Software Defined Computing BSC10124

Group A & B

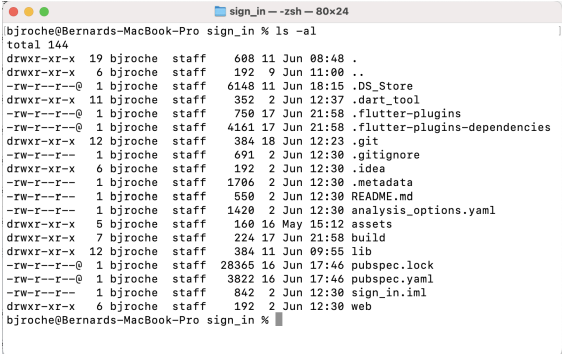
Assignment 1

(Please check Moodle for the submission date)

Emmanuel Abiodun Adeniyi

Student no. 71739

**Question 2 (25%): Understanding the computing Environment**



The image presents a standard Unix-like file system hierarchical structure. This forms the basis of how those operating systems organize, and later manage, data. Now, let me break down the elements important to the discussion and their significance:

The file system then starts with the root directory, which is represented by a single forward slash (/). It illustrates a level of a directory from which other directories and files are branched. It is that point through which one starts to navigate the whole file system.

Directly under the root we have a number of standard directories that serve specific purposes:

1. /bin: It holds the very basic executables of binary files that are needed for minimal functionality to be able to boot up the system or reach single-user mode.

2. /etc: This directory integrates the system-wide configuration files. The most important reasons are system administration and customization.

3. /home: This is normally where all user home directories reside. Each user will usually have a subdirectory from this directory for their personal files and settings.

4. /var: Variable data goes here, spool files, log data, transient temporary files. This directory is anticipated to grow in size, and its contents are expected to be changed often.

5. /usr holds user programs and data. It'll probably also have a secondary hierarchy under it, with its own bin, lib, and share directories.

It has separation of concerns: system files, user files, and variable data are kept in different places, which is good for housekeeping and security. This also makes system maintenance and backups easier. What makes this file system efficient at keeping its contents organized and easy to navigate is its hierarchical nature. There are paths to enable the user or the program to find files; these can be thought of as instructions through such a tree-like structure.

**Question 3 (25%): Lessons learned**

Tool selected: Git



Explanation:

Git is a distributed version control system developed to handle projects of any size efficiently and at merchants of speed. It was created by Linus Torvalds in 2005 for development regarding the Linux kernel. Git allows multiple developers to work on the same project simultaneously.

Essentially, Git keeps a record of changes to your files. It does this by taking snapshots of all your project contents at any particular point in time. It calls these snapshots "commits". Every commit is like a landmark in the history of the project that you are able to go back to if need be.

One of the core Git principles is a development model using branches that let developers stray away from the main line of development and work on features independently of the main codebase. These branches can later be merged back into the main line when the feature is complete.

Git is local. Except in a few operations, almost everything happens locally on your machine. In fact, this is a major advantage: Git's speed and the fact that it allows you to work offline. Once you are ready to collaborate, you will "push" your changes to a remote repository, usually hosted on services like GitHub or GitLab.

Advantages:

Distributed: Each developer gets a full copy of the repository, including its entire history. It allows working offline and also serves as a backup of the entire project onto every developer's machine.

Branching and merging: The model of branching for Git is light, which enables the creation of feature branches where experimentation is easy with a view to merging changes.

Speed: Because most of the operations Git does are local, it is much quicker than any centralized version control system.

Data integrity: This is because Git uses SHA-1 hashes to check the integrity of all your files and their history, ensuring that none of your data has been damaged or tampered with in any way.

Flexibility: Git supports a wide range of workflows, from simple linear development to complex branching strategies used by large teams.

Disadvantages:

Learning curve: For beginners, Git can be problematic. There are many different commands and concepts to know.

Binary files: Although Git supports the management of binary files, it is not exactly optimized for these El pesado archivo binario grande puede aumentarToo large binary files can therefore bloat the repository size.

No built-in access control: Git itself doesn't have user management or access control features, requiring additional tools for these functions.

The flexibility of Git makes it possible to handle very complex workflows, which can become confusing if they are not well-managed.

