عنوان مستند:

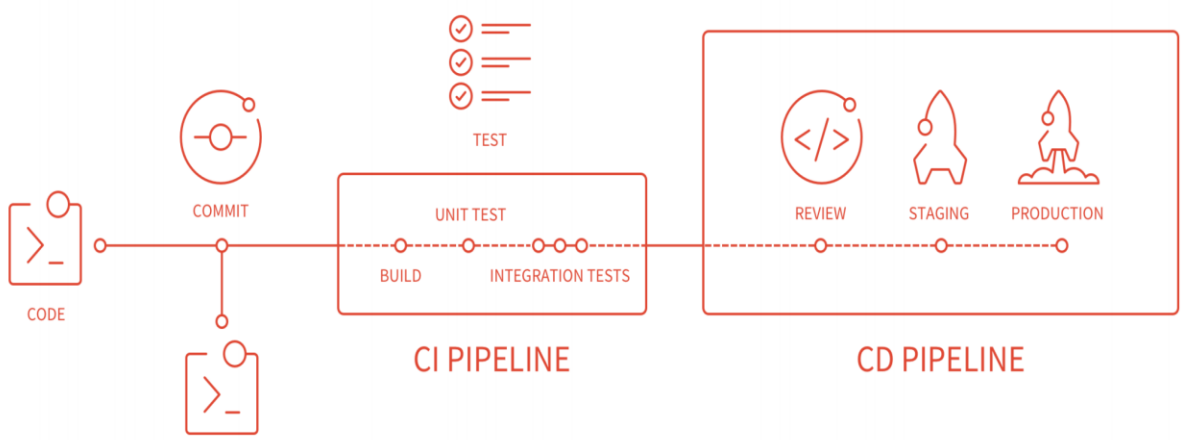
راه‌اندازی Gitlab-CICD برای پروژه‌های بهاران

بهار ۱۳۹۸

فهرست

۱ معرفی Gitlab-CICD

Gitlab قابلیت Continuous Integration و Continuous Deployment/Delivery را درون خود دارد که در راستای خودکارسازی مراحل Build، Test و Deploy پروژه های نرم افزاری می توان از آن استفاده کرد. در شکل ۱ نحوه کارکرد Gitlab-CICD نشان داده شده است، برای استفاده از این قابلیت کافیست تنظیماتی را در پروژه و در Gitlab انجام دهیم. در ادامه مراحل انجام این کار توضیح داده می شود.



شکل ۱ Gitlab-CICD

۲ نحوه استفاده از Gitlab-CICD

برای استفاده از سرویس Continuous Integration/Continuous Deployment، باید فایلی با عنوان gitlab-ci.yml. را به روت دایرکتوری پروژه اضافه کرده و تنظیماتی را در پروژه انجام دهیم تا از یک Gitlab-Runner استفاده نماید، سپس با هر commit یا push ای بر روی پروژه CICD pipeline فراخوانی می شود.

فایل gitlab-ci.yml. به Gitlab-Runner می گوید که چه کاری باید انجام دهد. بصورت پیش فرض این فایل شامل سه Stage است: Build, Test و Deploy که از بین آنها می توان Stage های مدنظر خود را انتخاب کرد تا برروی پروژه اعمال شود. همچنین می توان Stage های دلخواه خود را به آن اضافه کرد. هر

Stage شامل Job مربوط به خود است . مثلا در Stage Build درون فایل gitlab-ci.yml. ذکر می کنیم که پروژه Build شود (دستورات در Runner اجرا می شوند و پروژه نیز از منابع Runner استفاده می کند)

با توجه به اینکه پروژه های شرکت از Nexus Repository Manager استفاده می کنند علاوه بر تنظیمات Runner و gitlab-ci.yml. ، باید اطلاعات Nexus (URL, Username, Password) را نیز به پروژه اضافه نماییم.

پس از انجام این تنظیمات، با Push کردن به Git repository، Runner بصورت خودکار Pipeline را اجرا می کند که در قسمت Pipelines از پروژه می توان اجرای آن را مشاهده کرد.

۲,۱ اضافه کردن فایل **gitlab-ci.yml.**

پروژه هایی که در بهاران وجود دارند در دو دسته قرار می گیرند : ۱- اپلیکیشن ها (خود یک پروژه مستقل بوده و در نهایت run و operate می شود. ۲- پروژه های dependency: این پروژه ها در قالب dependency برای سایر پروژه ها استفاده می شوند (jar یا war فایل آن ها باید در اختیار پروژه هایی که به آنها نیاز دارند قرار گیرد)

۲,۱,۱ فایل **gitlab-ci.yml.** مربوط به اپلیکیشن های مستقل

در فایل gitlab-ci.yml. مشخص می کنیم که CI چه چیزی را روی پروژه ما انجام دهد، این فایل را باید در Root repository پروژه قرار دهیم. زمانی که push ای روی repository انجام شود، Gitlab به دنبال این فایل می گردد و job هایی که درون آن تعریف شده اند را در Runner اجرا می کند. در لینک زیر فایل gitlab-ci.yml. که برای پروژه های اپلیکیشن (Independent) شرکت نوشته شده، قابل ملاحظه است:

http://gitlab.baharan.ir:10080/snippets/17

آنچه که در این فایل نوشته ایم در زیر نشان داده شده است:

```
#replace "test" in "App_Name" with your Application Name

image: 192.168.253.10:8082/openjdk:8

variables:
```

```
  MAVEN_CLI_OPTS: "-s .m2/settings.xml --batch-mode"
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"
  App_Name: "test"

services:
  - 192.168.253.10:8082/docker:dind

stages:
  - build
  - build-image
  - deploy
  - cleanup

build:
  stage: build
  only:
    - master@GroupName/ProjectName
  script:
    - chmod a+x mvnw
    - './mvnw $MAVEN_CLI_OPTS package -Pprod -DskipTests'
  artifacts:
    expire_in: 60 mins 0 sec
    paths:
        - target

build-image:
  stage: build-image
  only:
    - master@GroupName/ProjectName
  when: manual
  before_script:
   - git clone http://192.168.251.80:10080/public-group/devops.git
   - cp -Rv devops/DevOps/src/main/docker/app-scripts src/main/docker/
   - cp -v devops/DevOps/src/main/docker/itext-4.2.0.jar target/$App_Name/WEB-INF/lib
  script:
        - cp -R target/$App_Name src/main/docker/
        - cd src/main/docker
        - docker rm -f $App_Name || true
        - docker rmi -f 192.168.253.10:8083/$App_Name:latest || true
        - docker build -t 192.168.253.10:8083/$App_Name:latest .
  tags:
    - shell

deploy:
  stage: deploy
  only:
    - master@GroupName/ProjectName
  when: manual
  script:
    - docker rm -f $App_Name || true
    - docker-compose -f src/main/docker/app-compose.yml up -d
```

```
  tags:
    - shell

cleanup_job:
  stage: cleanup
  only:
    - master@GroupName/ProjectName
  when: manual
  script:
    - echo "Cleaning up"
    - rm -rf "%CACHE_PATH%/%CI_PIPELINE_ID%"
```

نکته: *دقت داشته باشید که مقدار متغیر App_Name را باید از ''test'' به اسم اپلیکیشن خود تغییر دهید و بجای GroupName و ProjectName نیز باید اسم گروه و پروژه خود را وارد کنید.*

همانگونه که مشاهده می شود در این فایل gitlab-ci.yml.، از یک Image داکر، openjdk با ورژن ۸ استفاده شده است (پروژه از نوع maven است). همانگونه که در مستند مربوط به Nexus توضیح داده شد تنظیماتی را برروی nexus شرکت انجام داده ایم تا نقش Docker-Hub Proxy را داشته باشد. بنابراین Image مربوطه را از سرور Nexus دریافت می نماییم. (قبلا تنظیمات مربوط به لاگین کردن به Nexus Docker Hub و Nexus Docker Host را انجام داده ایم و در مستند مربوط به nexus موجود است. در اینجا نیز ماشینی که به عنوان Gitlab Runner از آن استفاده می کنیم را به Nexus Docker Hub و Nexus Docker Host لاگین کرده ایم تا Image های مورد نیاز را از سرور Nexus دریافت کند. آدرس IP که مشاهده می شود مربوط به سرور Nexus است).

