BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Devops Platforms – Maven and Jenkins

Bharani Dharan Krishnaswamy

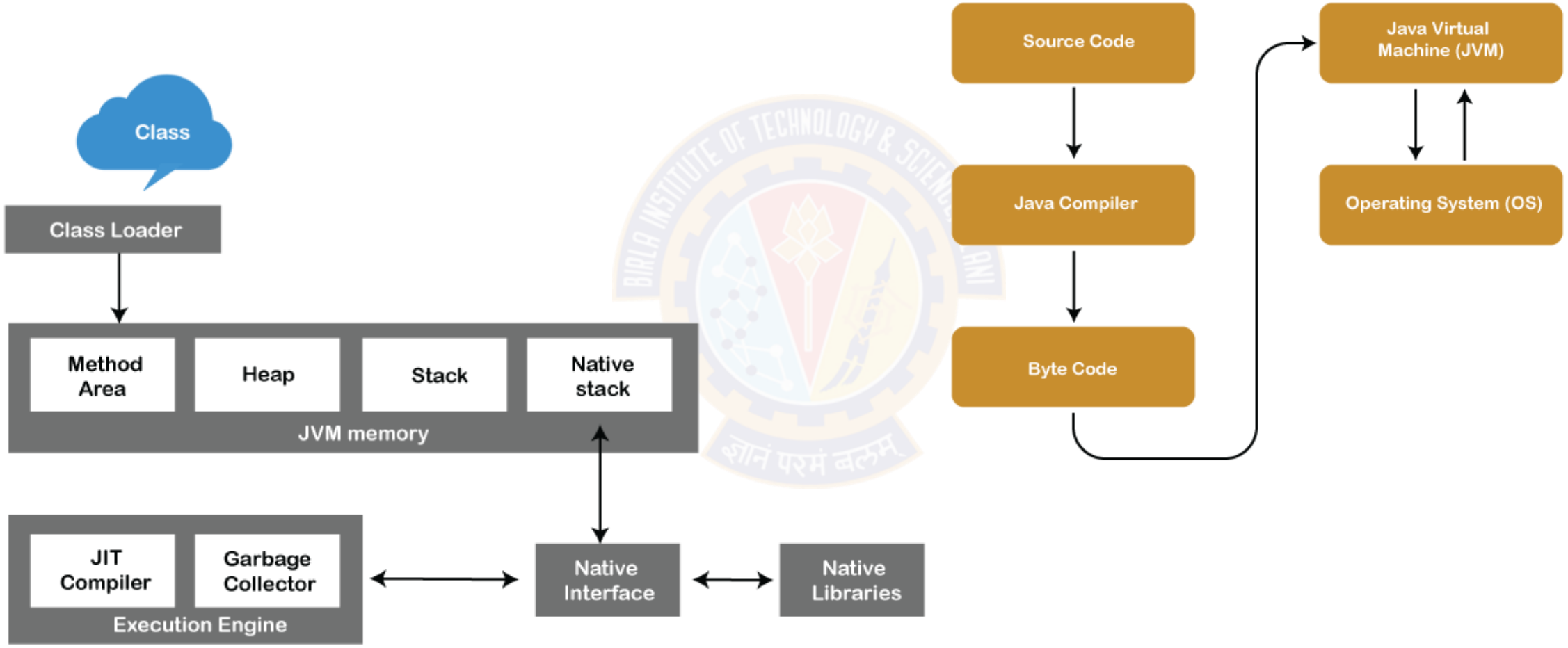
Introduction

Things to Know

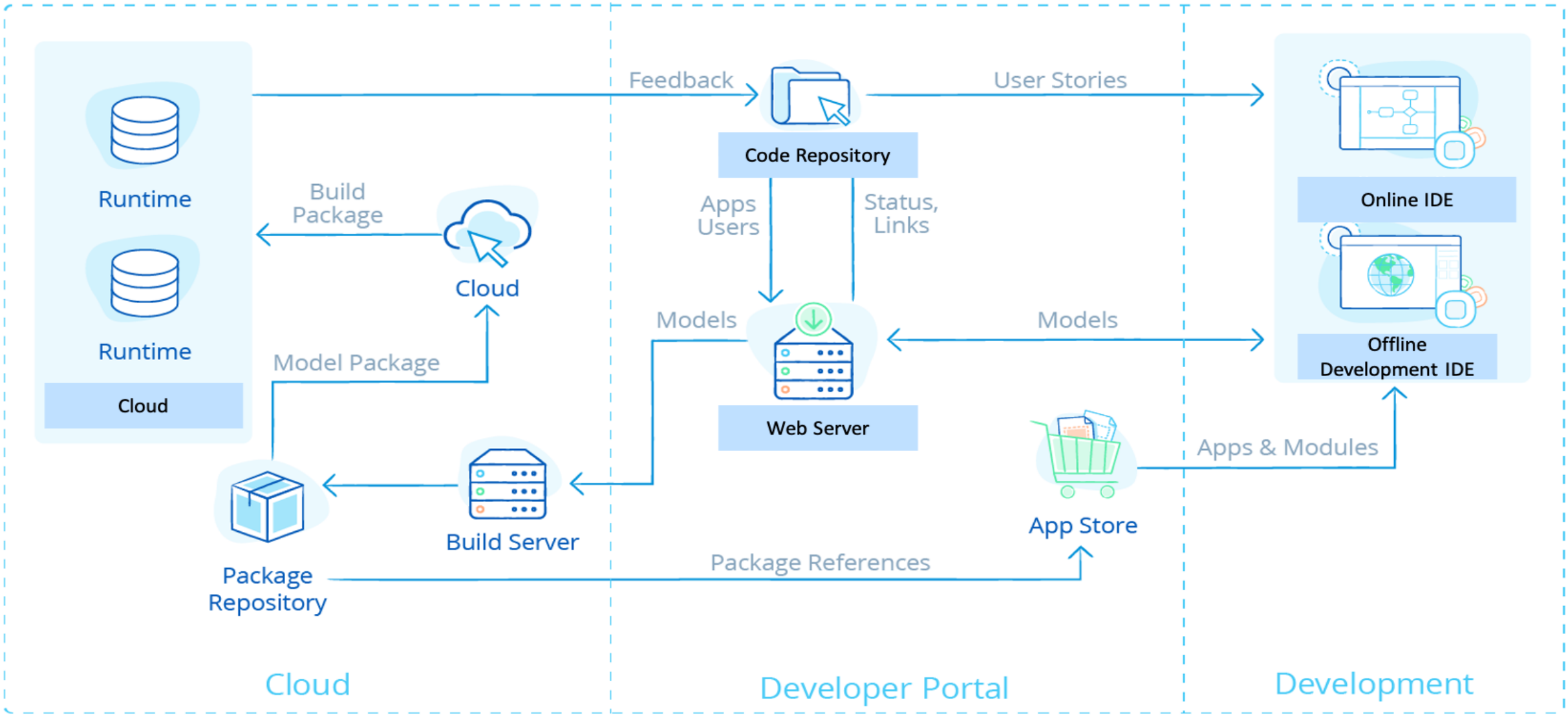
- Basics of OS & Networking – **Linux, TCS, IP**
- Scripting – Shell Scripting (**Linux** - Preferred / Windows)
- Configuration Management (**GIT**, Bitbucket, Teamcity, SVN, **Jenkins**)
- Version Control (**GIT**, Bitbucket, Teamcity, SVN etc.,)
- Build Automation (**Maven**, Ant, Gradle, Groovy)
- Continuous Integration (**GIT**, Bitbucket, Teamcity, SVN etc.,)
- Continuous Deployment (**Jenkins**, CircleCI etc.,)
- Deployment Automation (**Cron**, **Shell Scripts, Jenkins**)
- Infrastructure Distribution & Orchestration (Solution Architecture & Network Topology)
- Monitoring & Analytics (Google Analytics, Firebase Analytics, Prometheus, Grafana, New Relic, Azure App Insights, Elk, Fluentd etc.,)
- Testing Tools (**Junit**, Karma, Jasmine, Cucumber, Mockito, Selenium, Appium etc.,)
- Quality Measurement Tools (**ESLint**, **JSLint**, Apache Jmeter, Microfocus LoadRunner)

