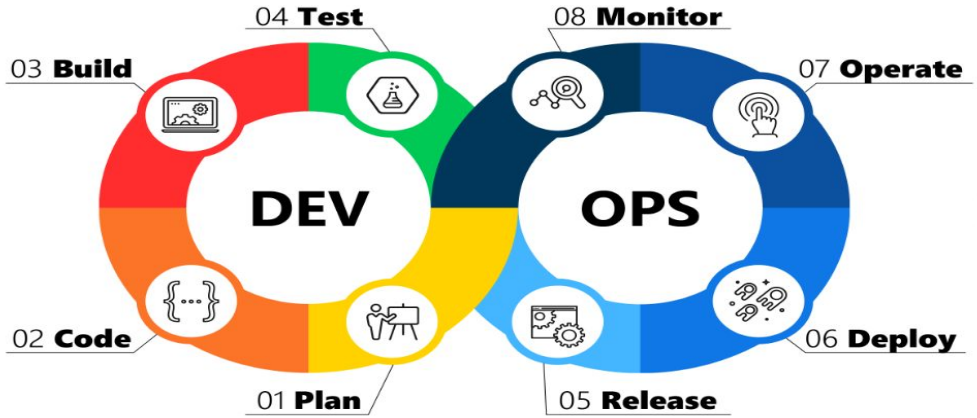
Course Code: IT-41

DevOps

Chapter - 1
Introduction to DevOps



Traditional IT Operations

- Developer codes, works on Dev environment / Local setup
  - If breaks set up everything again or request IT for new environment
  - IT sets up again (X days)
- QA tests on QA environment or Dev environment
  - If breaks set up everything again or request IT for new environment
  - IT sets up again (X days)
- Dev ships code to IT
- IT takes downtime
- Deploys
  - If breaks, war room meeting
  - It works on my machine! Blame games, etc.
- IT skeptic of any **CHANGE**

Traditional Thinking

Operations World

- Care About:
  - Everything is stable
  - Standards
  - Templates
  - Not getting bothered at 2:00 am
- Success:
  - Software is stable
  - Backup and restore works
  - Systems are operating within defined thresholds

Developers World

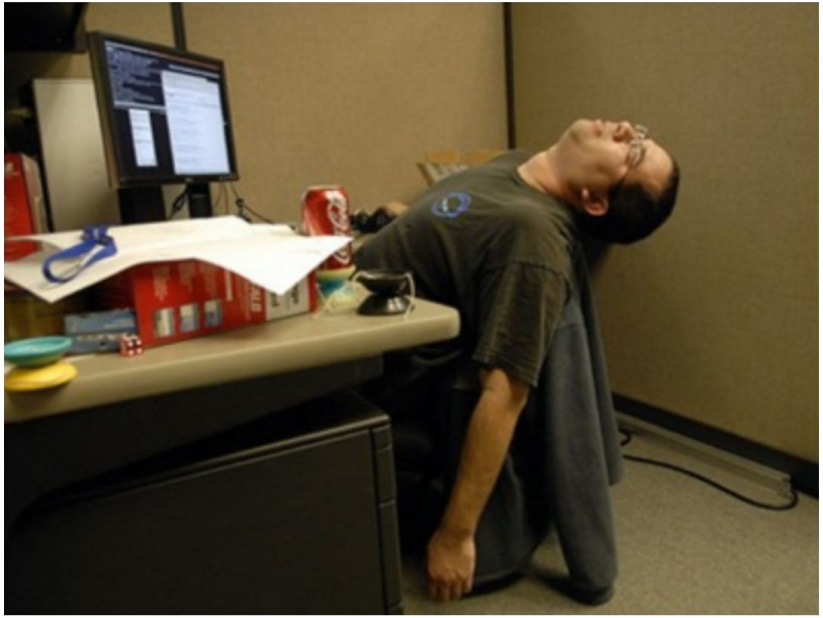
- Care About:
  - Writing Software
  - Working Code
  - APIs
  - Libraries
  - Sprints
- Success:
  - Software Works - Laptop and Test
  - Finished Sprint

Test Team World

- Care About:
  - Writing Test Code
  - Working Test Code
  - APIs
  - Frameworks
  - Sprints
- Success:
  - Test Works - Laptop and Testbed
  - Finished Sprint

How Developers See Ops

How Ops See Developers



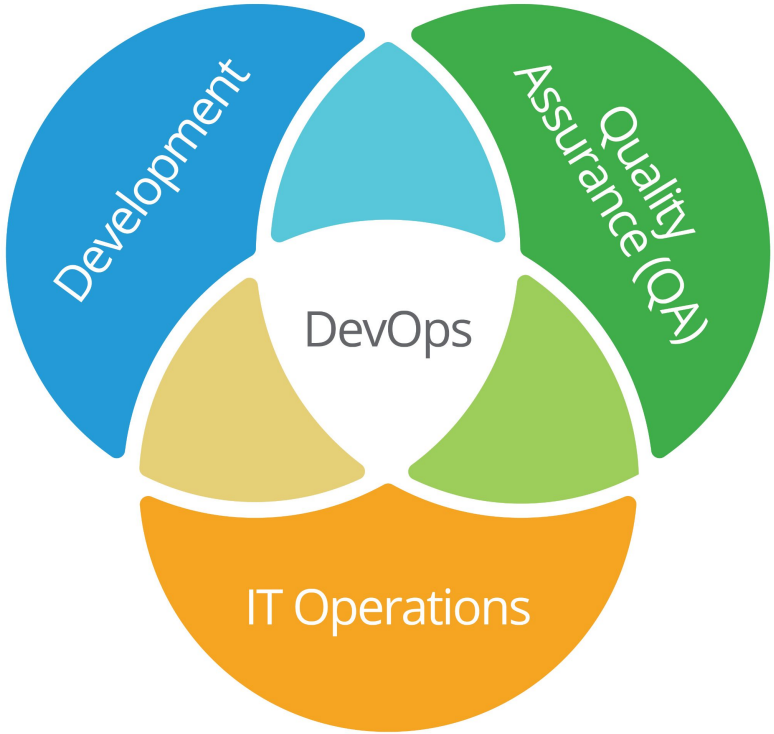
How Ops See Test Team

History of Devops

- Patrick Debois, a Belgian consultant, project manager, and agile practitioner is one among the initiators of DevOps.
- **2007**: While consulting on a data center migration for the Belgium government, system administrator **Patrick Debois** becomes frustrated by conflicts between developers and system admins. He ponders solutions.
- **August 2008**: At the **Agile Conference in Toronto**, software developer **Andrew Shafer** posts notice of a “birds of a feather” session entitled “**Agile Infrastructure**”.
- Exactly one person attends: You guessed it, Patrick Debois. And he has the room to himself; thinking there was no interest in his topic, Andrew skips his own session! Later, Debois tracks down Shafer for a wide-ranging hallway conversation. Based on their talk, they form the **Agile Systems Administration Group**.
- **June 2009**: At the **O’Reilly Velocity 09** conference, **John Allspaw** and **Paul Hammond** give their now-famous talk entitled “**10 Deploys a Day: Dev and Ops Cooperation at Flickr**”. Watching remotely, Debois laments on Twitter that he is unable to attend in person. Paul Nasrat tweets back, “Why not organize your own Velocity event in Belgium?”
    a. 10+ Deploys per Day: Dev and Ops Cooperation at Flickr - 46 Minutes, June 23 2009
    b. This presentation helped in bring out the ideas for DevOps and resolve the conflict of Dev versus Ops:
        i. “It’s not my code, it’s your machines!”
        ii. "It’s not my machines, it’s your code!”
    c. Traditional thinking:
        i. Dev’s job is to add new features.
        ii. Ops’ job is to keep the site stable and fast.
    d. **Ops’ job is NOT to keep the site stable and fast!**
    e. **Ops’ job is to enable the business.**
    f. **The business requires change, but change is the root cause of most outages!**
    g. You have two options:
        i. Discourage change in the interests of stability, or
        ii. **Allow change to happen as often as it needs to.**
    h. **Lowering risk of change through tools and culture.**
    i. **Ops** who think like devs, **Dev**s who think like ops!
- **October 2009**: Debois decides to do exactly that—but first, he needs a name. He takes the first three letters of development and operations, adds the word “days,” and calls it **DevOpsDays**. The conference doors open on October 30 to an impressive collection of developers, system administrators, toolsmiths, and others. When the conference ends, the ongoing discussions move to Twitter. To create a memorable hashtag, Debois shortens the name to #DevOps. And the movement has been known as **DevOps** ever since.
- If you need to choose one author to learn DevOps culture and practices, you must read Gene Kim's books

Define Devops

- **DevOps** is short for **Dev**elopment and **Op**eration**s**. It concentrates on collaboration between developers and other parties involved in building, deploying, operating, and maintaining software systems.
- DevOps is a **software engineering culture** that unites the development and operations team under an umbrella of tools to automate every stage.
- **DevOps is a set of cultural norms and technical practices that enable this fast flow of work from dev through test through operations while preserving world class reliability.** - Gene Kim



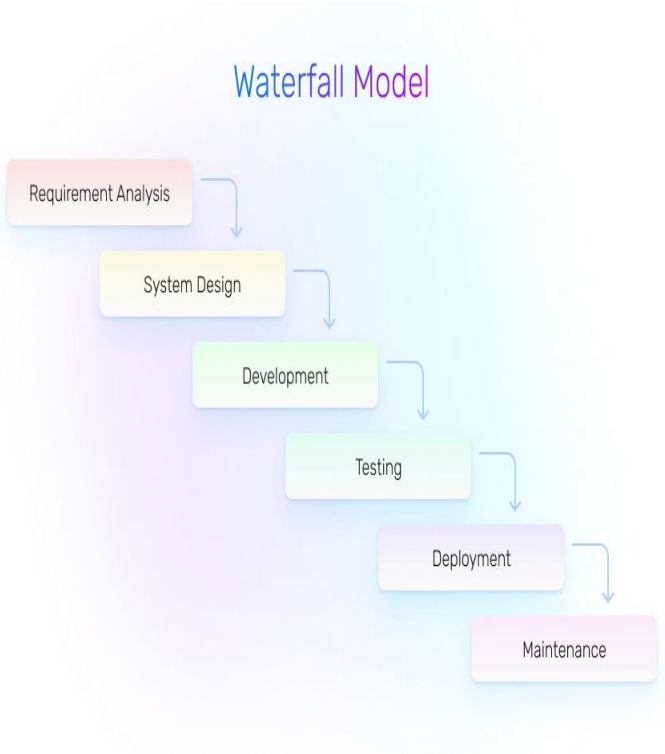
1. SDLC models
2. Lean DevOps
3. Agile
4. ITIL

1. Types of SDLC models

- Waterfall
- Iterative
- Spiral
- V-shaped
- Agile

Waterfall model

Advantages	Disadvantages
Simple to use and understand	The software is ready only after the last stage is over
Development stages go one by one	Not the best choice for complex and object-oriented projects
Easy to classify and prioritize tasks	Integration is done at the very end, which does not give the option of identifying the problem in advance

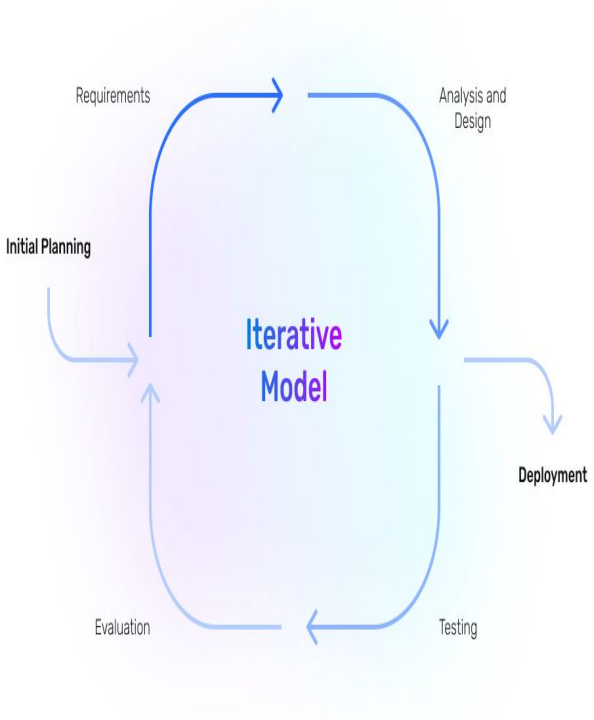


Use cases for the Waterfall SDLC model:

- The requirements are precisely documented
- Product definition is stable
- The technologies stack is predefined, which makes it not dynamic
- No ambiguous requirements
- The project is short

Iterative Model

Advantages	Disadvantages
The paralleled development can be applied	Constant management is required
The shorter iteration is – the easier testing and debugging stages are	Bad choice for the small projects
Flexibility and readiness to the changes in the requirements	Risks analysis requires involvement of the highly-qualified specialists

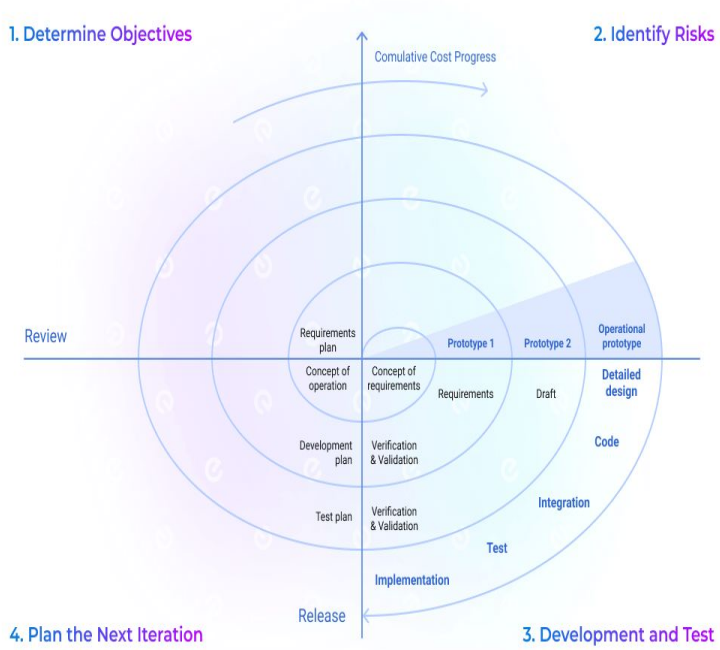


Use cases for the Iteration model:

- The requirements for the final product are clear from the beginning
- The project is large and includes complex tasks
- The main task is predefined, but the details may change in the process

Spiral Model

Advantages	Disadvantages
The development process is precisely documented yet scalable to the changes	The risk control demands involvement of the highly-skilled professionals
The scalability allows to make changes and add new functionality even at the relatively late stages	Can be ineffective for the small projects
The earlier working prototype is done – sooner users can point out the flaws	Big number of the intermediate stages requires excessive documentation

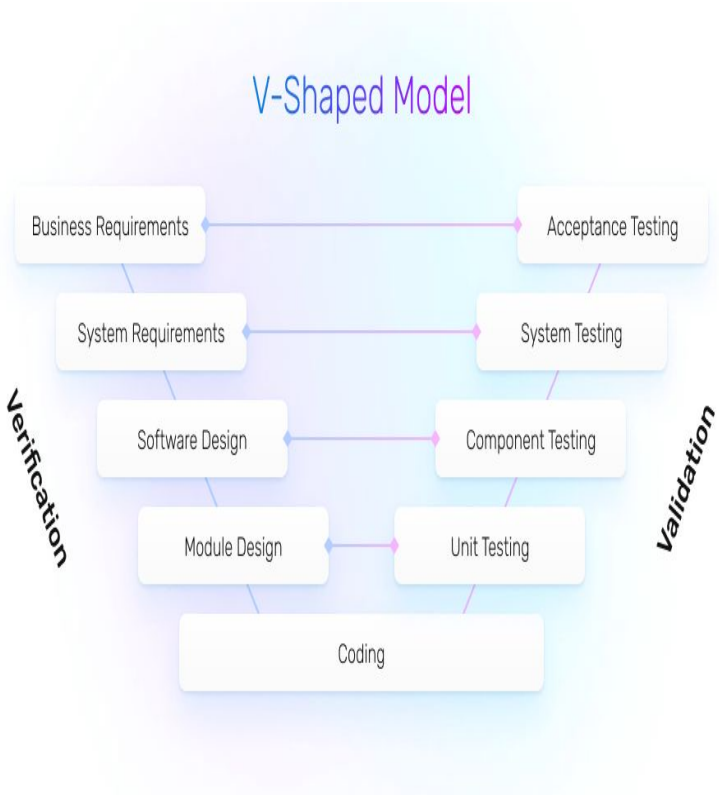


Use cases for the Spiral model

- The customer isn't sure about the requirements
- Significant edits are expected during the software development life cycle
- Risk management is highly essential for the project

V-Shaped Model

Advantages	Disadvantages
Every stage of V-shaped model has strict results so it's easy to control	Lack of the flexibility
Testing and verification take place in the early stages	Bad choice for the small projects
Good for the small projects, where requirements are static and clear	Relatively big risks

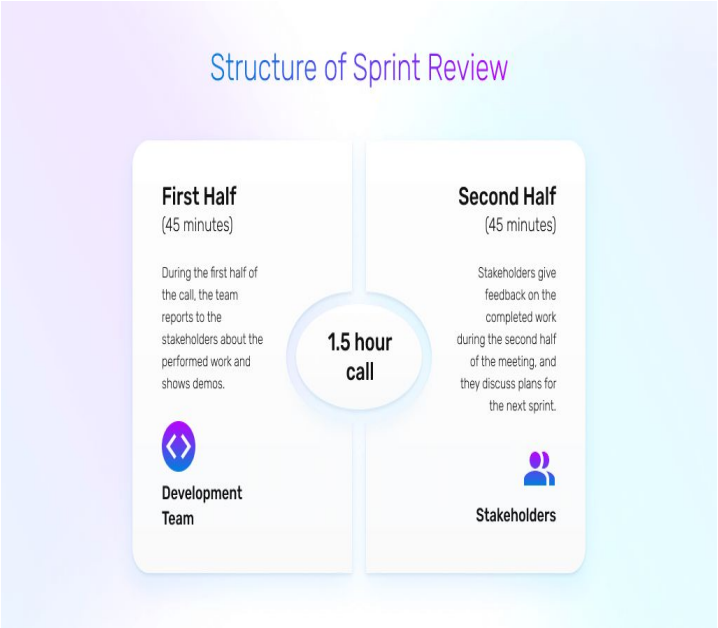


Use cases for the V-shaped model:

- For the projects where accurate product testing is required
- For the small and mid-sized projects, where requirements are strictly predefined
- The engineers of the required qualification, especially testers, are within easy reach

3. Agile

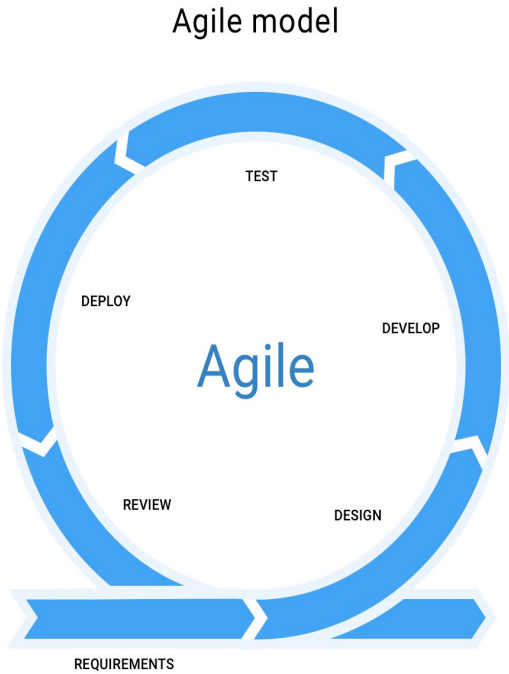
Agile is a philosophy, not a specific development approach.



1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts

Agile Model

Advantages	Disadvantages
Corrections of functional requirements are implemented into the development process to provide the competitiveness	Difficulties with measuring the final cost because of permanent changes
Project is divided by short and transparent iterations	The team should be highly professional and client-oriented
Risks are minimized thanks to the flexible change process	New requirements may conflict with the existing architecture
Fast release of the first product version	With all the corrections and changes there is possibility that the project will exceed expected time



Use cases for the Agile model:

- The users' needs change dynamically
- Less price for the changes implemented because of the numerous iterations
- It requires only initial planning to start the project

2. Lean DevOps

Lean DevOps is a development philosophy that focuses on both velocity and simplicity via a combination of culture, best practices, and workflow automation tools.

Why Devops?

The fundamental role of a DevOps engineer is to bridge communication the gap between the development and operations teams, which were originally working in silos.

