**Build tools**

Build tools are commonly known as programs that automate the process of building an executable application from source code. This building process includes activities like compiling, linking and packaging the code into an executable form

**The activities include:**

* Downloading the dependencies.
* Compiling source code to the form of binary code.
* Packaging that binary code.
* Running the tests.
* Deploying them to the production systems.

**Types:**

* Scala oriented Build Tool (SBT):
* CMake:
* Terraform:
* Apache Maven:

**Software Development Kit:**

In order to use a SDK, a developer needs an iOS SDK to build iOS apps, an Android toolkit for building Android apps,

SDKs include documentation, application programming interfaces (APIs), code samples, libraries and processes, as well as guides that developers can use and integrate into their apps. Developers can use SDKs to build and maintain applications without having to write everything from scratch.

**Difference between SDKs and APIs**

An API is code that enables two software programs to communicate with each other. An API defines the correct way for a developer to request services from an operating system or other application and expose data within different contexts and across multiple channels.

When a developer uses an SDK to develop applications and create systems, those apps have to communicate with other applications. An SDK includes an API to enable that communication.

**Compilation:**

The compilation is **a process of converting the source code into object code**. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.

**What are Dependencies**

Dependency is a broad software engineering term used to refer when a piece of software relies on another one. Simply put, if Program A requires Program B to be able to run, Program A is dependent on Program B. This makes Program B a dependency of Program A.

It doesn’t really matter what it is, if your program needs to run correctly, it’s a dependency. Common examples of dependencies are programming libraries, Online services, programming scripts etc.

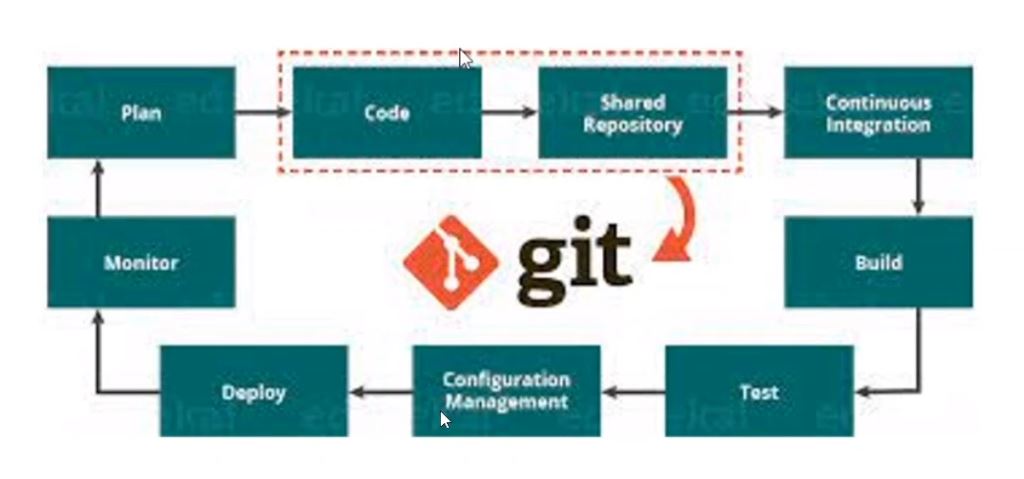
**Artifact**

**Artifact** is highly associated and related to specific methods or processes of development. Methods or processes can be project plans, business cases, or risk assessments. Distinct gathering and collections of detailed information are generally organized and incorporated into artifact sets. A set generally represents complete aspect of system. This is simply done to make development and establishment of complete software system in manageable manner. An artifact is one of many kinds of tangible by-products produced during the development of software.

**DevOps pipeline**

A DevOps pipeline is a set of automated processes and tools that allows both developers and operations professionals to work cohesively to build and deploy code to a production environment. While a DevOps pipeline can differ by organization, it typically includes build automation/continuous integration, automation testing, validation, and reporting. It may also include one or more manual gates that require human intervention before code is allowed to proceed

Software development process:



Devops trying to automate it with least human intervention

Multiple developers on multiple platforms

From repository and integration, devops come into picture

Build process starts through build tools

Build tools:

C – make

Java – Maven

**MAKE:**

We define makefile. It has set of instructions to read the code and to compile for C or C++

Every piece of code contains a main code and dependencies. How these two interact has to be defined in makefile.

**Dependencies:**

A piece of code interacts with another piece of code or application, to get desired result.