Java

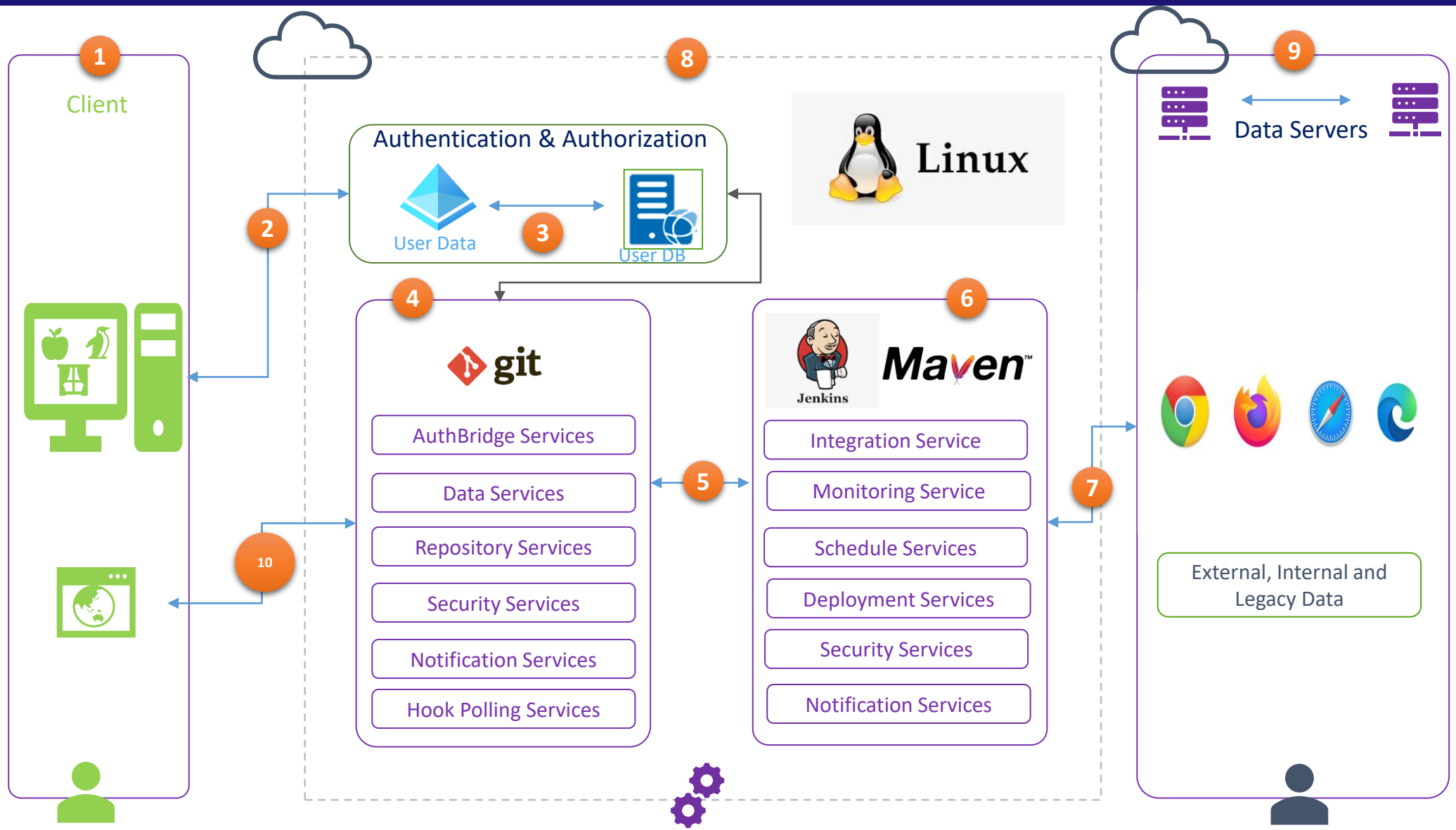
Architecture



Deployment Architecture



CI / CD Data Flow



1. Development Environment where code is developed by developers.
2. Code is Pushed via secured tunnel.
3. Authentication will be done using AD Authentication Mechanism.
4. Code gets stored in GIT Repository and new code gets pulled into the local repository after conflict resolution.
5. Build gets triggered based on the options configured in Jenkins
6. All the Build process gets executed one after the other in multiple containers.
7. The final executable is pushed to the respective store.
8. Linux Environment
9. Updates get published to the stores and end users are notified.
10. Updates gets patched into the end user devices.

Maven

Introduction

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices
- Uses POM (Project Object Model) file for Build.

Maven

POM File

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- The Basics -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <packaging>...</packaging>
  <dependencies>...</dependencies>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <modules>...</modules>
  <properties>...</properties>

  <!-- Build Settings -->
  <build>...</build>
  <reporting>...</reporting>

  <!-- More Project Information -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <organization>...</organization>
  <developers>...</developers>
  <contributors>...</contributors>

  <!-- Environment Settings -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

Maven

POM File

- **groupId**: This is generally unique amongst an organization or a project.
- **artifactId**: The artifactId is generally the name that the project is known by.
- **version**: This is to provide the version for the particular build
- The three elements given above point to a specific version of a project, letting Maven know *who* we are dealing with, and *when* in its software lifecycle we want them.
- **package**: This is to provide the executional extension (example : jar / war). Default is jar.
- **Dependencies, dependency**: Most projects depend on others to build and run correctly. As an added bonus, Maven brings in the dependencies of those dependencies (transitive dependencies), allowing your list to focus solely on the dependencies your project requires.
- **classifier**: The classifier distinguishes artifacts that were built from the same POM but differ in content. It is some optional and arbitrary string that - if present - is appended to the artifact name just after the version number. Example a project that offers an artifact targeting Java 11 but at the same time also an artifact that still supports Java 1.8. The first artifact could be equipped with the classifier jdk11 and the second one with jdk8 such that clients can choose which one to use.

