**Automatic Build Creation & deployment(QA)**

**Tools Used:-**

**1.GIT :-** Git is a distributed version Control System.Git takes *snapshots* of a project, and stores those snapshots as unique versions.If you go off in a direction with your project that you decide was the wrong direction, you can just roll back to the last *good* version and continue along an alternate path.If you're collaborating, then when someone sends you changes, you can merge those changes into your working branch, and then your collaborator can grab the merged version of the project and continue working from the new current version.

**2.Maven:-** It makes a project easy to build

1. It provides uniform build process (maven project can be shared by all the maven projects)
2. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
3. It is easy to migrate for new features of Maven

**3.Jenkins:-** Jenkins is a Continuous Integration server.Basically Continuous Integration is the practice of running your tests on a non-developer machine automatically every time someone pushes new code into the source repository.

This has the tremendous advantage of always knowing if all tests work and getting fast feedback. The fast feedback is important so you always know right after you broke the build (introduced changes that made either the compile/build cycle or the tests fail) what you did that failed and how to revert it.

**4.Slack:-** Slack is a powerful, customisable, and intuitive tool for improving team collaboration and productivity.

Main benefits:

* Slack integrates with over 300 other services including Dropbox, Asana, Google Drive, Skype, and more.
* Slack makes it easy to work in one place, without needing to switch between applications.

5.Nexus :- Sonatype nexus is used for hosting Maven and other repositories. Currently the Open source version supports Maven, Docker, NuGet, npm, PyPI, Bower.

6. **Ansible** :- Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

7. **Liquibase** :- Liquibase is an open source database change management tool built on Java. Rather than writing SQL directly against the database to create, update or drop database objects, developers define their desired database changes in XML files.

**Server Provisioning:-**

Designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

**EFFICIENT ARCHITECTURE**

Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished.

Your library of modules can reside on any machine, and there are no servers, daemons, or databases required. Typically you'll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content.

**SSH KEY USED FOR COMMUNICATION**

Passwords are supported, but SSH keys with ssh-agent are one of the best ways to use Ansible. Though if you want to use Kerberos, that's good too. Lots of options! Root logins are not required, you can login as any user, and then su or sudo to any user.

Ansible "authorized\_key" module is a great way to use ansible to control what machines can access what hosts. Other options, like kerberos or identity management systems, can also be used.

**ssh-agent bash  
ssh-add ~/.ssh/id\_rsa**

**Continuous Integration:**- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by Unit Test Cases, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily.

In the Jenkins,[Multibranch Pipeline jobs](https://jenkins.io/doc/pipeline/) get most of their configuration from a script that you put in the source repository. This is a great solution, since we can version the configuration and don’t have to manually enter at Git repository for branches and builds all of those branches automatically the Jenkins user interface. Another great feature is that a multibranch pipeline scans

While the syntax for defining a Pipeline, either in the web UI or with a Jenkins file, is the same, it’s generally considered best practice to define the Pipeline in a Jenkins file and check that into source control.

A Jenkins file is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. Consider the following Pipeline which implements a basic three-stage continuous delivery pipeline.

Below is the jenkinGit repository Knew about the commits with the help of jenkins files.

Creating a Jenkins file, which is checked into source control, provides a number of immediate benefits:

* Code review/iteration on the Pipeline
* Audit trail for the Pipeline
* Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

s file format which triggers the job on every commit on release branch.

**Jenkins file (Scripted Pipeline)**

*node { // <1>  
 stage('Build') { // <2>  
 /\* .. snip .. \*/  
 }  
 stage('Test') {  
 /\* .. snip .. \*/  
 }  
 stage('Deploy') {  
 /\* .. snip .. \*/  
 }  
}*

Stages which are defined in jenkins file can be seen on Jenkins User Interface.

In Git we are using two types of branches:-

-Release Branch

-Story Branch

**Event :-**

In start of every sprint new Release branch is created.From this every developer takes its story branch and Start Working on any CR.

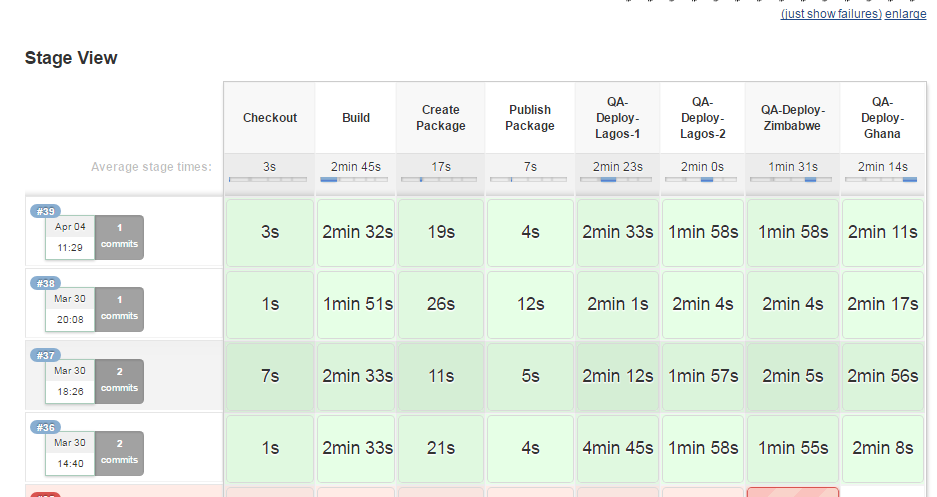
After completing CR Code is merged into release branch and same Release Branch is build and uploaded in Nexus Repository

Command used for this is :-

*Mvn clean install*

To upload build in Nexus:-

*curl -v -u <username>:<password> --upload-file file\_name*



**Stages:-**

**1.Checkout :-**

Source Code is Checkout from Git after merging.

**2.Build:-**

Project is Build with the help of maven and unit test cases are executed on given project which results in either success or failure of the build. We will get notified on slack automatically whether build is successful or failed. if the build is successfull then only it is copied to nexus otherwise previous war is replaced.

**To executed unit test cases:-**

*mvn clean test*

**3.Create Package:-**

A package is created after building project which contains a war (Web Archive file )

And db Scripts.

**4.Public Package:-**

**Package is published to Nexus Repository into gZip format.**

**5.QA Deployment:-**

-**First Jboss (Application Server) is shutdown for updates**

-gZip package is downloaded from nexus.after extraction from it

War files is replaced in all the Application Servers.

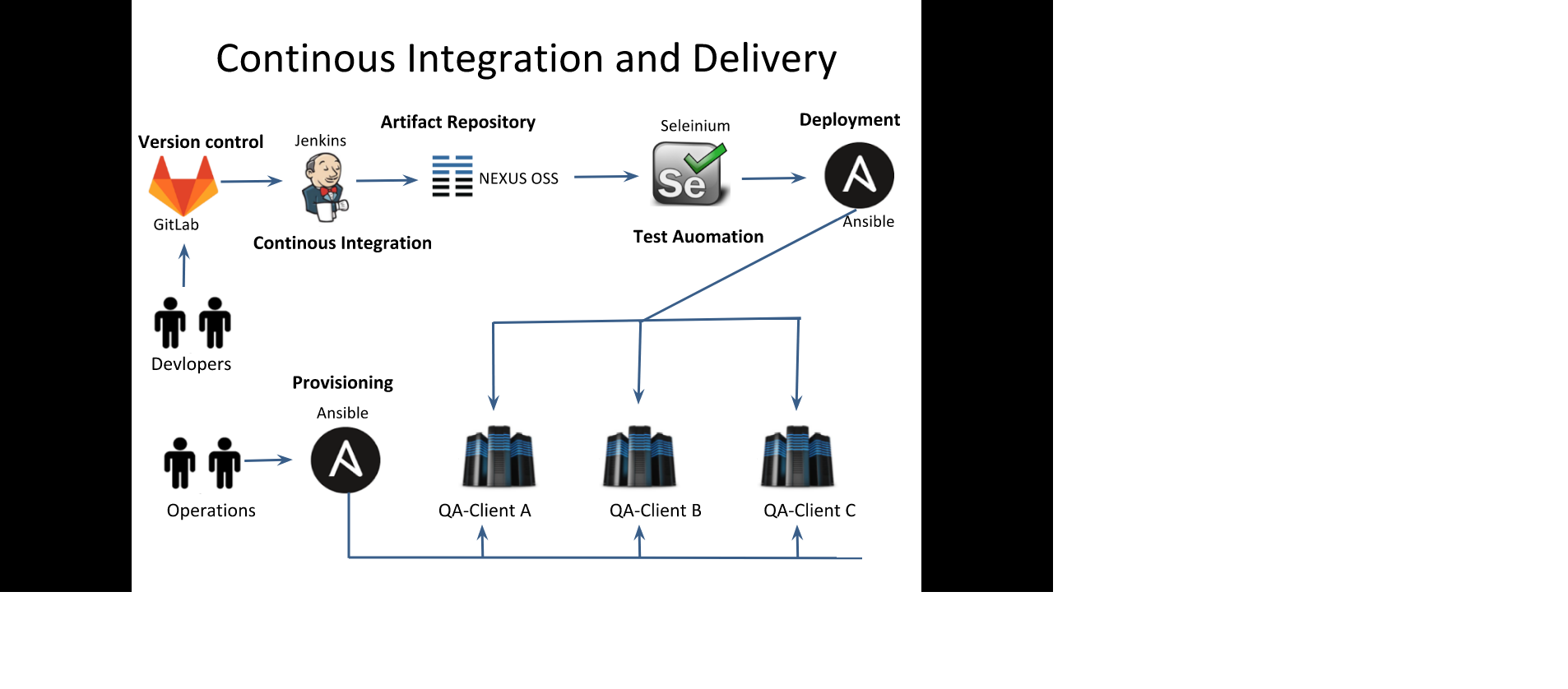
-Db Scripts are executed with the help of liquibase.

-Command use for running liquibase Scripts is:-

Java -jar liquibase.jar update

-Server is started again

- Smoke Testing is done by testers to check whethers the links ,pages are working or not.



**Manual Build Creation & deployment (UAT/Production)**

-a repository is tagged whenever a release is planned.

Tagging means it points a pointer to latest commit.

From there a branch is created.

git fetch --tag:- is used to show all the tags on your local environment.

git checkout -b “<Branch Name>”tagName :- is used to create a new branch from this tag.

To build project :-

Mvn clean install command is used.

**Artifact Deployment**

Artifact Deployment includes Application and DataBase Updates.

**Application**

War files created are copied to UAT and production servers with the help of scp from the local environment.

scp <path of war File> username@ipofServer:<path>

**DataBase**

DB Scripts will be executed on the UAT and production Environment manually.

DB Scripts will be taken out from the branch that has been created from tag.

-First Jboss (Application Server) is shutdown for updates

-war files are copied into deployment folder.

-db scripts is executed manually.

-jboss is started again

Final inspection will be done By SQA Team.