**B2B SAAS DevOps Implementation**

**Objective**

This document will describe how we are going to implement DevOps pipeline model for our B2B application to achieve Continuous Integration, Continuous Delivery.

**Application Overview**

B2B application is a java application and it is compatible with Jdk 8u261 version. We will use Tomcat (*version 9*) Webserver to run this application.

**Prerequisites**

* GitHub Enterprise Account
* Server with Jenkins and Docker

**Environments**

* Development
* Staging
* Production
* DR (Disaster Recovery – Only if we have On Premises servers)

**Managing BaseLine GitHub Repository**

We are going to create a repository and make it available for the companies that integrate with buyer exchange and also same will be available to developers by following steps.

* Login GitHub Enterprise Account
* Integrate Github with our organization’s LDAP or Active Directory to manage users.
* Create Organization
* Create a repository and make it public for the clients
* Who are all part of our organization will be available as member in Github.
* Prevent push and commit from other users except product owners for this repository by managing access to the repository.
* Create Teams for our developers on Github and manage access to each developer.
* Duplicate the public repository and make it as private to manage code internally by our developers.
* Once commit happen in master branch of private repository we will push the same code to public repository to be available for the clients.

**Security Vulnerabilities**

We will use Veracode tool to check security vulnerability on third-party components in B2B application’s baseline code. We will integrate this tool into Jenkins using veracode plugin and will do vulnerability scan by adding this as a part of pipeline before build.

**Code Integration**

We are going to manage code integration by using below GitHub features.

* Git Describe Command
* GitHub Actions

**Git Describe Command Overview**

Git describe is a useful command and it is used for version control. This command will find the most recent tag and return it with adding suffix on it with number of commits done after the tag and abbreviated object name of the most recent commit.

*git describe <Options>*

Example: If tag is 1.0.1 and we have 10 commits after it, git describe will return us 1.0.1–10-g1546

**Gitflow and Branching**

Here we are going to use GitHub Actions to achieve integration part. Below diagram will show for flow that how to manage source code by creating branches for developers and version control in Continuous integration and Continuous Delivery from Git to Production. Here 3 phases are there.

**Phase 1:** Developer make changes in code and then push the changes to git which will trigger the Jenkins pipeline. Pipeline will create a docker image and tags it with output of git describe command then push into docker registry and deploy container to dev environment. This process will repeat until a developer raise a pull request.

**Phase 2:** Once a Developer raise a pull request, Reviewer will review the code and accept it if satisfied then merge into release branch. New version will be updated by removing updated suffix given by describe command, Pipeline will create new docker image with new tag then push it to registry and deploy into staging environment.

**Phase 3:** If staging environment is success, then push the changes into master branch. This time pipeline won’t create new docker image, it will use the same docker image which was created when staging. Same version will be release on production environment.

Developer

Feature Branch

Jenkins

Pull Request

Push Image

st

Push Image

Docker Registry

Docker CLI

Development Environment

Test Pull Request

Reviewer

Reviewer

Master Branch

Staging Branch

Jenkins

Jenkins

Deploy

Deploy

DR

Optional Deploy

Staging Environment

Production Environment

Merge Request

Deploy

Verify Staging

Merge Request

**Build and Unit Test**

We will use Maven with Junit framework for Unit Testing. We will configure Junit test cases into POM.xml file. We will perform unit test and publish report using Junit. We will run unit test by giving below maven command.

*“mvn clean test”*

Once unit test success, maven will build the B2B application using below command.

*“mvn clean package”*

We will include these steps in pipeline like below to build docker image and push it to docker hub.

*“stage('Mvn Package'){*

*def mvnHome = tool name: 'maven-3', type: 'maven'*

*def mvnCMD = "${mvnHome}/bin/mvn"*

*sh "${mvnCMD} clean package"*

*}*

*stage('Build Docker Image'){*

*sh 'docker build -t baton/b2b:2.0.0 .'*

*}*

*stage('Push Docker Image'){*

*withCredentials([string(credentialsId: 'docker-pwd', variable: 'dockerHubPwd')]) {*

*sh "docker login -u baton -p ${dockerHubPwd}"*

*}*

*sh 'docker push baton/b2b:2.0.0'*

*}”*

**Managing Artifacts**

Here we are using Nexus repository to archive artifacts and manage. Also we are using Docker Registry to store Docker images. We will pull this image for deployment base on expression given in Jenkins parameters.

**Integration Test**

For Integration test we use SonarQube tool to analysis b2b application. After configuration of SonarQube on Jenkins, we have to modify POM.xml file with testing parameters using sonar plugin. After configuration of POM.xml we can implement the same into pipeline like below.

“stage('SonarQube analysis') {

steps {

sh "/usr/bin/sonar-scanner"

}

}”

**Managing Release**

We have created a pipeline job to deploy B2B application from git to production. In case any failures occurred in the middle of release Jenkins automatically revert back the changes to previous build and we will configure email alert with every checkpoint.