Stage هایی که در این فایل تعریف کرده ایم عبارتند از:

- Build
- Build-image
- Deploy
- cleanup

در ادامه هر یک از این stage ها را توضیح می دهیم.

- **استیج Build**:

در این استیج پروژه Build می شود و war یا jar فایل مربوط به آن ساخته می شود (با توجه به آنچه که در فایل pom.xml نحوه packaging را تعریف کرده ایم). پس از build شدن پروژه تعریف کرده ایم که خروجی این استیج را به مدت ۶۰ دقیقه در حافظه gitlab آپلود نماید (تا در مراحل بعد از این فایل ها استفاده نماییم). این مرحله به صورت اتوماتیک انجام می شود.

- **استیج build-image**:

در این استیج ایمیج داکر مربوط به اپلیکیشن ساخته می شود. (با توجه به مزایایی که داکر فراهم می کند، operation پروژه ها از طریق داکر خواهد بود). این مرحله باید به صورت دستی اجرا شود.

- **استیج deploy**:

بعد از ساخت image مربوط به اپلیکیشن باید container مربوط به آن را start نماییم تا پروژه deploy شود. در این مرحله این کار را انجام می دهیم. این مرحله باید به صورت دستی اجرا شود.

- **استیج cleanup**:

در نهایت در مرحله آخر cache مربوط به Gitlab-ci از Gitlab پاک می شود. این مرحله باید به صورت دستی اجرا شود.

۲،۱،۱،۱ سایر پکیج ها و فایل های مورد استفاده در فایل ***.gitlab-ci.yml***

همانگونه که در استیج build از فایل gitlab-ci.yml. دیده می شود، ابتدا پروژه clone http://192.168.251.80:10080/public-group/devops.git می شود و برخی فایل ها از آن در پروژه کپی می شوند. این فایل ها عبارتند از :

۱- فایل ds-deploy.py :

این فایل هنگام ساختن ایمیج اپلیکیشن استفاده می شود، که در آدرس http://gitlab.baharan.ir:10080/snippets/28 این فایل وجود دارد و مقدار آن عبارت است از:

 # WRITTEN BY AMIRHOSSEIN GHORAB
 # 11/16/2016

```
# Jython
# PYCHARM

import os, sys

class Class_Datasource_Creator:

    def __init__(self, temp_domain_home, tmp_dsnumber, arr_list_dsname, arr_list_dsdbname, arr_list_dsjndiname,
                 arr_list_dsdriver, arr_list_dsurl, arr_list_dsusername, arr_list_dspassword,
                 arr_list_dstestquery, arr_list_dsmaxcapacity, arr_list_dsglobaltransaction):

        self.domain_home = temp_domain_home
        self.total_dsnumber = tmp_dsnumber
        self.list_dsname = arr_list_dsname
        self.list_dsdbname = arr_list_dsdbname
        self.list_dsjndiname = arr_list_dsjndiname
        self.list_dsdriver = arr_list_dsdriver
        self.list_dsurl = arr_list_dsurl
        self.list_dsusername = arr_list_dsusername
        self.list_dspassword = arr_list_dspassword
        self.list_dstestquery = arr_list_dstestquery
        self.list_dsmaxcapacity = arr_list_dsmaxcapacity
        self.list_dsglobaltransaction = arr_list_dsglobaltransaction
        print ('[*]Initiation Done')

    def create_datasource(self, x):
        print ("++++++++++++++++ CREATE NEW DATASOURCE
++++++++++++++++++++++++++")
        print ('list_dsname[x]', self.list_dsname[x])
        print ('list_dsdbname[x]', self.list_dsdbname[x])
        print ('list_dsjndiname[x]', self.list_dsjndiname[x])
        print ('list_dsdriver[x]', self.list_dsdriver[x])
        print ('list_dsurl[x]', self.list_dsurl[x])
        print ('list_dsusername[x]', self.list_dsusername[x])
        print ('list_dspassword[x]', self.list_dspassword[x])
        print ('list_dstestquery[x]', self.list_dstestquery[x])
        print ('list_dsmaxcapacity[x]', self.list_dsmaxcapacity[x])
        print ('list_dsglobaltransaction[x]', self.list_dsglobaltransaction[x])

        print
("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
")

        cd('/')
        create(self.list_dsname[x], 'JDBCSystemResource')
        # createJDBCSystemResource(list_dsname[x])
        cd('/JDBCSystemResource/' + self.list_dsname[x] + '/JdbcResource/' + self.list_dsname[x])
        cmo.setName(self.list_dsname[x])

        cd('/JDBCSystemResource/' + self.list_dsname[x] + '/JdbcResource/' + self.list_dsname[x])
        create('myJdbcDataSourceParams', 'JDBCDataSourceParams')
        cd('JDBCDataSourceParams/NO_NAME_0')
        set('JNDIName', self.list_dsjndiname[x])
        set('GlobalTransactionsProtocol', self.list_dsglobaltransaction[x])

        cd('/JDBCSystemResource/' + self.list_dsname[x] + '/JdbcResource/' + self.list_dsname[x])
```

```
        create('myJdbcDriverParams', 'JDBCDriverParams')
        cd('JDBCDriverParams/NO_NAME_0')
        set('DriverName', self.list_dsdriver[x])
        set('URL', self.list_dsurl[x])
        set('PasswordEncrypted', self.list_dspassword[x])
        set('UseXADataSourceInterface', 'false')

        print ('create JDBCDriverParams Properties')
        create('myProperties', 'Properties')
        cd('Properties/NO_NAME_0')
        create('user', 'Property')
        cd('Property/user')
        set('Value', self.list_dsusername[x])

        cd('../../')
        create('databaseName', 'Property')
        cd('Property/databaseName')
        set('Value', self.list_dsdbname[x])

        print ('create JDBCConnectionPoolParams')
        cd('/JDBCSystemResource/' + self.list_dsname[x] + '/JdbcResource/' + self.list_dsname[x])
        create('myJdbcConnectionPoolParams', 'JDBCConnectionPoolParams')
        cd('JDBCConnectionPoolParams/NO_NAME_0')
        set('TestTableName', 'SQL SELECT 1 FROM DUAL')
            set('CapacityIncrement',1)
        set('MaxCapacity',200)
        set('InitialCapacity',1)

        assign('JDBCSystemResource', self.list_dsname[x], 'Target', 'AdminServer')

        updateDomain()

    def create_multi_datasource(self):
        print ("")
        print ("Number Of Datasources = ", self.total_dsnumber)
        print ("")
        readDomain(self.domain_home)
        for x in range(int(self.total_dsnumber)):
            self.create_datasource(x)
        closeDomain()
        exit()

domainname = os.environ.get('DOMAIN_NAME', 'base_domain')

domainhome = os.environ.get('DOMAIN_HOME', '/u01/oracle/user_projects/domains/' + domainname)

print ("Creating Datasource")

list_dsname = ["" for x in range(int(dsnumber))]
list_dsdbname = ["" for x in range(int(dsnumber))]
list_dsjndiname = ["" for x in range(int(dsnumber))]
list_dsdriver = ["" for x in range(int(dsnumber))]
list_dsurl = ["" for x in range(int(dsnumber))]
list_dsusername = ["" for x in range(int(dsnumber))]
list_dspassword = ["" for x in range(int(dsnumber))]
list_dstestquery = ["" for x in range(int(dsnumber))]
```

```
list_dsmaxcapacity = [0 for x in range(int(dsnumber))]
list_dsglobaltransaction = ["" for x in range(int(dsnumber))]

total_dsnumber = int(dsnumber)

if total_dsnumber > 50:
   print ("Total Number is grater than 50! ERROR CODE 0")

for x in range(0, total_dsnumber):
   list_dsname[x] = str(eval('dsname_' + str(x)))
   if list_dsname[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsdbname[x] = str(eval('dsdbname_' + str(x)))
   if list_dsdbname[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsjndiname[x] = str(eval('dsjndiname_' + str(x)))
   if list_dsjndiname[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsdriver[x] = str(eval('dsdriver_' + str(x)))
   if list_dsdriver[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsurl[x] = str(eval('dsurl_' + str(x)))
   if list_dsurl[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsusername[x] = str(eval('dsusername_' + str(x)))
   if list_dsusername[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dspassword[x] = str(eval('dspassword_' + str(x)))
   if list_dspassword[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dstestquery[x] = str(eval('dstestquery_' + str(x)))
   if list_dstestquery[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsmaxcapacity[x] = str(eval('dsmaxcapacity_' + str(x)))
   if list_dsmaxcapacity[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

   list_dsglobaltransaction[x] = str(eval('dsglobaltransaction_' + str(x)))
   if list_dsglobaltransaction[x] == "":
      print ("Field Can't be empty! ERROR CODE 0")

obj_datasources_creator = Class_Datasource_Creator(domainhome, total_dsnumber, list_dsname, list_dsdbname,
                     list_dsjndiname, list_dsdriver, list_dsurl, list_dsusername,
                     list_dspassword, list_dstestquery, list_dsmaxcapacity,
                     list_dsglobaltransaction)
obj_datasources_creator.create_multi_datasource()
```