Dependency file:- <main.o>

Just in the form of functions, of what code has to interact with which dependency. Why? The easier the code, the easier is the integration, build process, development.

Code from different languages cannot be merged. But the functionalities of the codes can be merged

We can call for a function but not a code.

At the end we get a <.exe> file as output file. Final result.

**Artifactory** – all the necessary files needed for compilation and mapped in main.o file so execution exe file contains binary which has make file instructions and main.o file.

Artifactory can be used to store dependency files in cloud available for everyone, API calls (URL of file location)

**Ex:**

While <.c> file is given, it should be as a function. That is, void main () should be changed to void fact()

Three codes: big3.c fact.c fib.c

**main.c**

#include <stdio.h>

main() {

fact();

fib();

big3();

}

the order in which it is compiled in is according to <main.o> file.

makefile:

gcc – compiler for c

-o –compiles object file

-c – compiles c files

here in the first step as final.exe doesn’t exist,

Final.exe:main.o big3.o fact.o fib.o

gcc -o Final.exe:main.o big3.o fact.o fib.o

main.o:main.c

gcc -c main.c

fact.o:fact.c

gcc -c fact.c

big3.o:big3.c

gcc -c big3.c

fib.o:fib.c

gcc -c fib.c

so we have to give final.exe in the second line

to the compiler

(gcc -o Final.exe:main.o big3.o fact.o)

* Sudo yum install gcc –y

Commands for make file:

* Make

Should be saved as makefile.

If it is not saved as makefile, then

* make –f <name>
* make clean –f <name> 🡪 for deleting .o files

The order in which it is compiled in is according to <main.o> file.

If there are several functions, we can create one resource () {a1 a2 a3} in main.c and give all the functions necessary in the resource file.

**Maven**:

* maven as a build tool to manage or compile java (.java) projects
* earlier it was hosted on maven-apache, now c-shark, ruby, scala languages code can also be compiled in maven.

in make dependency file is object file main.o which has all the libraries, dependency are mentioned or defined. **in maven pom.xml is dependency. POM - project object model.**

**what does pom.xml contains?**

core element of maven. located in root directory.

* **-**dependencies/libraries -->
* -plugins
* -project versions
* -mailing list
* -repository details
* -build profile

java scripting is mainly confined to web applications.

we need a server to deploy the end compiled files.

* in make it gives executable file in the location itself, Linux server itself. but when it is maven, it is web application for internet. we have to deploy them to web server like apache tomcat.

Ex: apache tomacat

in tomcat installation --> webapps folder

webapps - all our compiled files will come and sit here.

in 8080 ports

manager

host-manager --> webapps

here we deploy the artifact or executable files.

for maven, final output is .jar, .war, .ear also known as artifacts used for another environment.

not as executable file. unpack to achieve functionality

these outputs cannot be used on interface or cli

they are like a zip file which has code and dependencies together. when we unpack,

then it can be used for creating webserver interface or its components (departments, components, etc.)

maven is not just a compiling tool, but also

* java is very dynamic language it has library/framework it can integrate with many frameworks for getting the desired outcome

many types of libraries, can be run in many platforms,

whenever the code is executed, the compiler goes to the library or framework and gets it

to work area or environment to give the final result.

if there is a build failure in maven project, if there is a discrepancy in pom.xml, we should know what is the issue.

dependencies or libraries acts as a link or call for the pom has dependencies --> this acts as a link --> when .jar file is unpacked -->

with the link from dependencies from pom and downloads the required ones.

**Maven build cycle:**

8 stages:

1. validate - check project is correct, dependencies, necessary details, libraries or pom.xml or throws up error
2. compile - compiles source code line by line and start generating binary
3. test - compiled source code is tested in the suitable framework as defined in pom.xml, done by maven itself
4. package - made into packages like .jar, .war, .ear (most commonly used as they can be unpacked in any env)
5. integration-testing - prototype step - ex: if it has micro services : it gives different jar. all these jars are brought together integrated and tested. in above testing, it is tested individually
6. verify - verifies
7. install - if local repository is defined, goes to pom.xml and installs the dependencies.
8. deploy - wherever package is created can be deployed there. all developers, testers can access all these.

* menu is one micro service
* checkout is one micro service.

using maven, we can debug and selectively start from any step. Yet, all the steps are run in the background. before maven, there used to be ANT. ANT's evolved version is maven. ant was slow, long scripting, not pom.xml architecture, tough to debug

maven has local repository, central repository, internet. if the requirement for maven is not met, it goes

to internet and downloads what is required.