Date / /
Page L&W

* Introduction to version control systems (VCS)
A version control system (VCS) is a slw tool that helps a team of slw developers work together and maintain a complete history of their work. It's essentially a system that records changes to a file or set of files over time so that you can recall specific versions later.

* Why VCS important?

- collaboration: Multiple developers can work on the same project simultaneously without overwriting each other's changes.
- History tracking: Every change ever made to the codebase is recorded, including who made it, when and why. This allows for easy auditing and understanding the evolution of the project
- Rollbacks: If a bug is introduced or a change causes issus, you can easily revert to a previous, stable version of the code.
- Branching and Merging: Developers can work on new features or bug fixes in isolated 'branches' without affecting the main codebase. Once complete, these changes can be merged back into the main line of development
- Backup: Your entire project history is stored, often on a remote server, providing a robust backup in case of local data loss

* Types of VCS

1) Centralized version control Systems (CVCS):
- There is a single, central server that stores all the versioned files, and clients check out files from that central place.
- ex :- SVN (subversion), CVS (concurrent versions system), Perforce.
- drawbacks : A single ~~input~~ point of failure (if the central server goes down, no one can collaborate or save versioned changes)

2.) Distributed version control systems (DVCS):
- clients don't just check out the latest snapshot of the files; they fully mirror the entire repository, including its full history. This means every developers has a complete copy of the project
ex: Git, Mercurial, Bazar.
- Advantage
  - No single point of failure
  - faster operations (most operations are local)
  - Allows for ~~co~~ more complex workflows and collaboration patterns.

* Git
- leading vcs
- Git = local version control software on our machine
- GitHub = cloud / Internet-based repository

* Git and GitHub Workflow stages.

- Workspace - Area where the developer is working and creating / modifying code.
- Index / staging area :- Area that closely mimics the local repository
- Local repository :- The local area for all the code.
- Remote repository :- The remote area (GitHub) for all the code

- When user says git add file, it will get added to the staging area.
- When user says git commit file, it will get added to the local repository.
- When user says git push - the changes will go to the remote repository.

* Fundamental Git commands

1. Configuration
- Before you start using Git, you should configure your username and email address. This information will be attached to your commits.

- git config --global user.name "Your Name"
- git config --global user.email "your_email@"

2. Initializing a Repository:
- To start tracking a new project with Git, you need to initialize a Git repository in your project directory

• git init

3. cloning Repository:
- If you want to work on an existing project (like the one you created on GitHub), you will clone it to your local machine

• git clone [repo url]

4. checking status:
- This command shows you the current state of working directory and staging area. It tells you which files are modified, staged or untracked.
• git status

5. Adding files to the staging area.

• Add a specific file: git add [Filename]
• Add multiple files: git add file1.js file2.css
• Add all modified/new files in the current directory (be careful with this one)
git add .

6. committing changes

- git commit -m "your meaningful commit message"

7. Viewing commit History:
This command shows a log of all commits in the repository.
- git log
- git log --oneline (shows a more condensed, single-line view of commits)

8. Pushing changes to a Remote Repo.

- git push origin [branch-name]

9. Pulling changes from a remote repo.

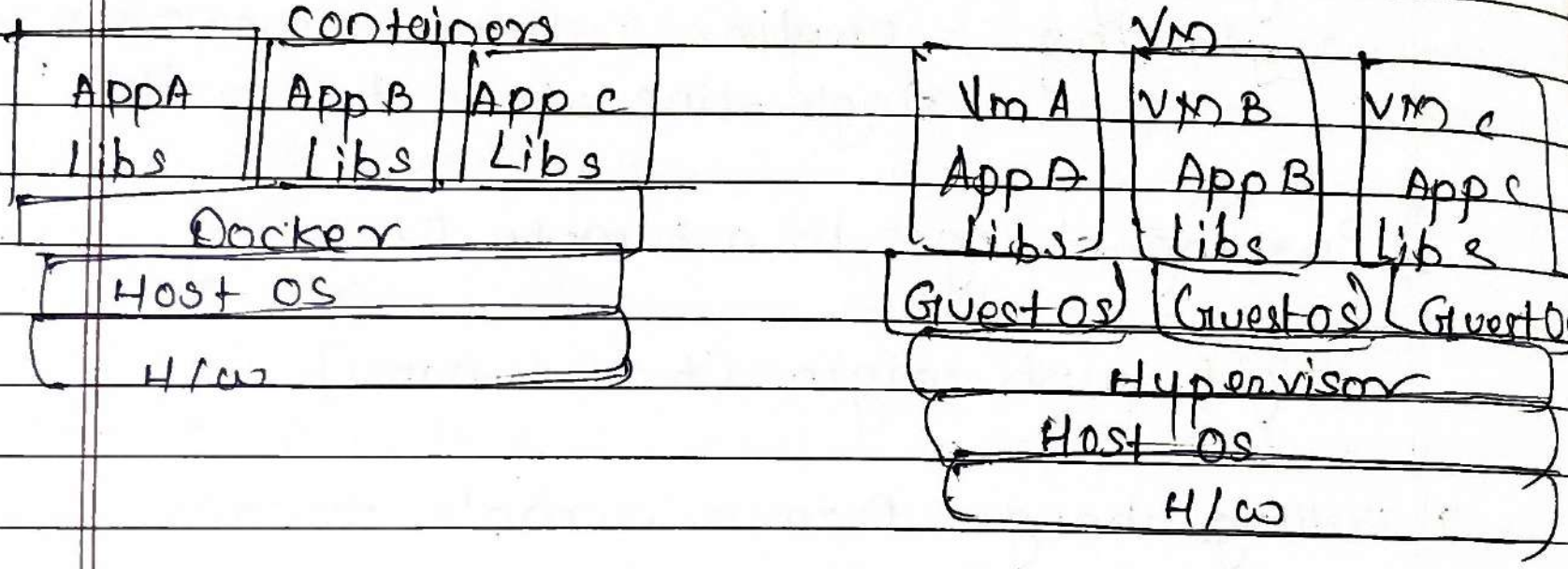
git pull origin [branch-name]

10. Branching and merging.

- view branch: git branch [new-branch name]
- switch branch: git checkout [branch-name]
  or git switch [branch-name]
- create and switch: git checkout -b [new-branch name]
- merge the other branch: git merge [source-branch]

Date / /
Page L&W

* Introduction to containers
- containers or lightweight, portable units that package an application with its dependencies and environment. They:
  - Run consistently across diff computing environments
  - Use fewer resources than traditional virtual machines.



- checking running status: - sudo systemctl status docker
- Enabling Docker: sudo systemctl enable docker
- starting Docker: sudo systemctl start docker

* Run a container
  - docker run -d it <container-name>

- docker run ubuntu - container will start and stop immediately
- docker ps - see the running containers
- docker run it ubuntu - will start in interactive terminal and ~~open~~ run till we keep it open

- docker run -dit ubuntu - container will run in the detached mode .. Even if we exit from interactive terminal

- docker exec it <container id> /bin/bash
  - to ~~execute~~ open interactive terminal.
- docker stop <three letter of container id>
  - stop the container.
- ~~docker~~ docker images :- Show list of Docker images on our local machine
- docker inspect < three letter of container id >.
- docker pull nginx - Pull the image from Docker hub onto our local machine
- docker rmi <image id> It will untag the image (mark it as unused)
- docker image prune - to remove the image complotely (like grabage collection)
- docker run -dit --name=my_ubuntu ubuntu
  :- assign name to docker containes.
- docker ps -I :- see only the latest containes
- docker rm <containes name> :- remove containes
- docker logs < ~~docker run~~ container id> : view logs

* Port Mapping / Port Forwarding
  - is the process of making a container's internal port accessible from the host machine or outside world.

* Port Publishing in Docker
- is the process of exposing a container's port to the outside world (i.e, your local machine or network) using the -p or --publish flag
- when you publish a port, Docker maps a port on your host machine to a point inside the container, allowing external access

Syntax:

docker run -p <host_port>:<container_port> <image>

* Docker Volume
- is a persistent storage mechanism used to store data outside of the container's file system. It allows you to save and share data even if the container is stopped, removed or recreated.

- creating a volume
  - docker volume create mydata
- using a volume in a container
  - docker run -v mydata:/app/data my-image
    - mydata :- volume name
    - /app/data is the path inside the container where data will be mounted.

- Inspecting volumes:
  - docker volume ls # list all volumes
  - docker volume inspect mydata

- Removing a volume.
  - docker volume rm mydata

Note: volumes are not automatically deleted when a container is removed - this protects your data

* Instruction to Build Image

- FROM :- set the base image
- CMD :- Command to be executed when the image is run as a container (or pass options to ENTRYPOINT)
- RUN :- Executes a command to help build our image
- EXPOSE :- opens new ports
- VOLUME :- sets a disk share
- COPY :- copies files from the local disk to the image
- ENV : Add an environment variable
- ENTRYPOINT :- Determines which executable runs when a container starts (use CMD to pass options to the executable)

* Buid and Execute
- docker build -t my-dai.-java
- docker run my-dai-java

- Push Docker Image on dockerhub
  - docker login

  # tag the local image

  - docker tag <image name> <username>/name:v1.0

  # push
  - docker push <username>/name:v1.0

* Stop and Remove all running containers.
  - docker stop $(docker ps -a -q)
  - docker rm $(docker ps -a -q)

* Docker Networking - Simplified Explanation
- By default, when you run Docker containers, they are connected to a default bridge nlw. In this setup, all containers can communicate with each other using their IP addresses.
- However, this default behaviour has a security risk:
- Since all containers on the bridge nlw can access one another any container can potentially interact with any other, which can lead to security issues.

- Solution:- Instead of using the default nlw, it's better to create a custom (user-defined) bridge nlw and attach containers to it. This approach offers several advantage

1. Better isolation:
- containers in a user-defined nlw are isolated from containers on the default bridge nlw and from other nlw.

