Maven.

It is a project management tool.

Build.

compile.

Run Tests.

Package Jar. Package War

Deploy to server.

maven uses convention over configuration.

myproject

- src/main/java
- src/main/resources
- src/test/java
- src/test/resources

mvn install

↳ compile

run.

package.

Archetypes.

Standalone
Webapp
ear
} archetype

Why maven?

Common interface

more time consuming

2. Dependencies.

<dependency>

<artifactId> junit.jar

</dependency>.

3. Repositories.

→ plugin.

Result:

Plugin model.

Compiler

Surefile.

Wsimport.

pom.xml:

Project object model.

<groupId> com.example </groupId>

<artifactId> Hellomaven </artifactId>

<packaging> jar </packaging>

<version> 1.0 </version>

} Maven Coordinates.

Building the project from Commandline:

cd Hello-maven/

$ mvn install

plugins:

Maven plugin is a collection of goals.

archetype: generate.
└ goal from archetype plugin

install: install
└ goal from install plugin.

compile

test

Package.

These goals can also take parameters:

mvn archetype: generate -DgroupId=com.example -Dartifa

Id. = hellomaven.

Plugin: goalId

Lifecycle phases:

Phase and Goals:

mvn install

1. process-resources ——— resources: resources

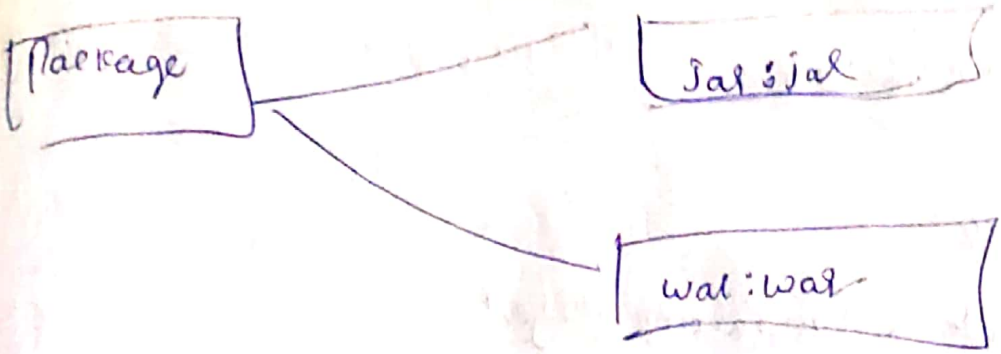
2. compile — compiler: compile

3. test — surefire: test

4. package. — jar: jar

Each phase is associated with resource: resource

↓ Plugin ↓ goal.



coordinates:-

Maven coordinates:

1. groupid
2. artifactid
3. version.
4. packaging

groupid : artifactid : packaging : version.

- groupid → com.example.
- artifactid → hellomaven
- packaging → jar or war project
- version → hellomaven-1.0.jar

Repositories:-

Default maven location.

http://repo.maven.apache.org/maven2/

Maven in Eclipse:

1. Use case
2. Create a standalone project
3. Customize setting
4. Implement the Data and Business Layer
5. Add dependencies
6. Build the project.

Product web:

Create
display

Weblayer
business layer
dao layer

Servlets
pojo
Spring

Customize the compiler setting:

Standalone

Business logic layer

ProductBo

productBOIMP

Data Access layer.

Data Access Layer

Product DAO

Product DAO Impl

Create

Read

Update

Delete

Building from command line

$ mvn install to build project.

Maven web application:

Create the servlets

Add product

Display product

Add the Servlet dependicy

Create the html page

Multimodule projects

Parent POM. Build & run

change the child poms

code the servlets. web.xml

To create a multi-module project

Create a parent folder — pom.xml

ProductServices — pom.xml

ProductWeb — pom.xml

$ maven clean install

Reactor decides build order

Scopes:

Maven scopes

```
<dependency>
    <scope>  </scope>
```

6 Scopes

1. compile	5. System
2. runtime	6. Import
3. provided	
4. test	

Compile time scope:

Available at
- build
- test
- run

provided: required during build, test run not need to part of application.

servlet-api.

Runtime: only for running the test

test

run. not for compilation.

test: available for compile the tests.

run the test Junit is example.

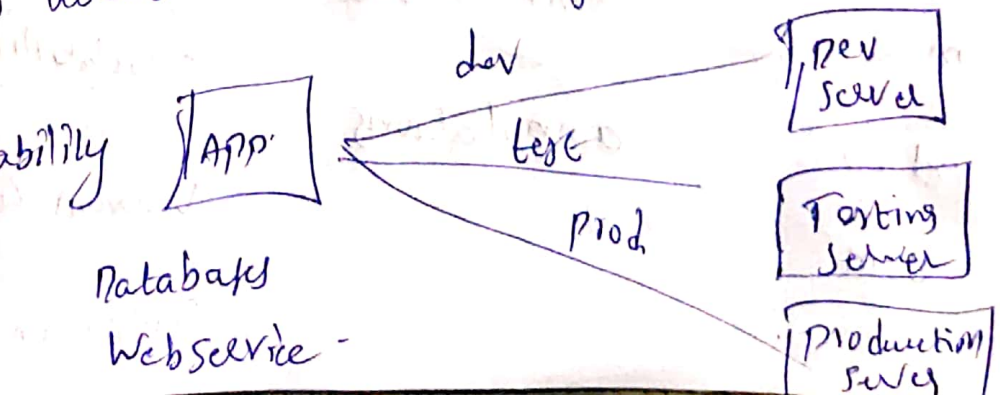
system scope is similar to provided.

we will copy this to sub directory of project.

Build portability:

Means developer able to build the application across any environment without changing any configuration.

Maven allows Build portability through profiles.



dev.
    application.properties

test
    application.properties

Prod application.properties

---

mvn install -pdev
↓
profile.

application.properties:

---

db.url = devurl

db.userName = dev

db.password = dev

mvn install -pprod.

Traditional Spring:

Core
mvc
dao
orm

xml based
annotations

module
availability
Version compatible
Deployment.

Spring Boot features:

1. Auto configuration
2. mvc — DispatcherServlet
3. orm — DataSource, Transaction manager

Spring Boot starters:

- Spring Boot - Starter pattern
- Spring Boot - Starter - Web
- Spring Boot - Starter - data - jpa

Embedded Servlet Container

- Tomcat
- Jetty
- Undertow

Spring Boot Actuators:

- autoconfig
- mappings
- info
- health
- metrics

```
@SpringBootApplication
Public class coreApplication {
Public static void main (String[] args) {
SpringApplication.run(coreApplication.class, args);
}
```

@EnableAutoConfiguration.

XML

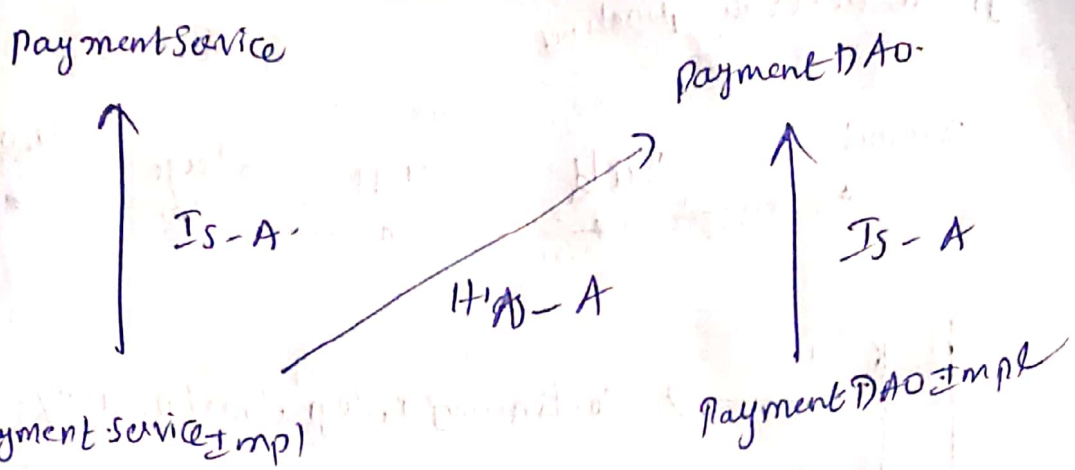
Java

H2 DB.

MySQL

Spring web

```
@RunWith(SpringRunner.class)
@SpringBootTest
@Test
Public void contextLoads() {
Public class coreApplicationTests {
@Test
Public void contextLoads() {
}
}
```



