**1.Explain the working and the differences between Maven, Gradle and SBT in detail.**

SBT SBT is very simple and it is focused on Scala it relies on Ivy for dependency management.

Maven Maven it's a great build tool and it enables to control the entire software lifecycle with XML files. Using the Project Object Model you can intercept all points of the software lifecycle from compile to test, packaging and deploy. Maven has it's own dependency manager. Issue in Maven is the XML syntax, writing a POM can be annoying and too much expensive.

Gradle We can not compare Gradle with Maven (or SBT). Gradle is built on top of Maven, Ant and Ivy. It uses Maven repositories. Gradle doesn't use XML, it's a polyglot build tool. It combines the Ant API with the Groovy language to enable developers to write a build script with an intuitive DSL. With a few lines of code you can write a Gradle build script that can do the same things that Maven can do. With Gradle you can define your own task with the Groovy language and intercept programmatically your build execution. This functional approach is not for all developers,infact Maven it's good if you don't want this behavior in your build environment. Both Maven and Gradle have plugins to integrate your build with technologies used in your projects.

SBT: Sbt is an open source build tool for Scala and Java projects, similar to Java's Maven or Ant.

Its main features are:

native support for compiling Scala code and integrating with many Scala test frameworks build descriptions written in Scala using a DSL dependency management using Ivy (which supports Maven-format repositories) continuous compilation, testing, and deployment integration with the Scala interpreter for rapid iteration and debugging support for mixed Java/Scala projects

Maven Maven was released in 2004. Its goal was to improve upon some of the problems developers were facing when using Ant. Maven continues using XML as the format to write build specification. However, structure is diametrically different. While Ant requires developers to write all the commands that lead to the successful execution of some task, Maven relies on conventions and provides the available targets (goals) that can be invoked. As the additional, and probably most important addition, Maven introduced the ability to download dependencies over the network (later on adopted by Ant through Ivy). That in itself revolutionized the way we deliver software.

However, Maven has its own problems. Dependencies management does not handle well conflicts between different versions of the same library (something Ivy is much better at). XML as the build configuration format is strictly structured and highly standardized. Customization of targets (goals) is hard. Since Maven is focused mostly on dependency management, complex, customized build scripts are actually harder to write in Maven than in Ant.

Maven configuration written in XML continuous being big and cumbersome. On bigger projects it can have hundreds of lines of code without actually doing anything “extraordinary”.

Main benefit from Maven is its life-cycle. As long as the project is based on certain standards, with Maven one can pass through the whole life cycle with relative ease. This comes at a cost of flexibility.

In the mean time the interest for DSLs (Domain Specific Languages) continued increasing. The idea is to have languages designed to solve problems belonging to a specific domain. In case of builds, one of the results of applying DSL is Gradle

Gradle:

Gradle combines good parts of both tools and builds on top of them with DSL and other improvements. It has Ant’s power and flexibility with Maven’s life-cycle and ease of use. The end result is a tool that was released in 2012 and gained a lot of attention in a short period of time. For example, Google adopted Gradle as the default build tool for the Android OS.

Gradle does not use XML. Instead, it had its own DSL based on Groovy (one of JVM languages). As a result, Gradle build scripts tend to be much shorter and clearer than those written for Ant or Maven. The amount of boilerplate code is much smaller with Gradle since its DSL is designed to solve a specific problem: move software through its life cycle, from compilation through static analysis and testing until packaging and deployment.

Initially, Gradle used Apache Ivy for its dependency management. Later own it moved to its own native dependency resolution engine. Gradle effort can be summed as “convention is good and so is flexibility”.