Maven

POM File

- **type**: Corresponds to the chosen dependency type. This defaults to jar. While it usually represents the extension on the filename of the dependency, that is not always the case: a type can be mapped to a different extension and a classifier. Example ejb-client, test-jar
- **scope**: This element refers to the classpath of the task at hand (compiling and runtime, testing, etc.) as well as how to limit the transitivity of a dependency. There are five scopes available:
- **compile** - this is the default scope, used if none is specified. Compile dependencies are available in all classpaths. Furthermore, those dependencies are propagated to dependent projects.
- **provided** - this is much like compile, but indicates you expect the JDK or a container to provide it at runtime. It is only available on the compilation and test classpath, and is not transitive.
- **runtime** - this scope indicates that the dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.
- **test** - this scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. It is not transitive.
- **system** - this scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.

Maven

POM File

- **systemPath:** used *only* if the dependency scope is system. Otherwise, the build will fail if this element is set. The path must be absolute, so it is recommended to use a property to specify the machine-specific path such as ${java.home}/lib.
- **optional**: Marks a dependency optional when this project itself is a dependency. For example, imagine a project A that depends upon project B to compile a portion of code that may not be used at runtime, then we may have no need for project B for all project. So if project X adds project A as its own dependency, then Maven does not need to install project B at all. Symbolically, if => represents a required dependency, and --> represents optional, although A=>B may be the case when building A X=>A-->B would be the case when building X.
- There are other properties and configuration too in POM file. For more info, please check in : https://maven.apache.org/pom.html#What_is_the_POM

Maven

Semantic Versioning

- Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes
2. MINOR version when you add functionality in a backwards compatible manner
3. PATCH version when you make backwards compatible bug fixes

- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.
- For more info, look into : https://semver.org/

Maven

Commands

- mvn compile - To convert java file to class file
- mvn test - Execute the Test Cases
- mvn package - Package the class to Jar / War
- mvn install - Compile, Test and Package, Then copy the executable to the local dependency repository
- java -jar - Execute the jar file
- mvn --version / mvn -v –> To check the version of Maven

Maven

Plugins used

- surefire - Maven Plugin to run unit tests.
- Shade - Maven Plugin to package and artifact in Shaded jar (Fat Jar)

Cron Expressions

How to Schedule Cron Jobs ?

- * * * * * - Minute Hour DayOfMonth Month DayOfWeek

Name	Allowed Values
Minutes	0-59
Hours	0-23
Day of month	1-31
Month	0-11 or JAN-DEC
Day of week	1-7 or SUN-SAT

*	any value
,	value list separator
-	range of values
/	step values
@yearly	(non-standard)
@annually	(non-standard)
@monthly	(non-standard)
@weekly	(non-standard)
@daily	(non-standard)
@hourly	(non-standard)
@reboot	(non-standard)

Cron Expressions

Expression	Means
0 0 12 * *	Fire at 12:00 PM (noon) every day
0 15 10 * *	Fire at 10:15 AM every day
0 * 14 * *	Fire every minute starting at 2:00 PM and ending at 2:59 PM, every day
0 0/5 14 * *	Fire every 5 minutes starting at 2:00 PM and ending at 2:55 PM, every day
0 0/5 14,18 * *	Fire every 5 minutes starting at 2:00 PM and ending at 2:55 PM, AND fire every 5 minutes starting at 6:00 PM and ending at 6:55 PM, every day
0 0-5 14 * *	Fire every minute starting at 2:00 PM and ending at 2:05 PM, every day
0 10,44 14 3 WED	Fire at 2:10 PM and at 2:44 PM every Wednesday in the month of March
0 15 10 * MON-FRI	Fire at 10:15 AM every Monday, Tuesday, Wednesday, Thursday and Friday
0 15 10 15 *	Fire at 10:15 AM on the 15th day of every month
0 15 10 L *	Fire at 10:15 AM on the last day of every month
0 0 12 1/5 *	Fire at 12 PM (noon) every 5 days every month, starting on the first day of the month
0 11 11 11 11	Fire every November 11 at 11:11 AM

Thank You!

In our next session: