MAVEN

Build tools

make --> for C/C++ source code. make uses makefile. gcc compiler for c and g++ for c++

maven or ANT ---> java source code. maven uses pom.xml and ANT uses build.xml

Build tool maven:

Maven is a tool used to create artifacts/ binaries from code.

command : mvn clean install

pom.xml file:

POM is an acronym for Project Object Model. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc

Maven reads the pom.xml file, then executes the goal.

Elements of maven pom.xml file

For creating the simple pom.xml file, you need to have following elements:

Element	Description
project	It is the root element of pom.xml file.
modelVersion	It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0.
groupId	It is the sub element of project. It specifies the id for the project group.
artifactId	It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.
version	It is the sub element of project. It specifies the version of the artifact under given group.

Maven pom.xml file with additional elements

Here, we are going to add other elements in pom.xml file such as:

Element	Description
packaging	defines packaging type such as jar, war etc.
name	defines name of the maven project.
url	defines url of the project.
dependencies	defines dependencies for this project.
dependency	defines a dependency. It is used inside dependencies.
scope	defines scope for this maven project. It can be compile, provided, runtime, test and system.

Settings.xml - file which allows us to specify which local and remote repositories maven will use. We can also use it to store settings that we don't want in our source code, such as credentials.

A settings.xml file is usually found in a couple of places:

1. Global settings in Mavens home directory: ${maven.home}/conf/settings.xml

2. User settings in the user's home: ${user.home}/.m2/settings.xml

If both files exist, their contents are merged. Configurations from the user settings take precedence.

Tomcat : web-server

it is used to host our web application.

deploy : in simple words , copy the binaries to tomcat/webapps path

build ---> its executable or binary before testing

release----> tested build which is ready to release to customer.

patch build or hot fix : it's critical fix which needs to be delivered to customer within few hours.

we change only required files and those files will get recompiled, generate new build. so it takes less time as it compiles required files only

load build or full build: compile source code from scratch. Delete all intermediate files and compile all files. it takes more time as it compiles all

BVT or sanity test: it's basic functionality of build. it should never break

release note: release has tagname and known issues.

Deployment flow:

1. clone

clone the code from github

2. build

.war will be created in /target folder

command to build java project : mvn clean install (path where pom.xml)

3. deploy

copy the generated binaries to tomcat/webapps

verify the website by IP:8080

Assignment :

1) install tomcat + maven in 3 new instances (Amazon linux 2) (ssh anywhere , all traffic anywhere)

2) Clone from https://github.com/devopsacademy2015/my-sample-website.git .... build and deploy

3) Create pictorial diagram for deployment flo0077

Environment overview:

environment: set of servers , where the application is running

Testing environment ===> set of servers , where application is running and it is tested for issues. this is only accessible by testing team and not exposed to customers.

production environment ===> set of servers , where application is running and this application is accessible by end customers

RELEASE --> deployment to production environment.

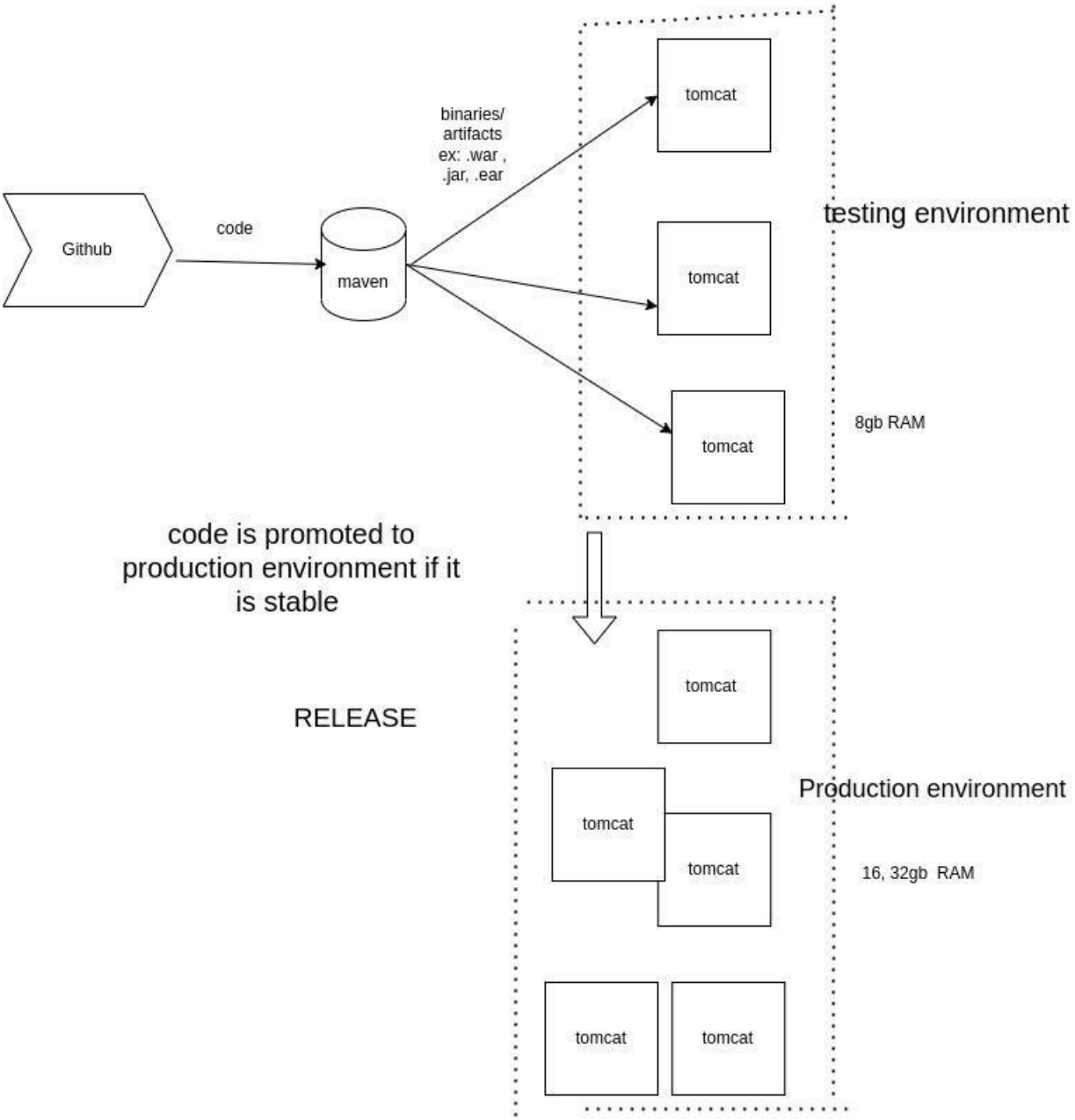
The code will be tested in testing env before RELEASING it to production.

ROLLBACK --> deploying the previous stable version. (reverting back to previous stable version)

ex: if there is critical bug or issue in logic , there will be no time to fix. Immediately we have to rollback

bug --> issue in production environment.

Representational picture of code flow



Maven Build Lifecycle

The Maven build follows a specific lifecycle to deploy and distribute the target project.
There are three built-in lifecycles:

- default: the main lifecycle, as it's responsible for project deployment
- clean: to clean the project and remove all files generated by the previous build
- site: to create the project's site documentation

Maven build Lifecycle

- *validate:* check if all information necessary for the build is available
- *compile:* compile the source code
- *test:* run unit tests
- *package:* package compiled source code into the distributable format (jar, war, …)

- *verify:* to check if the package is valid and meets necessary criteria
- *install:* install the package to a local repository
- *deploy:* copy the package to the remote repository

Q) My code needs library/dependencies, where will it search for the first and next?

First it will search in local repository, next it searches in remote repository

if it finds the library in remote repository --> it downloads and stores in local repository

if it doesn't find the library in remote repository --> it checks for it in central repository, downloads and stores it to local