Devops Stakeholders

Dev Includes all people involved in developing software products and services including but not exclusive to:

- Architects,
- business representatives,
- customers,
- product owners,
- project managers,
- quality assurance (QA),
- testers and analysts, suppliers …

Ops Includes all people involved in delivering and managing software products and services including but not exclusive to:

- Information security professionals,
- systems engineers,
- system administrators,
- IT operations engineers,
- release engineers,
- database administrators (DBAs),
- network engineers,
- support professionals,
- third party vendors and suppliers…

Devops Goals

Ensures effective collaboration between teams

Creates scalable infrastructure platforms

Builds on-demand release capabilities

Provides faster feedback

Important terminology

Artifact : Any description of a process used to create a piece of software that can be referred to, including diagrams, user requirements, and UML models.

Build Agent : A type of agent used in continuous integration that can be installed locally or remotely in relation to the continuous integration server. It sends and receives messages about handling software builds.

Commit : A way to record the changes to a repository and add a log message to describe the changes that were made.

Configuration Management : A process for establishing and maintaining consistent settings of a system. These solutions also include SysAdmin tools for IT infrastructure automation (e.g. Chef, Puppet, etc.).

Containerization : Resource isolation at the OS (rather than machine) level, usually (in UNIX-based systems) in user space. Isolated elements vary by containerization strategy and often include file system, disk quota, CPU and memory, I/O rate, root privileges, and network access. Much lighter-weight than machine-level virtualization and sufficient for many isolation requirement sets.

Continuous Delivery : A software engineering approach in which continuous integration, automated testing, and automated deployment capabilities allow software to be developed and deployed rapidly, reliably, and repeatedly with minimal human intervention.

Continuous Deployment : A software development practice in which every code change goes through the entire pipeline and is put into production automatically, resulting in many production deployments every day. It does everything that Continuous Delivery does, but the process is fully automated, and there's no human intervention at all.

Continuous Integration : A software development process where a branch of source code is rebuilt every time code is committed to the source control system. The process is often extended to include deployment, installation, and testing of applications in production environments.

Continuous Testing : The process of executing unattended automated tests as part of the software delivery pipeline across all environments to obtain immediate feedback on the quality of a code build.

Deployment Pipeline : A deployment pipeline is an automated manifestation of your process for getting software from version control into the hands of your users.
Event-Driven Architecture : A software architecture pattern where events or messages are produced by the system, and the system is built to react, consume, and detect other events.

Infrastructure-as-a-Service (IaaS) : A self-service computing, networking, and storage utility on-demand over a network.

Microservices Architecture : The practice of developing software as an interconnected system of several independent, modular services that communicate with each other.

Pair Programming : A software development practice where two developers work on a feature, rather than one, so that both developers can review each others' code as it's being written in order to improve code quality.

Production : The final stage in a deployment pipeline where the software will be used by the intended audience.

Staging : Used to test the newer version of your software before it's moved to live production.

Rollback : An automatic or manual operation that restores a database or program to a previously defined state.

Rolling update : a process of smooth updates for an app without any downtime, performed instance by instance. It uses Kubernetes to ensure uninterrupted app availability and positive user experience.

Source Control : A system for storing, tracking, and managing changes to software.

Test Automation : The use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes.

Virtual Machine (VM) : A software emulation of a physical computing resource that can be modified independent of the hardware attributes.

Bastion host : a special server used to access private networks and withstand hacker attacks.

Backup : a process of copying the important data to provide the reserve copy and enable restoration on demand, as well as the result of the backup process, an archive with files.

Build : a specific version of program code, mostly referred to as the stage of new feature development. The most important builds are Canary builds, where the new code is tested for compliance with the existing app functionality in the production environment before being shipped to the customers.

Bare-metal : the case when the software is installed on the physical devices (hard disks), omitting the virtualization layer.

Cluster : a set of interconnected instances (bare-metal servers, virtual machines, Kubernetes pods, etc.) that are treated as a single entity to enable load balancing, auto-scaling, and high availability.

Cron job : a scheduled process that will run a certain script on a server at the certain time.

IaC : Infrastructure as Code — one of the nimble superpowers of DevOps. It means that infrastructure configuration is done with machine-readable declarative files, not manually or using interactive tools. These files (like Kubernetes or Terraform manifests) can be stored in GitHub repositories, adjusted and versioned the same as code, thus providing efficient automation of infrastructure provisioning.

Orchestration : a practice of automating the IT tasks in the context of SOA, virtualization, environment provisioning. In short, it is a process of executing predefined tasks using predefined scripts executed with interactive tools like Terraform (which was built specifically for configuration orchestration purposes).

Source Code Management : Source code management (SCM) is **used to track modifications to a source code repository**. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. SCM is also synonymous with Version control.

Software Configuration Management: Software Configuration Management(SCM) is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

Hypervisor - type 2

Providers :

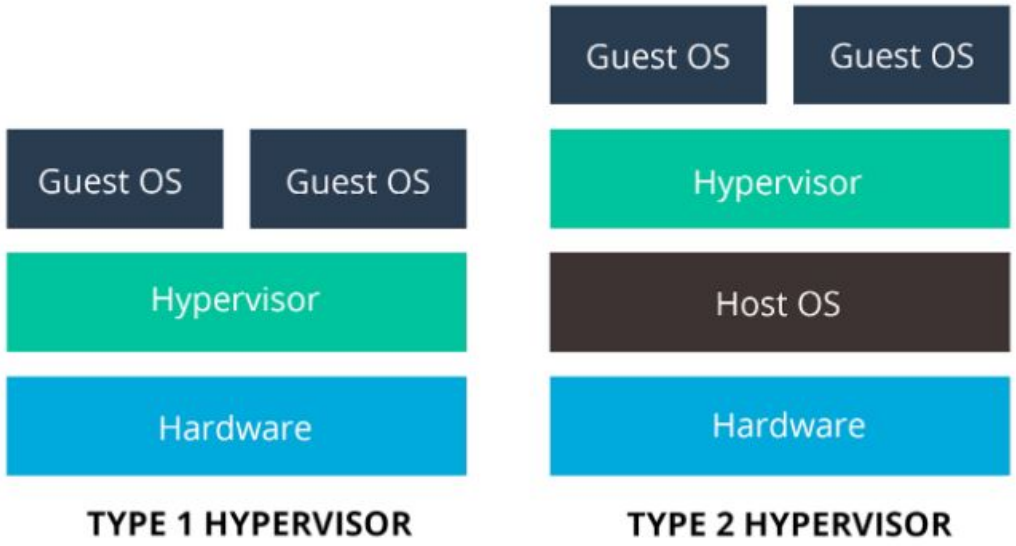
- Microsoft Virtual PC,
- Oracle Virtual Box,
- VMware Workstation,
- Oracle Solaris Zones,
- VMware Fusion,
- Oracle VM Server

Installation:

- Using any provider s/w

Linux commands:

- Basic commands



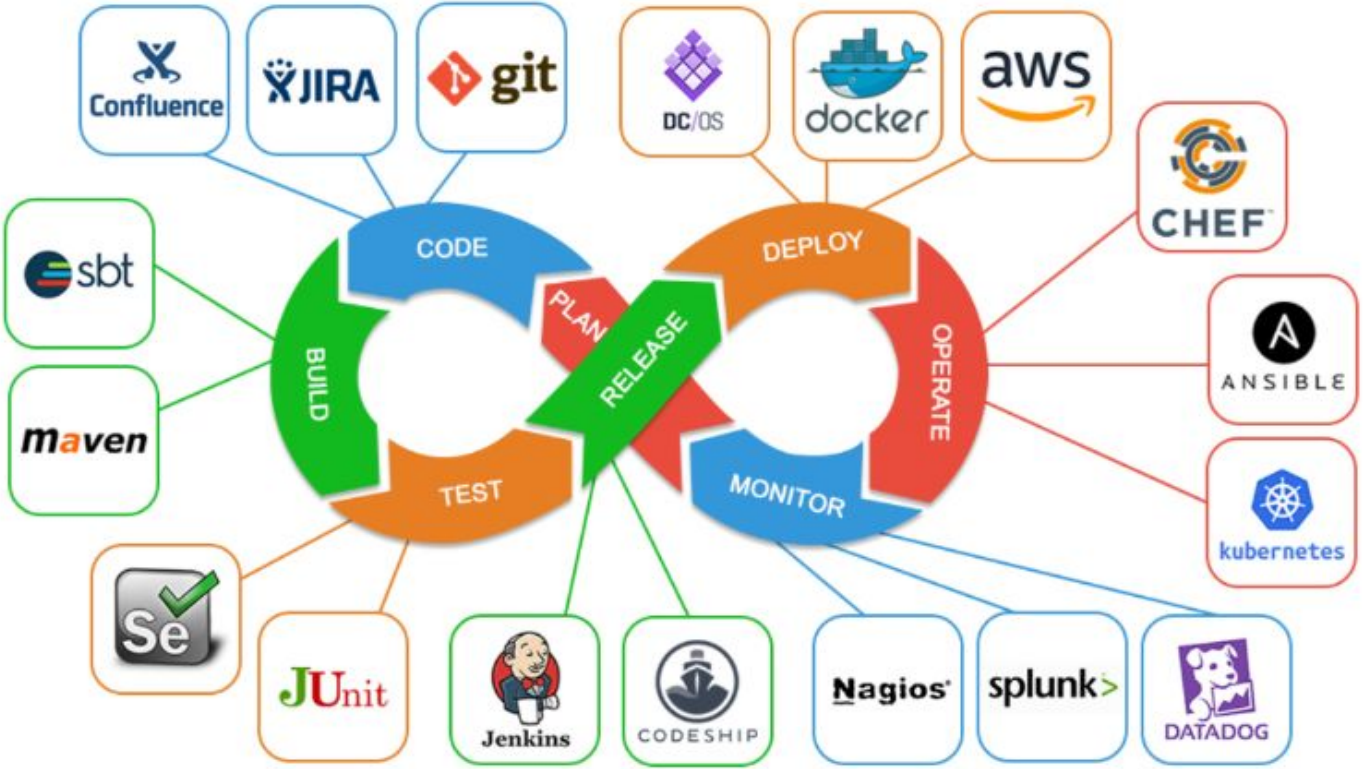
Devops perspective

DevOps is an IT mindset that encourages communication, collaboration, integration and automation among software developers and IT operations in order to improve the speed and quality of delivering software.

DevOps and Agile

- Satisfy customer through early and continuous delivery of valuable software.
- Deliver working software frequently with a preference for the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Replace non-human steps using tools.
- Improve the collaboration between all the teams.
- Automate to create a potentially shippable increment.

DevOps Tools



Configuration management

A process for establishing and maintaining consistent settings of a system.
These solutions also include SysAdmin tools for IT infrastructure automation (e.g. Chef, Puppet, etc.).

Continuous Integration and Deployment

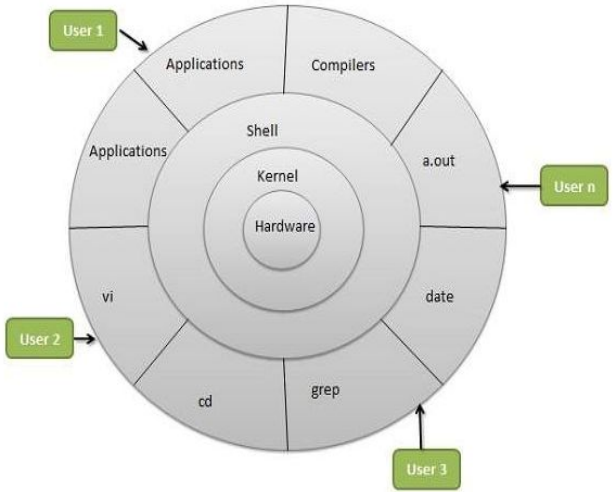
Continuous Integration : A software development process where a branch of source code is rebuilt every time code is committed to the source control system. The process is often extended to include deployment, installation, and testing of applications in production environments.

Continuous Deployment : A software development practice in which every code change goes through the entire pipeline and is put into production automatically, resulting in many production deployments every day. It does everything that Continuous Delivery does, but the process is fully automated, and there’s no human intervention at all.

Linux OS Introduction

Linux is one of popular version of UNIX operating System.
It is open source as its source code is freely available.
It is free to use.

Architecture



The architecture of a Linux System consists of the following layers:

Hardware layer – Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).

Kernel – It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.

Shell – An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.

Utilities – Utility programs that provide the user most of the functionalities of an operating systems.

Importance of Linux in DevOps

- Linux offers the DevOps team the flexibility and scalability needed to create a dynamic development process.

- Majority of enterprises today are running development projects which already have Linux supporting their operations.

Linux Basic Command Utilities - (DIY)

sudo	cp	free
uptime	kill	top
w	mv	tar
users	cat	grep
who	cd	pwd
whoami	tail	sort
ls	head	ssh
crontab	find	ftp
less	lsof	sftp
more	last	**man**
mkdir	ps	netstat

Linux Administration

Duties:

- Maintain all internet requests inclusive to DNS, RADIUS, Apache, MySQL, PHP.
- Taking regular back up of data, create new stored procedures and listing back-up is one of the duties.
- Analyzing all error logs and fixing along with providing excellent customer support for Webhosting, ISP and LAN Customers on troubleshooting increased support troubles.
- Communicating with the staff, vendors, and customers in a cultivated, professional manner at all times has to be one of his characteristics.
- Enhance, maintain and creating the tools for the Linux environment and its users.
- Detecting and solving the service problems ranging from disaster recovery to login problems.
- Installing the necessary systems and security tools. Working with the Data Network Engineer and other personnel/departments to analyze hardware requirements and makes acquiring recommendations.
- Troubleshoot, when the problem occurs in the server.

Environment Variables

Linux or UNIX environment variable is nothing but a name and the same name is holding some value or path.

The environment variable is providing the functionality to the end-user, how the application job will run on the Linux environment, and the custom behavior of the system

Syntax of Environment Variables

1. Set Environment Variable

    ```
    SET VARIABLE METHOD [ ENVIRONMENT NAME ] [ PATH ]
    ```

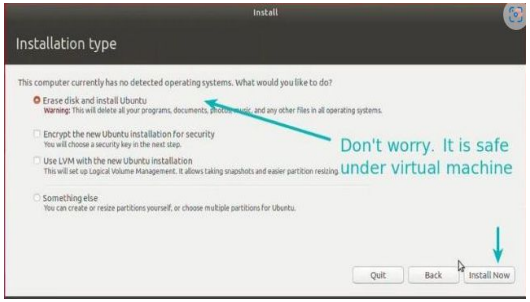
2. Call Environment Variable

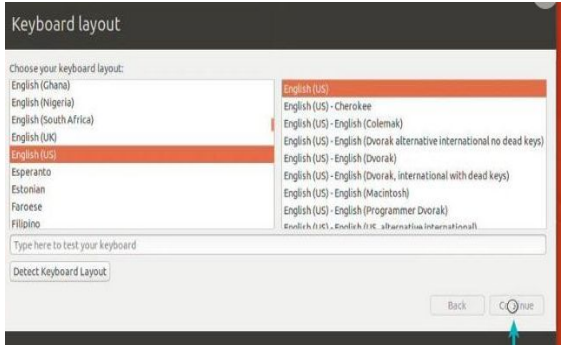
    ```
    echo $[ ENVIRONMENT NAME ]
    ```

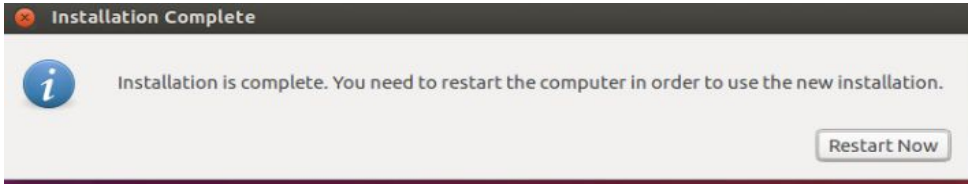
Networking

ifconfig	hostname
ip	dig
traceroute	nslookup
tracepath	route
ping	host
netstat	arp
whois	curl or wget

Linux Server Installation







RPM and YUM Installation

YUM(Yellow Dog **U**pdater, **M**odified**)** is the primary package management tool for installing, updating, removing, and managing software packages in Red Hat Enterprise Linux. YUM performs dependency resolution when installing, updating, and removing software packages.

RPM(RPM Package Manager → earlier **R**edHat **P**ackage **M**anager **)** is a popular package management tool in Red Hat Enterprise Linux-based distros. Using **RPM,** you can install, uninstall, and query individual software packages. Still, it cannot manage dependency resolution like YUM.

Chapter - 2
Version Control-GIT

2.1. Introduction to GIT

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

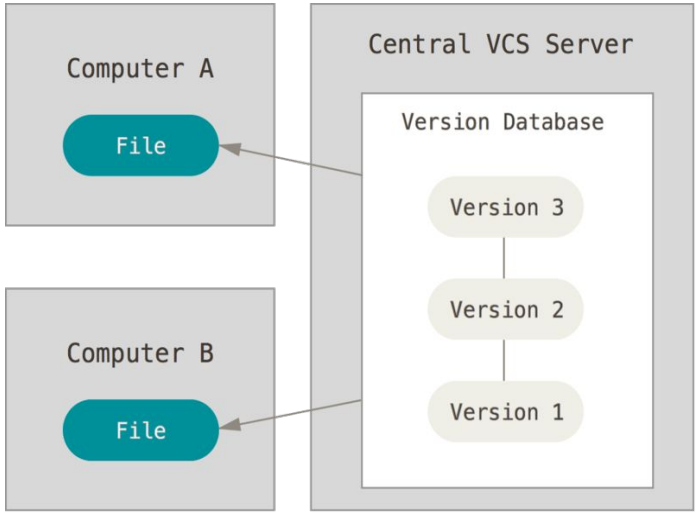
2.2. What is Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

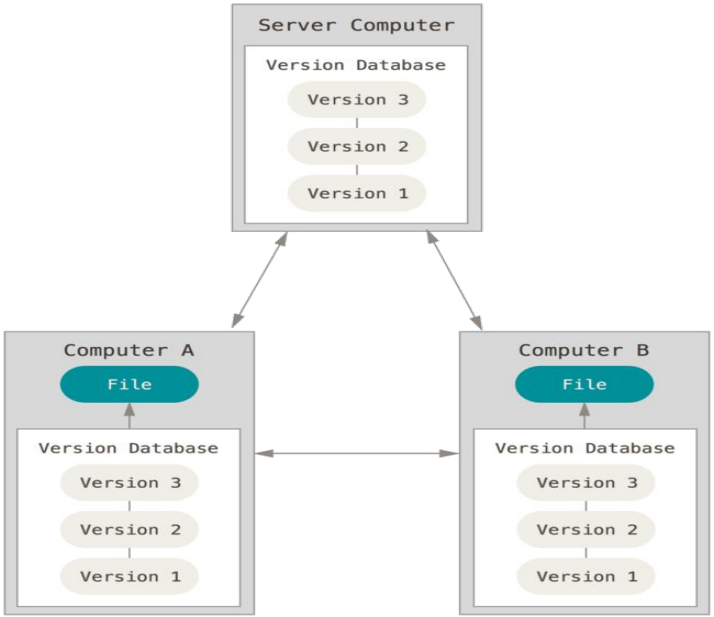
2.3. About Version Control System and Types

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Centralized Version Control Systems



Distributed Version Control Systems



2.4. Difference between CVCS and DVCS

CVCS	DVCS
A client need to get local copy of source from server, do the changes and commit those changes to central source on server.	Each client can have a local branch as well and have a complete history on it. Client need to push the changes to branch which will then be pushed to server repository.
Working on branches in difficult in CVS.	Working on branches is easier.
CVS system do not provide offline access.	Systems are workable offline as a client copies the entire repository on their local machine.
Slower as every command need to communicate with server	Faster as mostly user deals with local copy without hitting server every time
If CVS Server is down, developers cannot work	If DVS server is down, developer can work using their local copies

2.5. A short history of GIT

The Linux kernel is an open source software project of fairly large scope. During the early years of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files. In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.

In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked. This prompted the Linux development community (and in particular **Linus Torvalds,** the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper. Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities. It's amazingly fast, it's very efficient with large projects, and it has an incredible branching system for non-linear development

2.7. GIT Command Line

There are a lot of different ways to use Git.

There are the original command-line tools, and there are many graphical user interfaces of varying capabilities.
The command line is the only place you can run **all** Git commands — most of the GUIs implement only a partial subset of Git functionality for simplicity.

If you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true. Also, while your choice of graphical client is a matter of personal taste, **all** users will have the command-line tools installed and available.

2.8. Installing Git

Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution. If you're on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use `dnf`:

```
$ sudo dnf install git-all
```

If you're on a Debian-based distribution, such as Ubuntu, try `apt`:

```
$ sudo apt install git-all
```

2.9. Installing on Linux

Use the flowing command to install

$ sudo apt install git

2.10. Installing on Windows

There are also a few ways to install Git on Windows.

The most official build is available for download on the Git website.

Just go to https://git-scm.com/download/win and the download will start automatically.

Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to https://gitforwindows.org.

2.11. Initial setup

Now that you have Git on your system, you'll want to do a few things to customize your Git environment.
Git comes with a tool called git config that lets you get and set configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places:

1. [path]/etc/gitconfig file: Contains values applied to every user on the system and all their repositories. If you pass the option --system to git config, it reads and writes from this file specifically.
2. ~/.gitconfig or ~/.config/git/config file: Values specific personally to you, the user. You can make Git read and write to this file specifically by passing the --global option, and this affects **all** of the repositories you work with on your system.
3. config file in the Git directory (that is, .git/config) of whatever repository you're currently using: Specific to that single repository. You can force Git to read from and write to this file with the --local option, but that is in fact the default. Unsurprisingly, you need to be located somewhere in a Git repository for this option to work properly.

On Windows systems, Git looks for the .gitconfig file in the $HOME directory (C:\Users\$USER for most people).
You can view all of your settings and where they are coming from using: **$ git config --list --show-origin**

Your Identity

```
$ git config --global user.name "Harry Potter"
```

```
$ git config --global user.email hpotter@gmail.com
```

Your Editor

```
$ git config --global core.editor emacs
```

2.13. Creating repository

When starting out with a new repository, you only need to do it once; either locally, then push to GitHub, or by cloning an existing repository.

$ git init

Turn an existing directory into a git repository

$ git clone [url]

Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

2.14. Cloning, check-in and committing

$ git clone [url]

Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

$ git checkout [branch-name]

Switches to the specified branch and updates the working directory

$git add [file]

Snapshots the file in preparation for versioning

$ git commit -m "[descriptive message]"

Records file snapshots permanently in version history

2.15. Fetch pull and remote

$ git fetch

Downloads all history from the remote tracking branches

$git push

Uploads all local branch commits to GitHub

$ git pull

Updates your current local working branch with all new commits from the corresponding remote branch on GitHub.

2.16. Branching

Branches are an important part of working with Git. Any commits you make will be made on the branch you're currently “checked out” to. Use git status to see which branch that is.

2.17. Creating the Branches, switching the branches, merging

Create branch

$ git branch [branch-name]

Switch the branch

$git checkout [branch-name]

Merge the branch

$ git merge [branch]

Linux Hands ON

working with directories

pwd
cd
cd ~
cd ..
cd -
absolute and relative paths

ls
ls -a
ls -l
ls -lh
Mkdir
mkdir -p
Rmdir
rmdir -p

working with files

touch
rm
rm -i
rm -rf
Cp
cp -r
Mv

filters

$ grep

Example : $ grep Bel tennis.txt

$ cut

Example : cut -d: -f1,3 /etc/passwd | tail -4

$ tr

Example : cat tennis.txt | tr 'e' 'E'

$ wc

$ wc tennis.txt

sort

uniq (With uniq you can remove duplicates from a sorted list)

$ sort music.txt |uniq -c

basic Unix tools

~$ find
~$ date
~$ cal
~$ sleep 5
~$ time date
 gzip - gunzip

users

Identify yourself:
~$ **whoami**
useradd eg: # useradd -D
userdel eg: # userdel -r amit
usermod eg:
passwd

switch users with su
su as root
$ su root **$ su serena**
su - $username
$ su - harry
When no username is provided to su or su -, the command will assume root is the target.
$ su -
Password:

Linux Networking Commands

Ifconfig Command	`$ ifconfig`
Ping Command	`$ ping 192.168.0.103`
Traceroute Command	`$ traceroute 216.58.204.46`
Route Command	`$ route`
Nmcli Command	`$ nmcli dev status`
Netstat Command	`$ sudo netstat -tnlp`
host Command	`$ host google.com`
NSLookup Command	`$ nslookup 216.58.208.174`
	`$ nslookup google.com`
Tcpdump Command —>	`$ tcpdump -i eth1`

Chapter - 5
Build Tool Maven

An Introduction

Agenda

- What is Maven?
- What Maven can do?
- Advantages over Ant
- Archetypes
- Plug-ins
- Goals
- Project Object Model
- Project Coordinates
- Build Life Cycle
- Simple Project-The Ant way
- Simple Project-The Maven way
- Building More than 1 project

What is Maven??

As per http://maven.apache.org/-

Apache Maven is a software project management and comprehension tool. Based on the concept of a **Project Object Model (POM)**, Maven can manage a project's build, reporting and documentation from a central piece of information.

"BUT IT IS NOT A MERE BUILD TOOL"

What Maven Can Do?

- Model our software project
- Centralize project information
- Manage our build process
- Gather data about our software project and the build itself
- Document the software, and our project

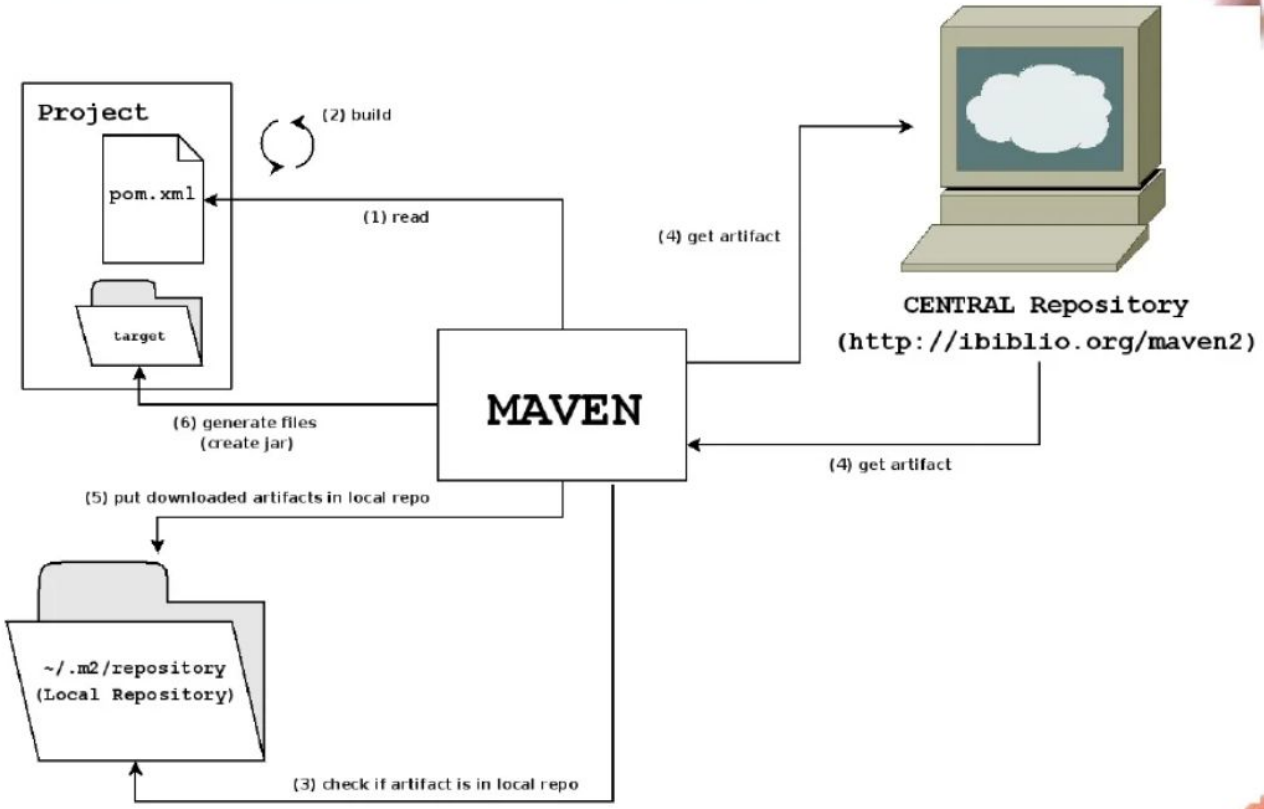
Advantages over Ant

- Eliminate the hassle of maintaining complicated scripts
- All the functionality required to build your project , i.e. , clean, compile, copy resources, install, deploy etc is built right into the Maven
- Cross Project Reuse- Ant has no convenient way to reuse target across projects, But Maven does provide this functionality

Advantages over Ant...

- Logic and Looping-Maven provides conditional logic, looping constructs and reuse mechanisms which were missing in Ant

How Maven Works?



What's all this Jargon??

- POM?
- Goal??
- Plugin???
- Artifact????
- Archetype?????

Archetype

- **In general** - a primitive type or an original model or pattern from which all other things of same kind are made.
- **For Maven** - a template of a project which is combined with some user input to produce a working Maven project that has been tailored to the user's requirements
- There are hundreds of archetypes to help us get started

Plugins

- A plugin provides a set of goals that can be executed to perform a task
- There are Maven plugins for building, testing, source control management, running a web server, generating Eclipse project files, and much more
- Some basic plugins are included in every project by default, and they have sensible default settings.

Goals

- A goal is like an ant target.
- We can write our own plug-ins with goals or use those provided by Maven
- For example, a Java project can be compiled with the compiler - plugin's compile-goal by running

mvn compiler:compile.

Project Object Model

- Fundamental unit of work in Maven.
- It is an XML file that contains information about the project and configuration details used by Maven to build the project.
- project.xml in Maven 1
- pom.xml in Maven 2 and onwards.

POM....(Contd)

- build directory, source directory, the test source directory, and so on.
- The goals or plugins
- Project Co-ordinates(GAV)
- Build Phases
- Project Type(Packaging-JAR, WAR,EAR)
- Dependencies

POM...(Contd)

- Minimal pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<version>1</version>
</project>
```

Coordinates..Whats that??

- Every Maven project is identified by its GAV co-ordinates
- GroupId(G)- Its an identifier for a collection of related modules. This is usually a hierarchy that starts with the organization that produced the modules, and then possibly moves into the increasingly more specialized projects. e.g com.abc.xyzproduct

Here comes Artifact

- Artifact(A)-Its a unique identifier or a simple name that is given to a module within a group.
- There can be multiple modules in a group

e.g. if a group has a module called webapp, its artifactId will be “webapp”

Version is version

- Version(V)- Its an identifier for the release or build number of project

e.g. 1.0-SNAPSHOT

Now what's this SNAPSHOT??

Version...(Contd)

- There are two types of versions
- Releases- a version that is assumed never to change. Only to be used for a single state of the project when it is released and then updated to the next snapshot.
- Snapshot- used by projects during development as it implies that development is still occurring that project may change

Build Life Cycle

- In Maven, the build is run using a predefined, ordered set of steps called the build life-cycle
- The individual steps are called phases, and the same phases are run for every Maven build using the default life-cycle, no matter what it will produce.
- The build tasks that will be performed during each phase are determined by the
- configuration in the project file, and in particular the selected packaging.

Build Life-Cycle...(Contd)

- Some of the most commonly used life-cycle phases in the default life-cycle are:
- **Validate** — checks build prerequisites
- **Compile** — compiles the source code identified for the build
- **Test** — runs unit tests for the compiled code
- **Package** — assembles the compiled code into a binary build result
- **Install** — shares the build with other projects on the same machine
- **Deploy** — publishes the build into a remote repository for other projects to use

All Said..
Now Let's Get Started...

Simple Project

- We have a simple Hello World Application to build.

A Simple Project: The Ant Way

- Assumption- Ant environment has been setup
- Now let us add following lines of code in a java file

```
final Logger logger =
LoggerFactory.getLogger(ExampleAction.class);
  public boolean execute() {
      logger.info( "Example action executed" );
      return true;
  }
```

A Simple Project: The Ant Way

- Visit the Logger Library's Homepage, find Jar's download link
- Download the Jar
- Copy the Jar files into your lib directory or classpath
- Create your build.xml file
- In build.xml file
  - Tell ant about compilation process- where to put compilation files, where java files are located
  - Tell about how to run the app
  - Tell ant about your all libraries, i.e. JAR files and their location
  - If those JARs have any further dependencies, mention those too
  - Tell ant where to put configuration files, if any
  - Tell ant where to put resources, if any

A Simple Project: The Maven Way

- Download Maven from http://maven.apache.org
- Unzip the contents in a local directory say /usr/local/maven or C:\maven
- Set environment variables MAVEN_PATH=<dir where tarball was extracted>
- PATH=MAVEN_HOME/bin(mind '/' in case of linux)
- To generate a sample project, all you need to run is

>mvn archetype:generate

- Choose the desired option from hundreds of templates provided.

  For a primitive project , we can choose **archetype-quick-start**

A Simple Project: The Maven Way

- Now we'll be prompted for GAV co-ordinates for the project and after providing those, we have directory structure ready-

A Simple Project: The Maven Way

- Now let us add following lines of code in a java file

```
final Logger logger =
LoggerFactory.getLogger(ExampleAction.class);
  public boolean execute() {
      logger.info( "Example action executed" );
      return true;
  }
```

A Simple Project: The Maven Way

- We have to add just these lines to our pom.xml and we are done

```
[...]
 <dependencies>
  [...]
  <dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-api</artifactId>
   <version>1.5.0</version>
  </dependency>
 </dependencies>
```

A Simple Project: The Maven Way

- With the dependency in place, let's try to compile the project :

simple-webapp> mvn compile

- If all things go well, required libraries will be downloaded and the build will succeed. The generated class files will be produced in the target/classes subdirectory.

A Simple Project: The Maven Way

- Let's try packaging the artifact:

```
simple-webapp$ mvn package
```

- This process will follow the same steps as it did before, but with the addition of the new class file and the dependency to the resulting web application.
- Firstly, the compiled sources are placed in target/simple-webapp/classes
- Secondly, the dependency added has been copied to target/simple-webapp/lib.
- This is because the Maven WAR plugin now knows that it will be required to run the web application by reading the project's dependencies. Of course, these files are also packaged into the resulting WAR file.

Thanks