۲- فایل app-deploy.py

این فایل هنگام ساختن ایمیج اپلیکیشن استفاده می شود، که در آدرس http://gitlab.baharan.ir:10080/snippets/27 این فایل وجود دارد و مقدار آن عبارت است از:

```
# WRITTEN BY AMIRHOSSEIN GHORAB
# 10/21/2016
# Jython
# PYCHARM

import os


print "Deploying Applications",temp_appname


#---- Temp Envirnonment ----#
domainhome = os.environ.get('DOMAIN_HOME', '/u01/oracle/user_projects/domains/base_domain')
appname    = os.environ.get('APP_NAME', temp_appname)
#apppkg     = os.environ.get('APP_PKG_FILE', 'accounting.war')
appdir     = os.environ.get('APP_PKG_LOCATION', '/u01/oracle/weblogic/webapps/'+ temp_appname)

#---- Disable Clustring ----#
#cluster_name = os.environ.get("CLUSTER_NAME", "DockerCluster")

readDomain(domainhome)

cd('/')
app = create(appname, 'AppDeployment')
#app.setSourcePath(appdir + '/' + apppkg)
app.setSourcePath(appdir + '/')
app.setStagingMode('nostage')

assign('AppDeployment', appname, 'Target', 'AdminServer')

#---- Disable Clustring ----#
#assign('AppDeployment', appname, 'Target', cluster_name)


updateDomain()
closeDomain()
exit()
```

۳- پکیج itext-4.2.0.jar

این پکیج در اپلیکیشن هایی که چاپ و تهیه pdf در آن ها وجود دارد مورد استفاده قرار می گیرد و در آدرس http://gitlab.baharan.ir:10080/snippets/33 این پکیج قرار داده شده است.

۲,۱,۲ فایل **.gitlab-ci.yml** مربوط به پروژه های **Dependency**

چنانچه پروژه ای که می خواهیم چرخه CICD را برروی آن اجرا نماییم، خود به عنوان یک Dependency در سایر پروژه ها مورد استفاده قرار گیرد و operation پروژه به تنهایی معنی نداشته باشد. تنها کافی است که jar فایل یا war فایل پروژه ساخته شود و در یک repository آپلود شود تا سایر پروژه هایی که به آن نیاز دارند بتوانند دانلودش نمایند. در زیر نمونه فایلی که برای چنین پروژه هایی مورد نیاز است آورده شده است:

```
# enter your group name and project name anywhere it is required.

image: 192.168.253.10:8082/openjdk:8

variables:
  MAVEN_CLI_OPTS: "-s .m2/settings.xml --batch-mode"

stages:
  - deploy
  - cleanup

deploy:
  stage: deploy
  only:
      - master@GroupName/ProjectName
  script:
    - chmod a+x mvnw
    - './mvnw $MAVEN_CLI_OPTS deploy -DskipTests'

cleanup_job:
  stage: cleanup
  only:
    - master@GroupName/ProjectName
  script:
    - echo "Cleaning up"
    - rm -rf "%CACHE_PATH%/%CI_PIPELINE_ID%"
```

این فایل را از آدرس http://gitlab.baharan.ir:10080/snippets/32 می توانید دریافت نمایید.

Stage هایی که در این فایل تعریف کرده ایم عبارتند از:

- Deploy

- Cleanup

در ادامه هر کدام را توضیح می دهیم.

- **استیج Deploy**

چنین پروژه هایی نیازی به Operation ندارند بنابراین تنها کافیست که Jar یا war فایل (با توجه به نوع packaging) آنها را در Nexus آپلود نماییم. با دستور maven deploy ابتدا پروژه build می شود و package آن تولید می شود و با توجه به اینکه اطلاعات Repository پروژه های Maven در Nexus را در Gitlab تنظیم کرده ایم، فایل پکیج (war or jar) پروژه در Nexus شرکت در Maven-releases Repository آپلود می شود.

- **استیج cleanup**:

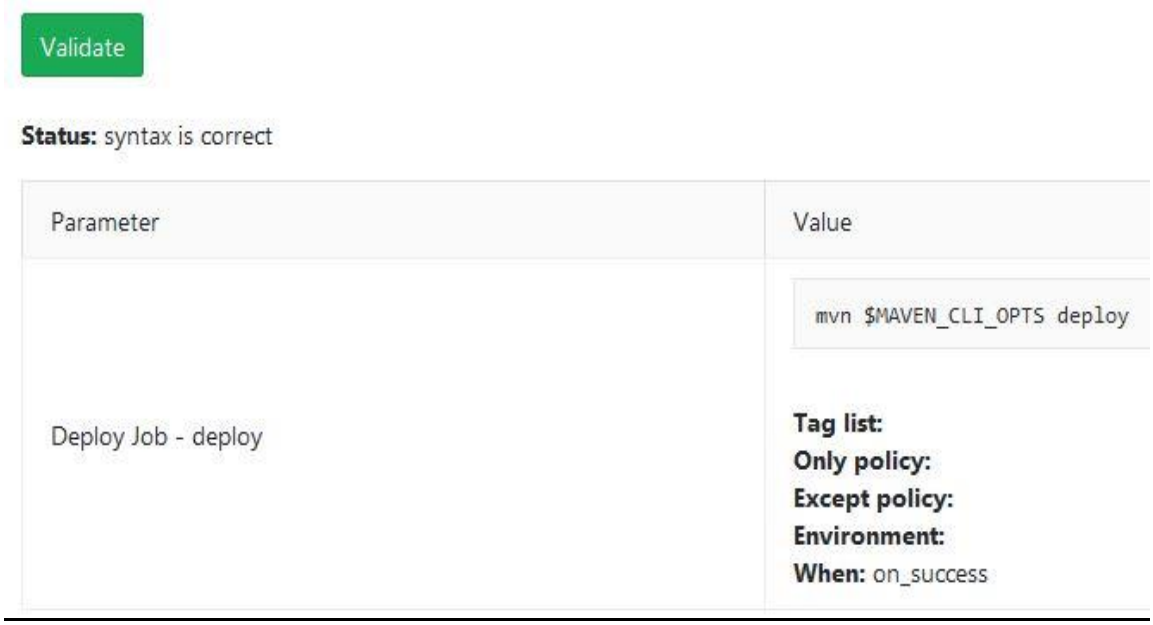
در نهایت در مرحله آخر cache مربوط به Gitlab-ci از Gitlab پاک می شود. این مرحله به صورت اتوماتیک اجرا می شود.

۲،۱،۳ تست کردن فایل **.gitlab-ci.yml**

برای اطمینان از صحیح بودن فایل gitlab-ci.yml. که نوشته ایم می توانیم از ابزار Lint در بخش ci/lint/ از namespace پروژه استفاده کرد. همچنین در بخش CI/CD ➔ Pipelines ➔ Jobs می توان "CI Lint" را یافت تا فایل را تست نماییم. سپس این فایل را در root دایرکتوری پروژه ذخیره می کنیم در شکل ۲ چک کردن یک نمونه کد gitlab-ci.yml. مشاهده می شود.

شکل ۲ تست کردن فایل gitlab-ci.yml.

خروجی که به ما می دهد در شکل ۳ نشان داده شده است که نتیجه بررسی و همچنین jobها (stageها) و دستورات هر job را به عنوان خروجی مشاهده می شود.



شکل ۳خروجی *ci-lint*

۲،۲ راه اندازی Gitlab-Runner

برای اجرای Gitlab-CI به یک ماشین نیاز است تا پردازش مورد نیاز مربوط به Stage های gitlab-ci.yml. را اجرا کند. به این ماشین Runner گفته می شود. Runner می تواند یک ماشین مجازی، سرور اختصاصی مجازی (VPS) ، کامپیوتر معمولی، یک کانتینر داکر و یا کلاستری از کانتینرها باشد. Gitlab و Runner از طریق یک API با هم در ارتباط هستند، در نتیجه تنها کافی است که ماشین Runner به سرور Gitlab دسترسی شبکه داشته باشد.

یک Runner می تواند اختصاصی و یا اشتراکی باشد، Runner اختصاصی تنها به یک پروژه تخصیص داده می شود. اما Runner اشتراکی می تواند توسط پروژه هایی که قابلیت Shared runner در آن ها تنظیم شده باشد مورد استفاده قرار گیرد. جهت کسب اطلاعات بیشتر در ارتباط با Gitlab Runner می توان به آدرس https://docs.gitlab.com/ee/ci/runners/README.html مراجعه کرد.

برای بررسی اینکه آیا Runner ای به پروژه ما اختصاص داده شده است یا خیر باید به قسمت Settings ➔ CI/CD مراجعه کرد.

جهت راه اندازی Gitlab-Runner ابتدا باید برروی ماشینی که می خواهیم Runner باشد، Gitlab-Runner را نصب نماییم، سپس باید آن را به آدرس Gitlab رجیستر نماییم.

۲,۲,۱ نصب Gitlab-Runner

Gitlab-runner را می توان برروی سیستم عامل های مختلف (لینوکس، مک، ویندوز و FreeBSD) نصب کرد. سه روش برای نصب Gitlab Runner وجود دارد: استفاده از Image داکر. دانلود فایل باینری بصورت دستی، و یا استفاده از یک Repository برای دانلود پکیج های rpm/deb. در ادامه نحوه نصب از طریق Repository برروی سیستم عامل Ubuntu توضیح داده شده است.

با استفاده از دستور زیر آدرس Repository رسمی Gitlab را به سیستم اضافه می کنیم:

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash
```

سپس با دستور زیر آخرین نسخه Gitlab-Runner را نصب می کنیم (همچنین می توان ورژن دلخواهی از آن را نصب کرد):

```
sudo apt-get install gitlab-runner
```

پس از نصب Gitlab-Runner باید آن را به Gitlab رجیستر کرد که در ادامه توضیح داده می شود.

۲,۲,۲ رجیستر کردن Gitlab-Runner

زمانی که Gitlab-Runner برروی سیستم ما نصب باشد، می توان سیستم را به Gitlab رجیستر کرد (البته باید از طریق شبکه سرور Gitlab و سروری که می خواهیم به عنوان Runner استفاده نماییم به هم متصل باشند). برای این کار دستورات زیر را اجرا می کنیم (این دستورات مربوط به رجیستر کردن Runner سیستم عامل لینوکس است)

1. `sudo gitlab-runner register`

با اجرای دستور فوق، از ما آدرس سرور Gitlab خواسته می شود. این آدرس را در پنل Gitlab می توان دید.

در ادامه در مورد انواع Runner ها توضیح داده می شود و سپس ادامه مراحل نصب بررسی می شود.

۲,۲,۲,۱ انواع Gitlab-Runner

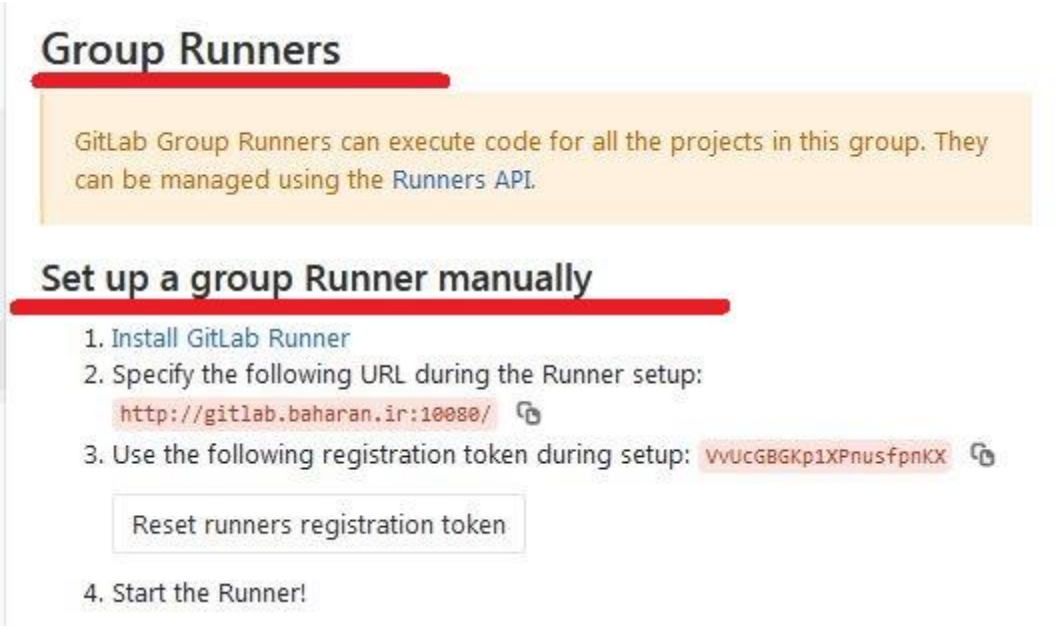
Gitlab این قابلیت را فراهم کرده است که بتوان سه نوع Runner تعریف کرد:

- Shared Runner: این نوع از Runner می تواند به تمامی پروژه هایی که در Gitlab ما وجود دارد اختصاص پیدا کند. برای تعریف چنین Runner ای به دسترسی Admin نیاز است. با انتخاب Admin Area ➔ Overview ➔ Runners پنل Runners نمایش داده می شود که می توان لیست همه Runner ها را مشاهده کرد. همچنین برای تعریف Runner جدید نیز مطابق شکل ۴ قسمتی نمایش داده می شود که آدرس URL و توکنی که برای رجیستر کردن Shared Runner نیاز است نشان داده شده است:

شکل ۴ آدرس و توکن Group Runner

- Group Runner: این نوع Runner تنها به پروژه هایی که درون آن Group هستند اختصاص داده می شود. برای تعریف این نوع Runner باید به این ترتیب عمل کرد your project ➔ Groups ➔ CI/CD ➔ Runners ➔ settings . که لیست همه Runner های آن گروه نمایش داده می شود و برای تعریف Runner جدید برای آن گروه نیز Registration token مشاهده می شود.

در شکل ۵ نحوه تعریف Group Runner نشان داده شده است.



شکل ۵ تعریف Group Runner

- Specific Runner: Runner ای است که تنها به یک پروژه اختصاص داده شده است و سایر پروژه ها نمی توانند از آن استفاده کنند. برای تعریف این نوع Runner باید به این ترتیب عمل کرد Projects ➔ your Project ➔ settings ➔ CI/CD ➔ Runners که لیست Runnerهای این پروژه و آدرس و توکن تعریف Runner جدید نمایش داده می شود.

در شکل ۵ نحوه تعریف Group Runner نشان داده شده است.

شکل ۶ تعریف Specific Runner

پس از پیدا کردن آدرس URL و Registration Token مربوط به نوع Runner ای که می خواهیم تعریف کنیم (shared, Group or Specific). آدرس را در در قسمتی که از ما می خواهد (بعد از اجرای دستور Gitlab-runner register) وارد می کنیم.

دقت داشته باشید که اگر هر دو سرور Gitlab و Runner در شبکه local وجود دارند باید بجای وارد کردن Domain Name از آدرس IP استفاده نماییم. مثلا بجای

http://gitlab.baharan.ir:10080/

باید آدرس سرور Gitlab را وارد کنیم یعنی:

http://192.168.251.80:10080/

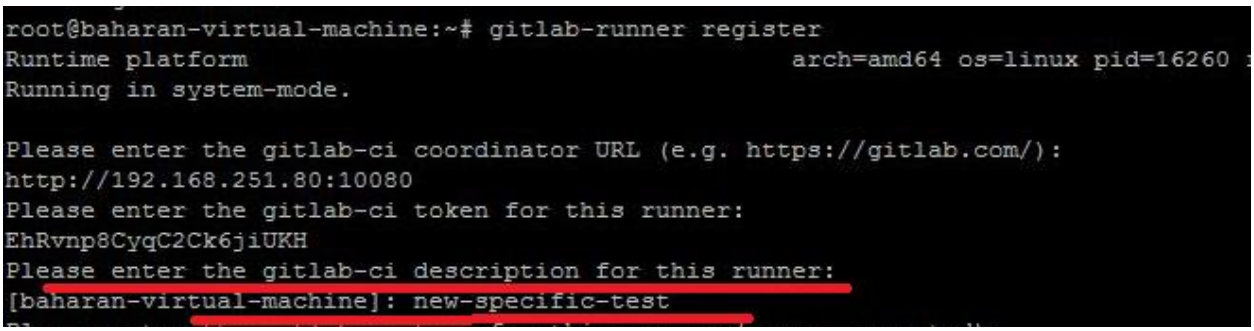
در شکل ۵ مشاهده می کنیم که بعد از وارد کردن دستور gitlab-runner register آدرس URL سرور Gitlab را می خواهد که آن را وارد کرده ایم.

شکل ۷ وارد کردن آدرس Gitlab

- پس از وارد کردن آدرس سرور یک توکن از ما می خواهد، در شکل ۴ نشان داده شد که توکن را با توجه به نوع Runner ای که می خواهیم تعریف کنیم، انتخاب می کنیم. در شکل ۸ ملاحظه می شود که بعد از وارد کردن آدرس Gitlab ، توکنی که از Gitlab گرفته ایم را وارد می کنیم.

شکل ۸ وارد کردن توکن

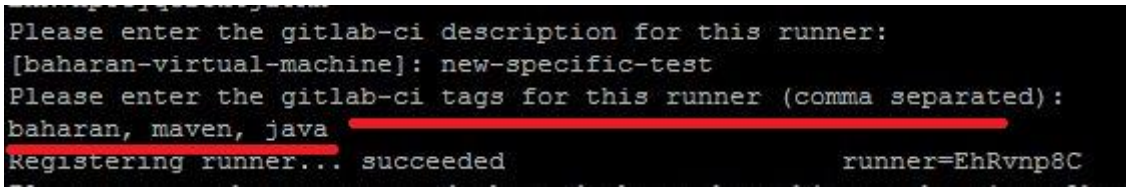
- بعد از وارد کردن توکن، یک Description از ما می خواهد، در واقع عنوانی که برای Runner می گذارد همین Description است. در شکل 9 ملاحظه می شود که به عنوان نمونه یک Description برای Runner تعریف کرده ایم.



شکل ۹ تعریف description برای Runner

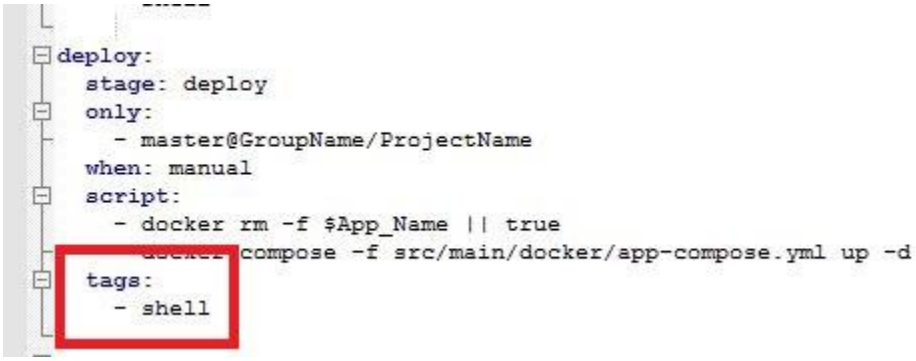
- در مرحله بعد از ما می خواهد که برای Runner تگ انتخاب نماییم (می توانیم تگ هم نگذاریم)، در شکل ۱۰ این مرحله مشاهده می شود. تگ گذاری به اختصاص Runner به استیج های .gitlab-ci.yml کمک می کند (در استیج از عبارت tags استفاده کرده و تگ Runner ای که می خواهیم آن استیج با آن Runner اجرا شود را می نویسیم در شکل ۱۱ مشاهده می شود که برای بعضی از Runner ها تگ shell گذاشته شده است، همچنین در شکل ۱۲ نیز مشاهده می شود که در فایل .gitlab-ci.yml برای یک استیج تگ shell را نوشته ایم. بعد از این، آن استیج تنها با Runner هایی اجرا می

شود که شامل این تگ باشند. همچنین در تنظیمات Runner ها در کنسول Gitlab می توانیم مشخص کنیم که آیا Runner تنها استیج های دارای تگ را اجرا کند یا سایر استیج ها را نیز اجرا کند.



شکل ۱۰ تعریف تگ برای Runner

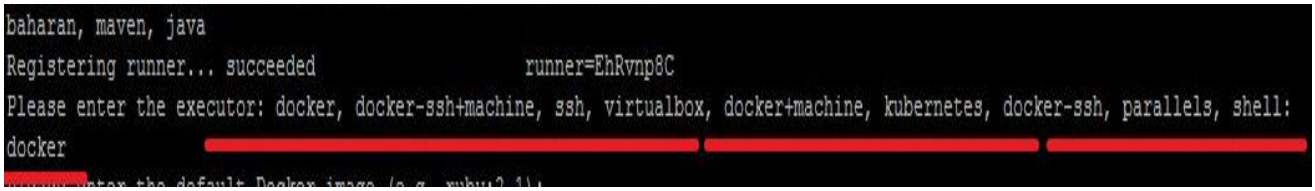
شکل ۱۱ Runner با تگ shell



شکل ۱۲ مشخص کردن تگ برای stage

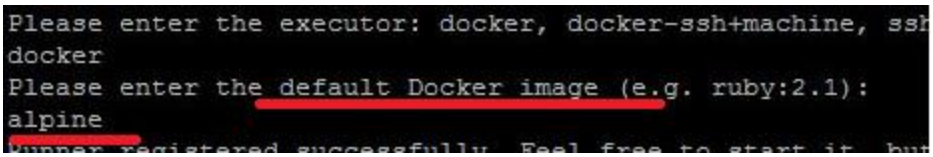
در این مرحله Runner با موفقیت رجیستر می شود.

- در مرحله بعد از ما می خواهد که یک Executor برای Runner انتخاب نماییم. Executer محیطی است که دستورات Jobهای فایل gitlab-ci.yml. را اجرا خواهد کرد. در شکل ۱۳ لیست Executorهایی که Gitlab-runner پشتیبانی می کند نشان داده شده است، برای نمونه ما executor داکر را انتخاب کرده ام تا دستوراتمان در محیط داکر اجرا شود.



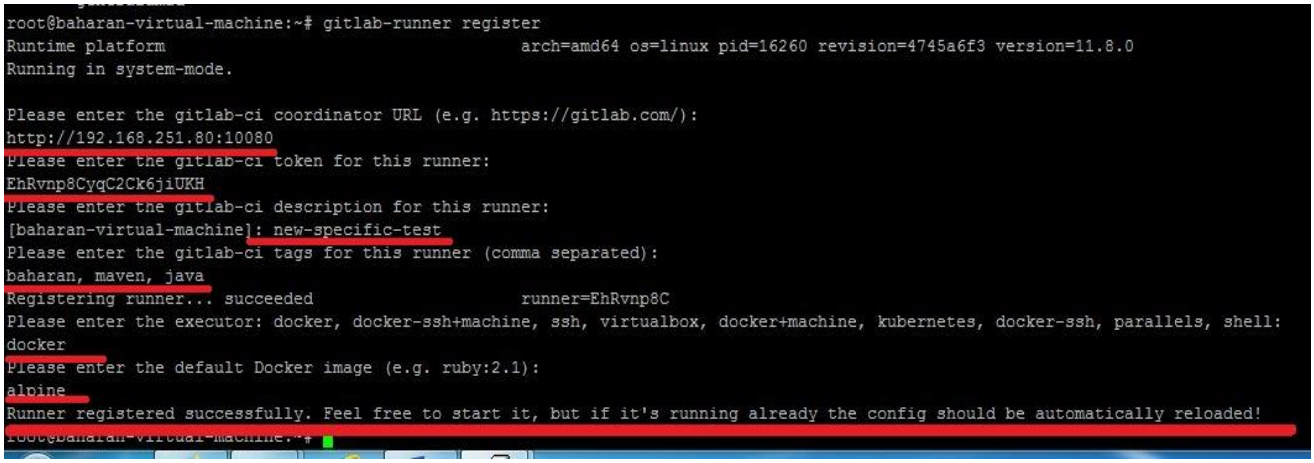
شکل ۱۳ انتخاب Executor برای Runner

- با توجه به اینکه Executor را از نوع داکر انتخاب کرده ایم از ما می خواهد که یک Image پیش فرض را برای اجرا انتخاب نماییم. اگر در استیج های فایل gitlab-ci.yml. ایمیجی مشخص نکنیم زمان اجرا داکر از این ایمیج پیش فرض استفاده می کند و دستورات در محیط آن اجرا خواهند شد. در شکل ۱۴ ملاحظه می شود که ما ایمیج alpine را انتخاب کرده ایم.



شکل ۱۴ مشخص کردن ایمیج پیش فرض برای Runner

بعد از مشخص کردن ایمیج پیش فرض، رجیستر کردن runner به اتمام می رسد و در نهایت می توانیم runner را در gitlab مشاهده نماییم admin area → runners . در شکل های ۱۵ و ۱۶ runner ای که تعریف کرده ایم قابل مشاهده است.



شکل ۱۵ مراحل رجیستر کردن Runner

شکل ۱۶ مشاهده Runner در Gitlab

- **مشاهده Runnerهای رجیستر شده در سرور**

برای مشاهده Runnerهایی که در یک سرور رجیستر کرده ایم می توانیم از دستور gitlab-runner list استفاده نماییم. برای مثال این دستور را در سرور (192.168.253.75) CICD اجرا می کنیم و نتیجه را در شکل ۱۷ مشاهده می کنید.

```
root@baharan-virtual-machine:~# gitlab-runner list
Runtime platform                    arch=amd64 os=linux pid=5939 revision=4745a6f3 version=11.8.0
Listing configured runners          ConfigFile=/etc/gitlab-runner/config.toml
runner-baharan-docker-1             Executor=docker Token=1532c2661fb7c7646096e912aaf6d8 URL=http://192.168.251.80:10080
runner-baharan-docker-2             Executor=docker Token=cf0c37cc1f19c65f308de82b71d731 URL=http://192.168.251.80:10080
runner-baharan-docker-3             Executor=docker Token=6d4281074dc8488189cd66ba915e29 URL=http://192.168.251.80:10080
runner-baharan-docker-4             Executor=docker Token=1a7ea70713a00703e277fd5441daa5 URL=http://192.168.251.80:10080
runner-baharan-docker-5             Executor=docker Token=d7e10e6a102875b444bf57b2dec46f URL=http://192.168.251.80:10080
runner-baharan-docker-6             Executor=docker Token=a9107858f1fcebe3d1daf323864e72 URL=http://192.168.251.80:10080
runner-baharan-docker-7             Executor=docker Token=4f58413263344a72f5b046a4d8516b URL=http://192.168.251.80:10080
runner-baharan-docker-8             Executor=docker Token=59718df7e46ea20b3c46e76cced63f URL=http://192.168.251.80:10080
runner-baharan-docker-9             Executor=docker Token=36f4d2ca52b34142192ae16e3d98a5 URL=http://192.168.251.80:10080
runner-baharan-docker-10            Executor=docker Token=6145dbbcdf5c4a062f17f1a5d50178 URL=http://192.168.251.80:10080
runner-baharan-shell-1              Executor=shell Token=a01242b9e79d946dae236098797728 URL=http://192.168.251.80:10080
runner-baharan-shell-2              Executor=shell Token=f4f1a112c81020c065d9aee6d1b291 URL=http://192.168.251.80:10080
runner-baharan-shell-3              Executor=shell Token=317b1eb0ec4409434b5d2865abddbe URL=http://192.168.251.80:10080
runner-baharan-shell-4              Executor=shell Token=e82a8f8483f30c250ba7d38a4328e4 URL=http://192.168.251.80:10080
runner-baharan-shell-5              Executor=shell Token=328ff4653cac5a68ef4de2716ec7ae URL=http://192.168.251.80:10080
new                                 Executor=docker Token=6fbda6111197dafef801bc742090af URL=http://192.168.20.80:10080/
mm-hfz                              Executor=docker Token=bd62b540b896e6db9978cbf5302064 URL=http://192.168.251.80:10080/
new-specific-test                   Executor=docker Token=2ad95af1a48f296e25ee7ace27300b URL=http://192.168.251.80:10080
root@baharan-virtual-machine:~#
```

شکل ۱۷ لیست Runner های سرور ۷۵

۲,۲,۲,۲ فایل Runners Configuration

در سروری که Runner روی آن تعریف می کنیم یک فایل تنظیمات مربوط به Runnerهای تعریف شده روی سرور در دایرکتوری /etc/gitlab-runner/config.toml وجود دارد در این فایل مقادیر تنظیمات که برای هر کدام از Runner ها مشخص کرده ایم نشان داده می شود. این تنظیمات هنگام رجیستر کردن Runnerها بصورت اتوماتیک تولید می شود. ولی ممکن است بعضی از تنظیمات را بعدا تغییر دهیم و یا ویژگی های جدیدی به این فایل اضافه نماییم. مثلا در زیر بخشی از فایل تنظیمات مربوط به gitlab-runner سرور cicd نشان داده شده است. مواردی که از فایل پیش فرض تغییر داده شده اند و یا به آن اضافه شده اند عباتند از concurrent=15 به این معنی که همزمان ۱۵ عدد Runner بتوانند در سرور کار کنند (به صورت پیش فرض این مقدار برابر با ۱ است)

Output_limit=40960 این ویژگی به صورت پیش فرض ذکر نشده است ولی مقدار پیش فرض 4096 کیلوبایت است، این ویژگی محدودیت حجم لاگی را مشخص می کند که در PipeLine در داخل Gitlab هنگام اجرای Stage ها نشان داده می شود. (برای پروژه هایی که لاگ فراوانی در خروجی دارند ، مقدار پیش فرض کم بوده و در نتیجه Stage با خطا مواجه می شود. بنابراین آن را ۱۰ برابر کرده ایم). و همچنین در Runner های داکری مقدار privileged را برابر true قرار می دهیم. نهایتا disable_cache را از false به true تغییر می دهیم.

