no se le coloca un ambiente y se deja tal cual:

Local: npm run start o npm start

2) Preparar Docker. Construir la imagen Docker:

- docker build --build-arg TYPE=qa -t mfsicof .

Nota: TYPE cambia en los pipelines de despliegue en Jenkins dependiendo del ambiente que apunte. Las opciones son dev, qa y prod.

Ejecutar el contenedor Docker:

- docker run -d -p 80:80 mfsicof

Nota: Se define el puerto, donde el primero es el externo y el segundo es el interno del contenedor.

Al hacer Docker Build se ejecuta la siguiente receta:

```
pipeline {
    agent any

    environment {
        DOCKER_HOST_URI = 'tcp://172.17.0.1:2375'
        GIT_REPO =
'http://10.1.140.120/ada-microservices-ecosystem/frontends/mfsicof.git'
        GIT_BRANCH = 'master'
        CREDENTIALS_ID = 'f0a2166c-1e70-47b4-9205-5284d47227f1'
        SLACK_CREDENTIALS = 'slack_secret_2'
        SLACK_CHANNEL = 'ada-ecosystem-deploy'
        SLACK_BASE_URL = 'https://slack.com/api'
        SLACK_USERNAME = 'jenkins'
    }

    stages {

        stage('Send Initial Notification') {
            steps {
                script {
                    def fullJobName = env.JOB_NAME.replaceAll('/', ' » ')
                    def buildCauseDescription =
currentBuild.getBuildCauses().first().shortDescription
                    def initialMessage = "${fullJobName} -
# ${env.BUILD_NUMBER} Started by ${buildCauseDescription} (${env.BUILD_URL})"
```

```
        slackSend(channel: env.SLACK_CHANNEL, message:
initialMessage, color: 'good', tokenCredentialId: env.SLACK_CREDENTIALS)
    }
}

stage('Checkout') {
    steps {
        script {
            checkout([
                $class: 'GitSCM',
                branches: [[name: GIT_BRANCH]],
                doGenerateSubmoduleConfigurations: false,
                extensions: [],
                submoduleCfg: [],
                userRemoteConfigs: [[
                    url: GIT_REPO,
                    credentialsId: CREDENTIALS_ID
                ]]
            ])
        }
    }
}

stage('Clean Workspace') {
    steps {
        script {
            echo "*****SE LIMPIA EL
ESPACIO DE TRABAJO*****"
        }
    }
}

stage('Set Environment Variables') {
    steps {
        script {
            echo "*****VARIABLES DE
ENTORNO*****"

            def tagName = sh(script: "jq -r '.version'
package.json", returnStdout: true).trim()
            def name = sh(script: "jq -r '.name' package.json",
returnStdout: true).trim()

            def containerName = name.replaceFirst(/^mf/, 'mf-')
            def repositoryName = "ecosystemuser/${name}"

            env.TAG_NAME = tagName
            env.NAME = name
            env.CONTAINER_NAME = containerName
            env.REPOSITORY_NAME = repositoryName
```

```

        echo "    TAG_NAME=${env.TAG_NAME}"
        echo "    NAME=${env.NAME}"
        echo "    CONTAINER_NAME=${env.CONTAINER_NAME}"
        echo "    REPOSITORY_NAME=${env.REPOSITORY_NAME}"
    }
}
stage('Docker Build & Push') {
    steps {
        script {
            echo "*****EJECUCION DE COMANDOS
DOCKER*****"
            sh """
                export DOCKER_HOST=${env.DOCKER_HOST_URI}
                docker build --no-cache --build-arg TYPE=prod -t
                ${env.REPOSITORY_NAME}:${env.TAG_NAME} .
                docker tag ${env.REPOSITORY_NAME}:${env.TAG_NAME}
                ${env.REPOSITORY_NAME}:latest
                docker push ${env.REPOSITORY_NAME}:${env.TAG_NAME}
                docker push ${env.REPOSITORY_NAME}:latest
            """
            echo "*****TERMINA EJECUCION DE COMANDOS
DOCKER*****"
        }
    }
}

stage('Deploy Docker Container') {
    steps {
        script {
            echo "*****INICIA EJECUCION Deploy Docker
Container*****"
            sh """
                export DOCKER_HOST=${env.DOCKER_HOST_URI}
                if [ \$(docker ps -aq -f name=${env.CONTAINER_NAME})
]; then
                    docker stop ${env.CONTAINER_NAME}
                    docker rm -fv ${env.CONTAINER_NAME}
                fi
                docker run -d -p 8095:80 --name
                ${env.CONTAINER_NAME} ${env.REPOSITORY_NAME}:latest
            """
            echo "*****TERMINA EJECUCION Deploy Docker
Container*****"
        }
    }
}

stage('Merge Branches') {
    steps {
        script {