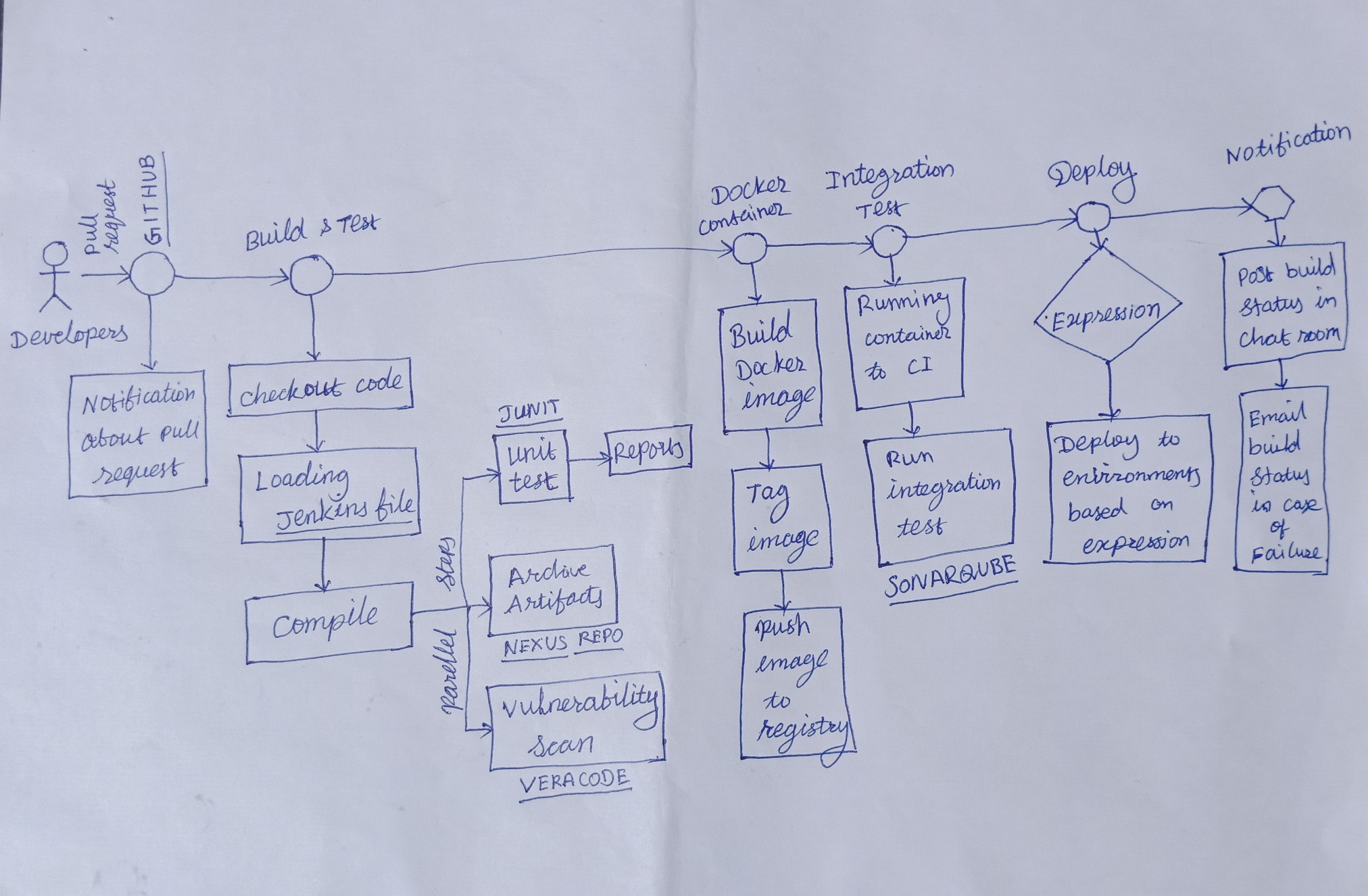
**Jenkins Configuration**

Before creating any pipeline job, we should install necessary plugin’s and integrate tools we are using in this project. In this project we are going to use below main tools to implement a CI/CD for B2B application.

* GitHub
* VeraCode
* Maven
* Junit
* SonarQube
* Docker
* Tomcat

**CI/CD Pipeline**

We will write a JenkinsFile and DockerFile along with B2B application’s source code on GitHub repository. So that we can create a pipeline job by selecting option “Pipeline script from SCM”. Below diagram will show the flow of pipeline.



Maven

* Once the developer commit the changes into private repository Jenkins job will be triggered and pull the code from master private repository.
* Once checkout the code, notification will be send on chat room on github and the Jenkins file which already available in repo will be loaded by Jenkins.
* Jenkins file will initiate CICD and start the pipeline.
* In first step, it will compile the code. if it success then run parallel jobs, unit test archive artifactory to nexus repository and then run vulnerability scan for code analysis using veracode tool.
* These all jobs succeed then go to next step other wise stop build and send notification with error message.
* In the next step pipeline will create docker image and tag it and then push into docker registry.
* Once docker image published, Jenkins will trigger integration test and SonarQube will do the analysis and publish the report. Once the report meet the requirments of testing then go to deployment step.
* In this step we will pass some parameters to choose environment automatically.
* For Example, We can choose environment by docker image tag as we are maintaining app version environment wise.
* For production release, jenkins will send the request to reviewer after staging deployment done and it will wait for the input from reviewer. If reviewer approved, Jenkins will continue production deployment, otherwise it will stop the build and wait for changes occur in github.

**Sample Jenkins File:**

In this file I have written a simple Jenkins file to deploy our B2B application.

“”

node{

stage('Git Checkout'){

git credentialsId: 'git-creds', url: 'https://github.com/javahometech/my-app'

}

stage('Mvn Package'){

def mvnHome = tool name: 'maven-3', type: 'maven'

def mvnCMD = "${mvnHome}/bin/mvn"

sh "${mvnCMD} clean package"

}

stage('Build Docker Image'){

sh 'docker build -t baton/b2b:2.0.0 .'

}

stage('Push Docker Image'){

withCredentials([string(credentialsId: 'docker-pwd', variable: 'dockerHubPwd')]) {

sh "docker login -u baton -p ${dockerHubPwd}"

}

sh 'docker push baton/b2b:2.0.0'

}

stage('Run Container on Dev Server'){

def dockerRun = 'docker run -p 8080:8080 -d --name b2b baton/b2b:2.0.0'

sshagent(['dev-server']) {

sh "ssh -o StrictHostKeyChecking=no ec2-user@172.31.18.198 ${dockerRun}"

}

} “

**Docker File:**

‘’

FROM tomcat:9

COPY \*.war /usr/local/tomcat/webapps/b2b-2.0.0.war

**‘’**