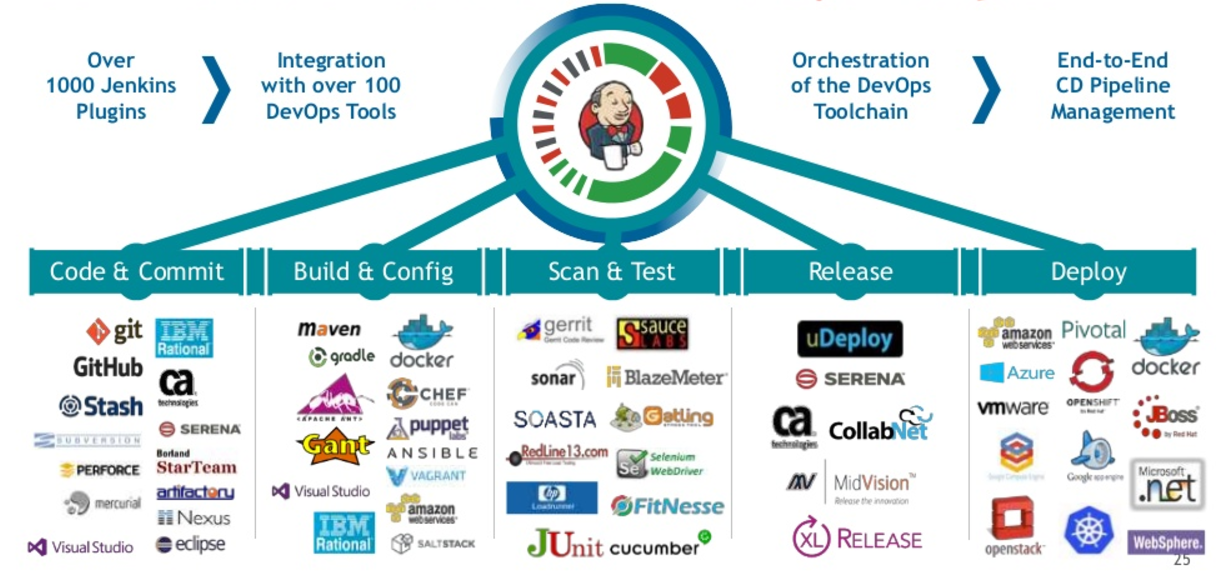
Personal Reflection:

Git has been instrumental in my student and aspiring software development work. It was serving to keep track of changes in my coursework and personal projects, providing that safety where I know I can experiment until my heart's content and revert if I need to. I use Git to handle version control in my programming assignments and group projects.

The ability to collaborate or share code, and also manage possible conflicts if we work on the same files, really helps when working with classmates. More specifically, the feature of branching really helped me in that it allows me to try a different way without touching the main codebase for a problem-resolution attempt. Looking into the future, I believe Git will benefit me further when I hit the professional world. Many software development teams make use of Git as a version control system; therefore, familiarization with it is a godsend.

I will use it once more for big, complex projects, probably integrating it with continuous integration and deployment pipelines. All my projects and contributions would show on my GitHub profile to an employer. The more that I learn, grow in the profession. Git is an even more deeply integrated tool not only for handling code but also for keeping project histories, facilitating review and collaboration among developers, and doing clean software engineering practices.

**Question 4 (25%):**



**Terraform:**

Terraform is an open-source infrastructure-as-code tool developed by HashiCorp. Imagine yourself as an architect, except instead of designing houses, you design cloud infrastructure. Terraform can be thought of as a magical blueprints set that does even more than describe the infrastructure—it will build it for you.

In Terraform, you write code that describes the state you want for your infrastructure: which cloud resources you want to exist, how they should be configured, and how they should interact. This code acts as a recipe for your infrastructure. You run Terraform, it reads this recipe, and makes it real by creating or modifying your cloud resources per your description.

The beauty of Terraform is that it can run infrastructures scattered across many cloud providers. It is almost a universal remote control for the cloud. Be it AWS, Google Cloud, Azure, or any combination thereof, Terraform speaks their language and can orchestrate them all.

One could think of Terraform as a very productive automated Lego builder. You provide the instructions in your code, and Terraform snaps the pieces—your infrastructure—together according to your specification, however complex the design may be.

**Docker:**

Docker merely represents a development platform for installing, shipping, and running applications in containers. If traditional applications were full-sized houses, Docker containers would certainly be tiny homes—compact, self-contained, portable.

Containers are a means of standardizing package applications and their dependencies into one constructs in software development. Entailing from this, such a-container will run anywhere computing environment are available without any changes to the application; thus, its fine execution anywhere it may be deployed is assured.

Think of Docker containers like standard delivery containers for software. Much similar to how shipping containers transformed world

trade by creating a simple, worldwide standard for moving goods around, so too do Docker containers standardize

software deployment, easing development, shipping, and running applications anywhere consistently.

Analogously, traditional applications are like food that has to be prepared from scratch in every restaurant, or environment. Docker containers are more like TV dinners—pre-packaged, ready to heat and serve, and guaranteed to taste the same no matter where you eat them.

Continuous Integration and Continuous Delivery/Deployment (CI/CD)

CI/CD is a framework of practices in software development that targets increasing the effectiveness and reliability of the process for constructing, testing, and deploying software.

Continuous integration is a little like having a proofreader for your code. It means that, every time you make any kind of change, it's checked against the rest of the codebase to make sure everything still works together harmoniously. In other words, you have something like a team of robots working on building and testing your product all the time for catching issues very early.

Well, Continuous Delivery does a bit more. You can think about it as similar to having a conveyor belt that apart from simply assembling and testing your product also packages the result ready for shipping. Any time you want you can press a button and send out the latest version of your software.

The final evolution would be one of Continuous Deployment, where the conveyor belt does not simply ready your product for shipment but sends it out to customers as soon as the product has passed all tests.

Analogically, one would think of CI/CD as something very much like a very efficient car factory assembly line where raw materials go in one end and come out of the other end, in this case, source code changes, through the assembling and rigorous testing at each stage to a fully functional and street-ready vehicle with very little human intervention.

**DevOps:**

DevOps is a cultural and professional movement that seeks to tear down traditional silos between software development teams (Dev) and IT operations teams. This was like knocking down the wall between the kitchen and dining section of a restaurant, enabling chefs and waiters to collaborate with each other more effectively.

In the DevOps model, development and operations teams collaborate during the entire lifecycle of software development, from design and development to testing and deployment. Such collaboration will unwrap faster cycles of development with fewer errors and reliable releases of software.

DevOps is based on automation, continuous feedback, and shared responsibility; much like turning a software development process into a well-oiled machine where everyone becomes both mechanic and operator.

An analogy would be traditional software development being like a relay race, wherein the baton flowing gets passed from one team to another. DevOps turns this into a game of soccer where all the players are in the field together, working toward a common goal.

**SSH (Secure Shell):**

SSH is the cryptographic networking protocol designed to enable secure communication over an unsecured network. This is like a secure, encrypted tunnel one logs into for access or to transfer data to another computer on the internet. All data you send is already encrypted when using SSH, making it really hard for anyone to intercept and understand your communication.

In contrast, it is mostly used for remote command-line logins and the execution of remote commands, but it may also be used to tunnel, which enables the forwarding of TCP ports and X11 connections. Think of SSH as a very secure, invisible bridge between two computers. You can walk across this bridge to access the other computer, and no one can see you or what you're carrying. A useful analogy here might be considering SSH to be very much like a diplomatic pouch carrying sensitive government documents. The contents are secret and protected, but only the addressee opens the pouch. Exactly the same in this scenario is the 'secure channel' that SSH makes, accessible and understandable only to intended parties at both ends.

**References**

Day 8-Basic Git & GitHub for DevOps .(https://media.licdn.com/dms/image/D4D12AQFMqmbxwIiXbg/article-cover\_image-shrink\_720\_1280/0/1704773923924?e=2147483647&v=beta&t=nTRiM081ic2\_lDDS3Mr3xMB4qS\_LWlvdMP4DJCWVXKo)

DevOps Toolbox: Jenkins, Ansible, Chef, Puppet, Vagrant, & SaltStack (https://cdn.hostadvice.com/2018/03/devopsjenkins.png)