```

```
def warningMessage = '
sh '''
    # Configuración de usuario git
    git config --global user.email "simon.gil@ada.co"
    git config --global user.name "simon.gil"
    GIT_USER='simon.gil'
    GIT_TOKEN='jwPyfnRVxbD2ihgVPByN'
    git config credential.helper "store --file=.git-
credentials"
    echo "http://${GIT_USER}:${GIT_TOKEN}@10.1.140.120"
> .git-credentials
    # Borrar las ramas locales develop, qa y pre-
production si existen
    delete_local_branches() {
        for BRANCH in "$@"; do
            if git show-ref --quiet refs/heads/\$BRANCH;
then
                echo "Borrando la rama local
\$BRANCH..."
                git branch -D \$BRANCH
            fi
        done
    }
    delete_local_branches develop qa pre-production
...
def branches = ['develop', 'qa', 'pre-production']
branches.each { branch ->
    def branchWarningMessage = sh(script: """
        # Función para fusionar ramas
        merge_branch() {
            BRANCH=\$1
            if git ls-remote --exit-code --heads origin
\$BRANCH >/dev/null 2>&1; then
                echo "La rama \$BRANCH existe en el
remoto. Haciendo pull y checkout..."
                git fetch origin \$BRANCH:\$BRANCH
                git checkout \$BRANCH
            else
                echo "La rama \$BRANCH no existe en el
remoto. Creando una nueva rama basada en master..."
                git checkout origin/master -b \$BRANCH
                git push origin \$BRANCH
            fi
            CURRENT_VERSION=\$(jq -r '.version'
package.json)
            echo "La versión de \$BRANCH :
\$CURRENT_VERSION"
            git checkout master
            MASTER_VERSION=\$(jq -r '.version'
package.json)
```

```

        echo "La versión de Master :
\${MASTER_VERSION}"
        if dpkg --compare-versions
"\${CURRENT_VERSION}" "le" "\${MASTER_VERSION}"; then
            echo "La versión en la rama \${BRANCH} es
igual o menor que la versión en master. Realizando merge..."
            git checkout \${BRANCH}
            git merge origin/master
            git push origin \${BRANCH}
        else
            echo "Advertencia: La versión en la rama
\${BRANCH} es superior a la versión en master. No se realizará el merge."
            return 1
        fi
        git checkout \${BRANCH}
    }
    merge_branch ${branch}
    "", returnStatus: true)

    if (branchWarningMessage != 0) {
        echo "Agrego mensaje advertencia."
        warningMessage += "Advertencia: La versión la
rama ${branch} es superior a la versión de master. No se realizará el merge
en dicha rama.\n"

        echo "Mostrar mensaje : (${warningMessage})"
    }
}
sh '''
# Limpiar credenciales
rm .git-credentials
git config --unset credential.helper
'''

    if (warningMessage) {
        echo "Warning : (${warningMessage})"
        echo "Agrego mensaje advertencia 2."
        def WAR_NAME = warningMessage
        env.WARNING_MESSAGE = WAR_NAME
        echo "Mostrar mensaje : ${env.WARNING_MESSAGE}"
    }
}
}
}

}

post {
    success {
        script {
            echo "Sending Slack notification for SUCCESS..."
            def fullJobName = env.JOB_NAME.replaceAll('/', ' » ')
            def buildStatus = currentBuild.result ?: 'UNKNOWN'
            def buildDuration = currentBuild.durationString

```

```
        def customMessage = "${fullJobName} - #${env.BUILD_NUMBER}
${buildStatus} after ${buildDuration} (${env.BUILD_URL})"
        slackSend(channel: env.SLACK_CHANNEL, message:
customMessage, color: 'good', tokenCredentialId: env.SLACK_CREDENTIALS)
        if (env.WARNING_MESSAGE) {
            slackSend(channel: env.SLACK_CHANNEL, message:
env.WARNING_MESSAGE, color: 'warning', tokenCredentialId:
env.SLACK_CREDENTIALS)
        }
    }
}
failure {
    script {
        echo "Sending Slack notification for FAILURE..."
        def fullJobName = env.JOB_NAME.replaceAll('/', ' » ')
        def buildStatus = currentBuild.result ?: 'UNKNOWN'
        def buildDuration = currentBuild.durationString
        def customMessage = "${fullJobName} - #${env.BUILD_NUMBER}
${buildStatus} after ${buildDuration} (${env.BUILD_URL})"
        slackSend(channel: env.SLACK_CHANNEL, message:
customMessage, color: 'danger', tokenCredentialId: env.SLACK_CREDENTIALS)
        if (env.WARNING_MESSAGE) {
            slackSend(channel: env.SLACK_CHANNEL, message:
env.WARNING_MESSAGE, color: 'warning', tokenCredentialId:
env.SLACK_CREDENTIALS)
        }
    }
}
unstable {
    script {
        echo "Sending Slack notification for UNSTABLE..."
        def fullJobName = env.JOB_NAME.replaceAll('/', ' » ')
        def buildStatus = currentBuild.result ?: 'UNKNOWN'
        def buildDuration = currentBuild.durationString
        def customMessage = "${fullJobName} - #${env.BUILD_NUMBER}
${buildStatus} after ${buildDuration} (${env.BUILD_URL})"
        slackSend(channel: env.SLACK_CHANNEL, message:
customMessage, color: 'warning', tokenCredentialId: env.SLACK_CREDENTIALS)
        if (env.WARNING_MESSAGE) {
            slackSend(channel: env.SLACK_CHANNEL, message:
env.WARNING_MESSAGE, color: 'warning', tokenCredentialId:
env.SLACK_CREDENTIALS)
        }
    }
}
aborted {
    script {
        echo "Sending Slack notification for ABORTED..."
        def fullJobName = env.JOB_NAME.replaceAll('/', ' » ')
        def buildStatus = currentBuild.result ?: 'UNKNOWN'
```

```

        def buildDuration = currentBuild.durationString
        def customMessage = "${fullJobName} - #${env.BUILD_NUMBER}
${buildStatus} after ${buildDuration} (${env.BUILD_URL})"
        slackSend(channel: env.SLACK_CHANNEL, message:
customMessage, color: '#439FE0', tokenCredentialId: env.SLACK_CREDENTIALS)
        if (env.WARNING_MESSAGE) {
            slackSend(channel: env.SLACK_CHANNEL, message:
env.WARNING_MESSAGE, color: 'warning', tokenCredentialId:
env.SLACK_CREDENTIALS)
        }
    }
}
}
}

```

3) Crear proyecto en GitLab.

Se crea un proyecto en GitLab con el siguiente repositorio:

<http://10.1.140.120/ada-microservices-ecosystem/frontends/mfsicof.git>

Se crea un webhook para el frontend con la siguiente URL:

Para clonar el proyecto una vez tengas permisos de Git:

code git clone <http://10.1.140.120/ada-microservices-ecosystem/frontends/mfsicof.git> Para instalar el ambiente se recomienda consultar la guía de primeros pasos.

Se recomienda consultar la guía de flujo Git.

4) Preparar Jenkins.

Adicional a la configuración de contenedores para Jenkins es necesario instalar jq. jq es una herramienta de línea de comandos para procesar JSON. Asegúrate de que esté instalada en el sistema donde Jenkins está ejecutando el script. En sistemas basados en Debian/Ubuntu, puedes instalar jq con:

1. `sudo apt-get install jq`
2. Configurar el Script en Jenkins:
3. Accede a la configuración del proyecto en Jenkins.
4. En la sección “Build”, añade o edita el paso “Execute shell”.
5. Copia y pega el script anterior en el campo de texto del shell.
6. Guardar y Probar:
7. Guarda la configuración del proyecto.

5.1 Acceder a Jenkins: Inicia sesión en tu instancia de Jenkins.

5.2 Crear un nuevo proyecto: Ve a “New Item”. Selecciona “Freestyle project” y da un nombre a tu proyecto. Haz clic en “OK”.

5.3 Configurar el repositorio Git: En la sección “Source Code Management”, selecciona “Git”. Ingresa la URL de tu repositorio GitLab y las credenciales necesarias.

5.4 Configurar el webhook de GitLab:

- En GitLab, ve a tu proyecto.
 - Navega a “Settings” > “Webhooks”.
 - Añade una nueva URL de webhook apuntando a tu Jenkins (e.g., <http://your-jenkins-url/gitlab-webhook/>). * *Selecciona los eventos que desees que disparen el webhook, como “Push events”.*
- 5.5 Añadir un paso de ejecución de shell:** En la sección “Build”, haz clic en “Add build step” y selecciona “Execute shell”. Copia y pega el siguiente script. # Definir la URI del host Docker `DOCKER_HOST_URI="tcp:172.17.0.1:2375"`

Exportar la variable DOCKER_HOST

```
export DOCKER_HOST=$DOCKER_HOST_URI
```

Extraer la versión y el nombre del archivo package.json

```
TAG_NAME=$(jq -r '.version' package.json) NAME=$(jq -r '.name' package.json)
```

Modificar el nombre para insertar un guion después de 'mf'

```
CONTAINER_NAME=$(echo $NAME | sed 's/^mf/mf-/')
```

Quitar el guion del nombre del repositorio

```
REPOSITORY_NAME="ecosystemuser/${NAME}"
```

Imprimir los nombres para visualización

```
echo "REPOSITORY_NAME: $REPOSITORY_NAME" echo "CONTAINER_NAME: $CONTAINER_NAME"
```

Construir la imagen Docker con el argumento de construcción

```
docker build --no-cache --build-arg TYPE=prod -t $REPOSITORY_NAME:$TAG_NAME .
```

Etiquetar la imagen con latest

```
docker tag $REPOSITORY_NAME:$TAG_NAME $REPOSITORY_NAME:latest
```

Empujar la imagen con la etiqueta latest a Docker Hub

```
docker push $REPOSITORY_NAME:$TAG_NAME docker push $REPOSITORY_NAME:latest
```

```
if [ "$(docker ps -aq -f name=$CONTAINER_NAME)" ]; then
```

```
    docker stop $CONTAINER_NAME
    docker rm -fv $CONTAINER_NAME
```

```
fi
```

Ejecutar el nuevo contenedor con la imagen “latest”

```
docker run -d -p 8095:80 --name $CONTAINER_NAME $REPOSITORY_NAME:latest
```

[←Regresar](#)

From:
<http://wiki.adacsc.co/> - Wiki

Permanent link:
<http://wiki.adacsc.co/doku.php?id=ada:howto:sicoferp:factory:new-migracion-sicoferp:front:iac&rev=1720103417>

Last update: **2024/07/04 14:30**