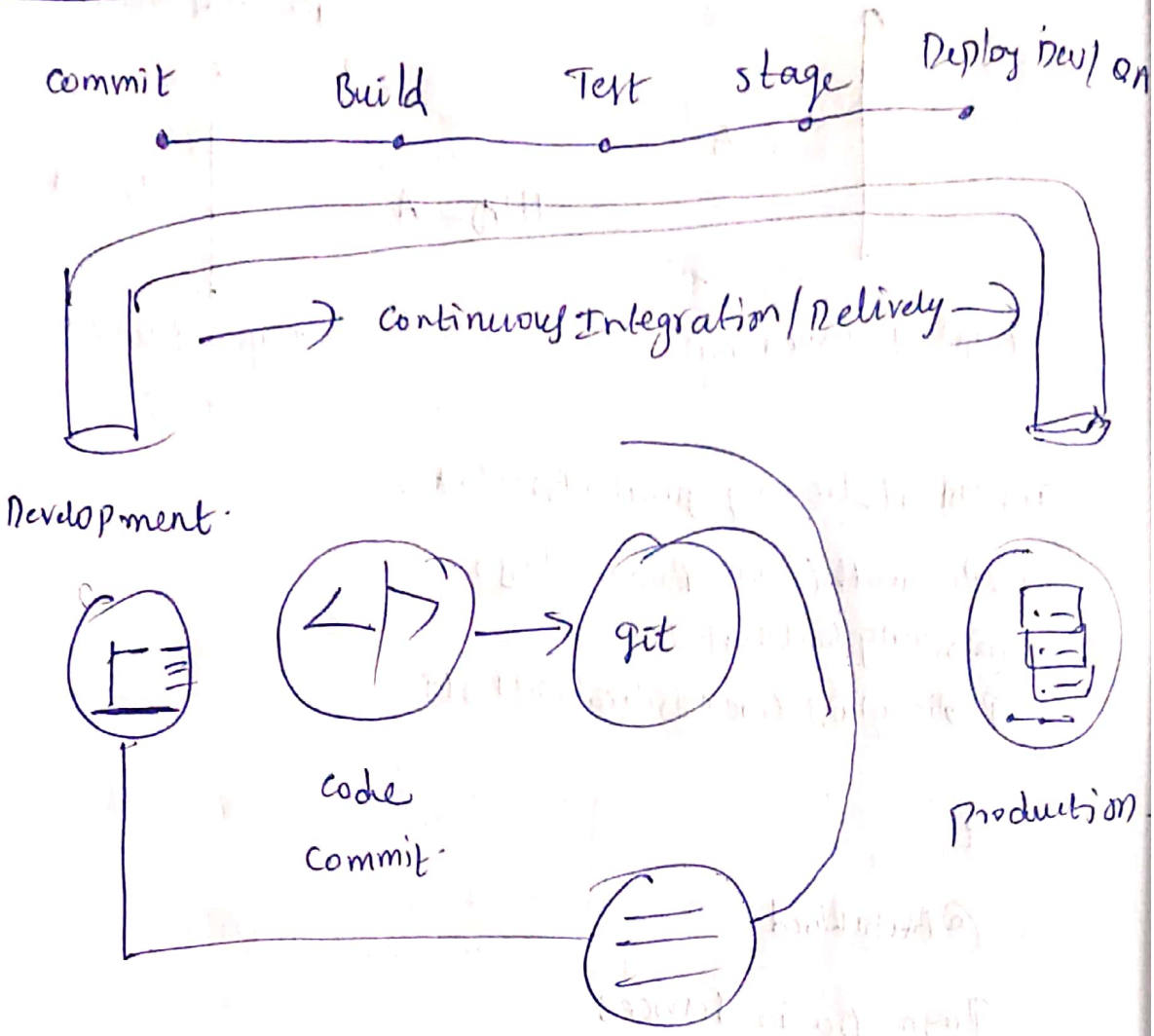

```
import static org.junit.Assert.*;
@RunWith(SpringRunner.class)
@SpringBootTest
Public class CoreApplicationTests
{
@Autowired
PaymentService service;

@Test
Public void testdependency()
{
assert.NotNull(service);
}
}
```

Jenkins Quick Start:

CI/CD Continuous Integration

Continuous Delivery / Deployment.

Typical delivery pipeline:



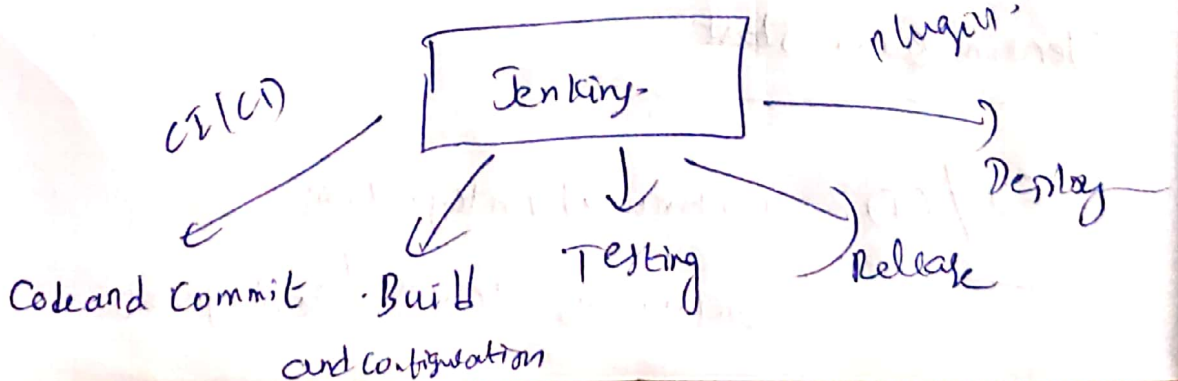
Source · Build Test Deploy monitor

Commit compile Integration preprod → Health

Codereview UnitTest System prod Unusual

Program War/jar/dll Load Activities

Image UI



AWS EC2

Elastic cloud compute.

AMI Amazon machine Image.

AMI

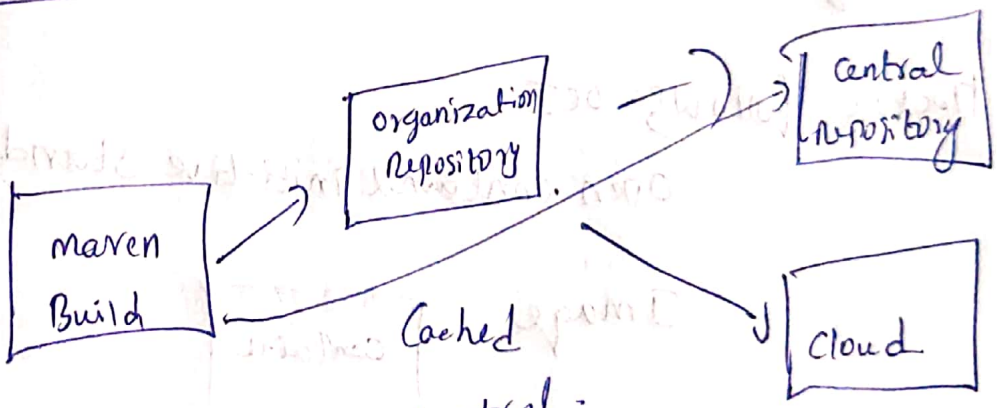
OS

Software

Java.
Python
mysql
Docker

Putty.
↓
tool for windows

Repository Manager:



Repository for our project artifacts

Nexus repository manager

Docker Image

Admin login

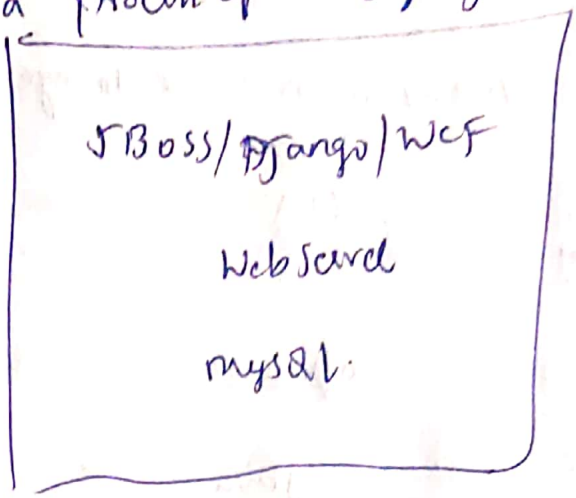
Proxy.

Docker is a containerization tool

We need :- It is a procss of bringing all together

Physical machine

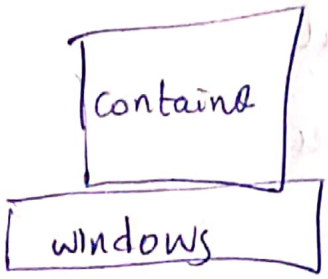
Vm

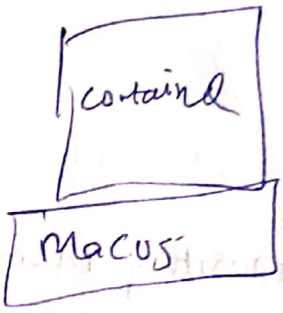


Containers

New Test

Stage



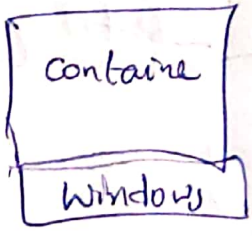


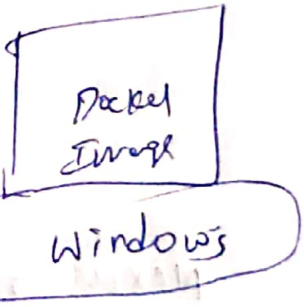
. Docker follows OCI

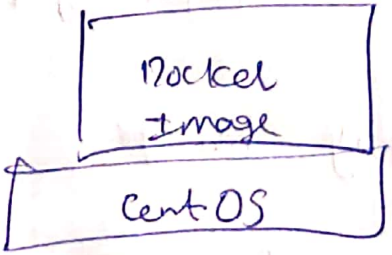
Open container initiative standard

Image

Container







simple

virtualization platform

Fast

Image

Containers

Repository types

1. hosted repository :- host our artifact
2. proxy repository :- Not a repository of own
3. group repository :-