```
concurrent = 15
check_interval = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "runner-baharan-docker-1"
  output_limit = 40960
  url = "http://192.168.251.80:10080"
  token = "1532c2661fb7c7646096e912aaf6d8"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "alpine"
    privileged = true
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = true
    volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
    cache_dir = "cache"
    shm_size = 0
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]

[[runners]]
  name = "runner-baharan-docker-2"
  output_limit = 40960
  url = "http://192.168.251.80:10080"
  token = "cf0c37cc1f19c65f308de82b71d731"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "alpine"
    privileged = true
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = true
    volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
    cache_dir = "cache"
    shm_size = 0
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]

[[runners]]
  name = "runner-baharan-shell-4"
  output_limit = 40960
  url = "http://192.168.251.80:10080"
  token = "e82a8f8483f30c250ba7d38a4328e4"
  executor = "shell"
  [runners.cache]
```

```
  [runners.cache.s3]
  [runners.cache.gcs]

[[runners]]
 name = "runner-baharan-shell-5"
 output_limit = 40960
 url = "http://192.168.251.80:10080"
 token = "328ff4653cac5a68ef4de2716ec7ae"
 executor = "shell"
 [runners.cache]
  [runners.cache.s3]
  [runners.cache.gcs]
```

در ادامه تغییراتی که در پروژه های Maven باید اعمال شوند تا بتوان از Gitlab-CICD در آنها استفاده کرد را توضیح می دهیم.

۲,۳ تنظیمات مربوط به پروژه های **Maven**

با توجه به اینکه پروژه های بهاران Maven ای هستند بنابراین برای کار با آنها باید از دستورات maven استفاده نماییم (maven package، maven deploy، ...) در این راستا اگر در Runner داکری که به Gitlab رجیستر کرده ایم از ایمیج maven استفاده نماییم (در خط اول فایل gitlab-ci.yml. باید ذکر کنیم image: 192.168.253.10:8082/maven:latest در نتیجه Runnerهای داکری که استیج های فایل .gitlab-ci.yml را اجرا می کنند، از این image استفاده می کنند و می توانند دستورات maven ای را اجرا کنند). اما با توجه به اینکه علاوه بر دستورات maven، دستورات دیگری نیز در استیج های gitlab-ci. قرار است انجام شوند (cp، cd، mkdir و ...) و ایمیج maven چنین دستوراتی را نمی تواند اجرا نماید، بنابراین base image را در فایل gitlab-ci. بجای maven از نوع openjdk قرار می دهیم (image: 192.168.253.10:8082/openjdk:8). و برای اجرای دستورات maven در jdk می توانیم از maven wrapper استفاده نماییم. برای استفاده از maven wrapper باید در پروژه تغییراتی اعمال نماییم که در ادامه این تغییرات را توضیح می دهیم.

۲,۳,۱ اضافه کردن **maven wrapper** به پروژه

برای استفاده از maven wrapper باید از یک قالب مشخصی پیروی نماییم و در این راستا چند فایل را باید در دایرکتوری های مشخصی به پروژه اضافه نماییم.

۱- اضافه کردن فایل mvnw به Root Directory پروژه:

این فایل را می توان از اینترنت و یا از پروژه های اوپن سورس مانند jhipster دانلود نمود. همچنین در آدرس http://gitlab.baharan.ir:10080/snippets/20 می توانید آن را دریافت نمایید.

۲- اضافه کردن فایل mvnw.cmd به Root Directory پروژه:

این فایل را می توان از اینترنت و یا از پروژه های اوپن سورس مانند jhipster دانلود نمود. همچنین در آدرس http://gitlab.baharan.ir:10080/snippets/21 می توانید آن را دریافت نمایید.

۳- اضافه کردن دایرکتوری ".mvn" به Root Directory

در روت پروژه باید این فولدر را ایجاد نماییم که در ادامه فایل هایی را به آن اضافه می کنیم.

۴- اضافه کردن دایرکتوری wrapper به دایرکتوری .mvn

بعد از ساختن دایرکنوری .mvn باید دایرکتوری wrapper را به آن اضافه نماییم.

۵- اضافه کردن فایل MavenWrapperDownloader.java به دایرکتوری .mvn/wrapper

این فایل را می توان از اینترنت و یا از پروژه های اوپن سورس مانند jhipster دانلود نمود. همچنین در آدرس http://gitlab.baharan.ir:10080/snippets/22 می توانید آن را دریافت نمایید.

۶- اضافه کردن فایل maven-wrapper.properties به دایرکتوری .mvn/wrapper

این فایل را می توان از اینترنت و یا از پروژه های اوپن سورس مانند jhipster دانلود نمود. همچنین در آدرس http://gitlab.baharan.ir:10080/snippets/23 می توانید آن را دریافت نمایید. این فایل در اصل شامل دستور زیر است:

distributionUrl=https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.6.0/apache-maven-3.6.0-bin.zip

همانطور که مشخص است این دستور شامل آدرس URL یک بسته است. اما از آنجایی که سرور های ما به اینترنت متصل نیستند، اگر با همین مقدار pipeline را اجرا نماییم با خطا مواجه خواهیم شد. برای حل این مشکل از Nexus کمک گرفته و یک Repository پروکسی را در آن ایجاد می کنیم و بجای آدرس URL فوق، آدرس Repository در Nexus را وارد می کنیم یعنی مقدار زیر را باید در این فایل بگذاریم:

distributionUrl=http://192.168.253.10:8081/repository/maven-runner/3.6.0/apache-maven-3.6.0-bin.zip

در نتیجه برای دانلود فایل مورد نیاز به maven-runner repository از nexus ارجاع داده خواهد شد و خطا برطرف می گردد.

۷- اضافه کردن پکیج maven-wrapper.jar به دایرکتوری .mvn/wrapper

این فایل را می توان از اینترنت و یا از پروژه های اوپن سورس مانند jhipster دانلود نمود. همچنین در آدرس http://gitlab.baharan.ir:10080/snippets/24 می توانید آن را دریافت نمایید.

۲,۴ تنظیمات مربوط به Nexus

با توجه به اینکه می خواهیم پروژه به Nexus Repository متصل باشد و Dependency های مورد نیاز را از آن دانلود نماید، همچنین اگر لازم شد Image مربوط به اپلیکیشن ها در Nexus ، Push شود. و همچنین برخی از اپلیکیشن ها خود Dependency برای اپلیکیشن های دیگر هستند و باید jar یا war فایل مربوط به آن ها ساخته شود و در Nexus ، Deploy شود، باید تنظیمات مربوط به دسترسی به Nexus را انجام دهیم.

۱. اضافه کردن دایرکتوری ".m2" به root directory پروژه
۲. اضافه کردن فایل settings.xml به دایرکتوری .m2

در فایل m2/settings.xml. اطلاعات دسترسی به nexus را وارد می نماییم (با توجه به اینکه پروژه از نوع Maven است باید فولدر m2. را در Root directory آن ساخته و فایل Settings.xml را درون آن قرار می دهیم). در آدرس زیر یک فایل نمونه settings.xml مشاهده می شود.

http://gitlab.baharan.ir:10080/snippets/18

آنچه که در این فایل می نویسیم، اطلاعات دسترسی به Nexus است. در زیر محتوای این فایل که از آن استفاده می کنیم آورده شده است (دقت کنید که بجای @@@@@ باید یوزرنیم و پسورد Nexus را وارد نمایید):

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
              http://maven.apache.org/xsd/settings-1.0.0.xsd">
        <mirrors>
                <mirror>
                        <!--This sends everything else to /public -->
                        <id>nexus</id>
                        <mirrorOf>*</mirrorOf>
                                <url>http://192.168.253.10:8081/repository/maven-public/</url>
                </mirror>
        </mirrors>
        <profiles>
                <profile>
                        <id>nexus</id>
                        <!--Enable snapshots for the built in central repo to direct -->
                        <!--all requests to nexus via the mirror -->
                        <repositories>
                                <repository>
                                        <id>central</id>
                                        <url>http://central</url>
                                        <releases>
                                                <enabled>true</enabled>
                                                <updatePolicy>never</updatePolicy>
                                        </releases>
                                        <snapshots>
                                                <enabled>true</enabled>
                                                <updatePolicy>never</updatePolicy>
                                        </snapshots>
                                </repository>
                        </repositories>
                        <pluginRepositories>
                                <pluginRepository>
                                        <id>central</id>
                                        <url>http://central</url>
```

```
                <releases>
                    <enabled>true</enabled>
                    <updatePolicy>never</updatePolicy>
                </releases>
                <snapshots>
                    <enabled>true</enabled>
                    <updatePolicy>never</updatePolicy>
                </snapshots>
            </pluginRepository>
        </pluginRepositories>
    </profile>
</profiles>

<activeProfiles>
    <!--make the profile active all the time -->
    <activeProfile>nexus</activeProfile>
</activeProfiles>

<servers>
    <server>
        <id>nexus</id>
        <username>@@@@@</username>
<password>@@@@@</password>
    </server>
    <server>
        <id>releases</id>
        <username>@@@@@</username>
<password>@@@@@</password>
    </server>
    <server>
        <id>snapshots</id>
        <username>@@@@@</username>
<password>@@@@@</password>
    </server>
</servers>
</settings>
```

۳. اضافه کردن <distributionManagement> به فایل pom.xml از پروژه

در این تگ که به فایل pom.xml اضافه می کنیم در واقع آدرس nexus repository را جهت deploy کردن اپلیکشن وارد می نماییم. محتوایی که اضافه می کنیم در آدرس زیر در دسترس است:

http://gitlab.baharan.ir:10080/snippets/19

مقداری که به این فایل اضافه می کنیم عبارت است از:

```
  <distributionManagement>
    <repository>
```

```
        <id>releases</id>
        <name>releases</name>
        <url>http://192.168.253.10:8081/repository/maven-releases/</url>
    </repository>
  </distributionManagement>
```

۲,۵ تنظیمات مربوط به Docker

در چرخه CICD قرار است که از مزایای داکر استفاده نماییم و از اپلیکیشن های مستقل (Independent) ایمیج داکری ساخته شده و سپس Container آنها برروی سرور CICD (192.168.253.75) اجرا خواهد شد. بنابراین برای ساختن ایمیج اپلیکیشن ها و همچنین اجرای Container آنها باید تغییراتی را در پروژه اعمال نماییم که در ادامه توضیح می دهیم.

۱- اضافه کردن دایرکتوری docker به دایرکتوری src/main

۲- اضافه کردن فایل Dockerfile به دایرکتوری src/main/docker

همانگونه که در چرخه CICD قبلی شرکت که از Jenkins استفاده می شد، ایمیج داکری اپلیکیشن ساخته می شد (که به Dockerfile نیاز هست) ، در اینجا نیز برای ساخت ایمیج اپلیکیشن نیز به فایل Dockerfile نیاز است. بنابراین با توجه به اینکه در چرخه قدیمی نیز این فایل وجود داشته است نیازی به تغییر وجود ندارد، در آدرس http://gitlab.baharan.ir:10080/snippets/25 یک نمونه از این فایل وجود دارد که در زیر نیز نشان داده شده است:

```
#=============================================================
#
# WRITTEN BY AMIRHOSSEIN GHORAB
# 10/21/2016
# Dockerfile
#=============================================================
#
# BUILD COMMAND ==> sudo docker build -t app-name:v1 .
# RUN COMMAND (background mode with -d flag enable)  ==> sudo docker run -d -p 8001:8001 app-name:v1
#=============================================================
#
```

```
#CHANGE " ENV APP_NAME="legal" " AND " COPY legal /u01/oracle/weblogic/webapps/ " FOR
YOR APPLICATION
#============================================================
#

# Pull base image (weblogic 12.2.1 Pre Build Domain)
FROM ahg-img-weblogic:v3

# Maintainer Me !!
MAINTAINER AMIRHOSSEIN GHORAB <he.amirhossein@gmail.com>

#APP_NAME=accounting
#APP_NAME=amadProject
#APP_NAME=shoppingOrganization
#APP_NAME=ammunitionWeapon
#APP_NAME=bazresiProject
#APP_NAME=calendar
#APP_NAME=evaluation
#APP_NAME=hrmall
#APP_NAME=transportation
#APP_NAME=university-testing-system
#APP_NAME=vaghef
#APP_NAME=valifaghihProject
#APP_NAME=tms

# Define variables
ENV APP_NAME="legal" \
    APP_PKG_LOCATION="/u01/oracle/weblogic/webapps/"

RUN mkdir -p /u01/oracle/weblogic/webapps

# Copy Scripts
COPY app-scripts/* /u01/oracle/weblogic/webapps/
COPY docker.properties /u01/oracle/weblogic/webapps/

# Copy application files
COPY legal /u01/oracle/weblogic/webapps/

# Running Weblogic Scripting Tool (WLST)
# First deploy an application and then config datasource
RUN wlst.sh -loadProperties /u01/oracle/weblogic/webapps/docker.properties
/u01/oracle/weblogic/webapps/app-deploy.py && \
    wlst.sh -loadProperties /u01/oracle/weblogic/webapps/docker.properties
/u01/oracle/weblogic/webapps/ds-deploy.py
```

توجه داشته باشید که با توجه به هر اپلیکیشن تغییرات مختص به آن باید در این فایل اعمال شود و کد فوق مختص به یک اپلیکیشن خاص است.

۳- اضافه کردن فایل docker.properties به دایرکتوری src/main/docker

با توجه به اینکه اپلیکیشن های شرکت با weblogic application server کار می کنند، هنگام ساخت ایمیج از اپلیکیشن ها نیاز به ساخت data source است که در این فایل آن را تعریف می نماییم (در چرخه Jenkins نیز از این فایل استفاده می شده است بنابراین نیازی به تغییر در این فایل وجود ندارد). در آدرس http://gitlab.baharan.ir:10080/snippets/26 یک نمونه از این فایل وجود دارد که در زیر نیز نشان داده شده است:

```
# WRITTEN BY AMIRHOSSEIN GHORAB
# 10/21/2016
# Jython
# PYCHARM
#------ DEPLOYMENT PROPERTIES ----------#

temp_appname=legal

#------ Number Of DataSources ----------#

dsnumber=2

#------ DATASOURCE 1 PROPERTIES ----------#

dsname_0=develop
dsdbname_0=orcl;create=true
dsjndiname_0=develop
dsdriver_0=oracle.jdbc.xa.client.OracleXADataSource
dsurl_0=jdbc:oracle:thin:@192.168.251.141:1522:orcl
dsusername_0=lgl
dspassword_0=lgl
dstestquery_0=SQL SELECT 1 FROM SYS.SYSTABLES
dsmaxcapacity_0=1
dsglobaltransaction_0=EmulateTwoPhaseCommit

#------ DATASOURCE 2 PROPERTIES ----------#

dsname_1=activiti
dsdbname_1=orcl;create=true
dsjndiname_1=activiti
dsdriver_1=oracle.jdbc.OracleDriver
dsurl_1=jdbc:oracle:thin:@192.168.251.141:1522:orcl
dsusername_1=lgl
dspassword_1=lgl
dstestquery_1=SQL SELECT 1 FROM SYS.SYSTABLES
dsmaxcapacity_1=1
dsglobaltransaction_1=EmulateTwoPhaseCommit
```

توجه داشته باشید که با توجه به هر اپلیکیشن تغییرات مختص به آن باید در این فایل اعمال شود و کد فوق مختص به یک اپلیکیشن خاص است.

۴- اضافه کردن فایل app-compose به دایرکتوری src/main/docker

جهت اجرا کردن Image اپلیکیشن برروی سرور از docker-compose استفاده می کنیم که برای این کار به یک فایل docker-compose.yml نیاز است که مشخصات container را باید به آن بدهیم. در آدرس http://gitlab.baharan.ir:10080/snippets/29 یک نمونه از این فایل به اسم app-compose.yml وجود دارد. که محتوای آن در در زیر آورده شده است:

```
#replace "test" with your application name
#change port 8080 with desired port, maybe another application is running on 8080


version: "2.4"
services:
 test:
  image: 192.168.253.10:8083/test:latest
  container_name: test
  mem_limit: 6G
  mem_reservation: 6G
  ports:
   - "8080:8001"
  environment:
   EUREKA_PORT: 8001
   EXPOSE_PORT: 8080
   TZ: "Asia/Tehran"
  extra_hosts:
   - "localhost:192.168.253.75"
  volumes:
   - "/etc/localtime:/etc/localtime:ro"
```

توجه داشته باشید که باید بجای ”test“ اسم اپلیکیشن گذاشته شود و بجای پورت "8080" شماره پورتی که می خواهیم اپلیکیشن روی آن کار کند را باید بنویسیم.