2. Easier communication:
- containers in the same user-defined nlw can communicate using ~~use~~ container names as hostnames (DNS)
- This means you don't have to remember IP addresses - just use container names, which is easier and more reliable.

open -
- Busybox : Tiny source slw that packages many common UNIX utilities into one executable binary
  - docker run -it busybox

* Docker compose
- Docker compose is a tool that allows you to define and run multiple-container Docker applications using a single file.
- Instead of starting containers one by one with complex docker run commands, you can define all your services (containers) nlw and volumes in on YAML file called "docker-compose.yaml"

* comands
- docker-compose up - Builds and starts all containers

- docker-compose up -d :- Runs $ in the background (detached mode)
- docker-compose down : . stops and removes all containers/n/w
- docker-compose ps : List all running services
- docker-compose logs : shows logs from all containers

* Docker Swarm
- Docker Swarm is Docker's built-in tool for orchestrating (managing) a cluster of Docker nodes. It allows you to run and manage multiple containers across multiple machines. In a co-ordinated and fault-tolerant way.

key concepts

- Node - A machine (VM) in the swarm
- Manger Node :- Controls the swarm and makes decisions
- Worker Node :- Execute tasks assigned by Manager Node
- Service :- A task defined (e.g runs 3 replicas of a web app).
- Task :- A single containes instance assigned to a node

* Commands
- Initialize swarm :- docker swarm init

- Join a worker Node to the swarm.

docker swarm join --token <token> <manager_ip> : 2377

* kubernetes

- Need for kubenetes

problems:-

- Manual Orchestration: Imagine deploying dozens or even hundreds of containers across multiple servers. Manually starting, stopping, scaling and monitoring them is a massive, error-prone task.
- Downtime during updates: updating applications often meant taking them offline, leading to service distruptions. Achieving zero-downtime deployments was complex.
- Resource Management: Efficiently distributing containers across available server resources (CPU, memory, storage) to maximize utilization and prevent overload was difficult.
- Scaling
- Self-healing: If a container or a server failed, there was no automatic mechanism to restart the appl? or move it to a healthy host.

- kubernetes addresses these challenges by providing an open-source platform for automating deployment, scaling and management of containerized applications. It acts as an 'OS for the cloud'

abstracting away the underlying infrastructure and providing a consistent environment for your applications.

- kubernetes automatically assigns DNS names and cluster IPs so services can find and talk to each other easily.
- Performs rolling updates of applications without taking the service offline - supports ~~roo~~ rollback if needed

* kubernetes cluster
A kubernetes cluster is set of nodes (physical or vm) that work together to run your containerized applications. It consists of two main types of nodes:

1] Management Node (control Plane Node/Master Node): It is responsible for managing and controlling the entire cluster, making decisions about where to run applications, monitoring their health, and responding to changes.
key components:
a] API server: The front-end for the kubernetes control plane. All communication with the cluster (from users or other components) goes through the API server.

b) Scheduler: Watches for newly created pods that have no assigned node, and selects a node for them to run on. It considers resource requirements, h/w constraints, policy constraints, affinity and anti-affinity specifications.

c) Controller Manager: Runs various controller processes that regulate the state of the cluster. For example, a Replicaset controller ensures that the desired number of Pods are running.

2) Worker Nodes:- These are the machines where your actual application workloads (containers) run. Each worker node has the following components:

(a) ~~Kubelet: An Agent that runs on each node, maintaining n/w rules on nodes and enabling communication~~

(a) Kubelet: An agent that runs on each node in the cluster. It communicates with the control plane, ensuring that containers are running in a Pod and healthy.

(b) Kube-proxy: A n/w proxy that runs on each node, maintaining n/w rules on nodes and enabling communicatn betn Pods, both within the node and across different nodes in the cluster.

(c) Container Runtime: The software responsible for running containers (e.g. Docker containers)

* Pods:- The smallest deployable unit in kubernetes. A pod represents a single instance of a running process in your cluster. It can contain one or more containers that are tightly coupled and share resources like networking and storage. Pods are ephemeral; if a Pod dies, a new one is created to replace it.

ex.

- kubectl run my-nginx --image nginx.
- kubectl get pods

* create a deployment and a pod inside it
kubectl create deployment my-nginx --image nginx

- clean up
- kubectl delete pod my-nginx
- kubectl delete deployment my-nginx

* Deployment:- A higher-level abstraction that manages a set of identical pods. Deployments provide declarative updates for Pods and ReplicaSets. You define the desired state and the Deployment controller works to achieve and maintain that state. Deployments are ideal for stateless applications.

* ReplicaSet: A controller that ensures a specified no of Pod replicas are running at any given time. Deployments use ReplicaSets under the hood to manage Pods.

* kubectl create deployment my-apache --image httpd
- kubectl scale deployment my-apache --replicas 2
- kubectl expose deployment my-apache --type=NodePort --port=80

* ServiceTypes: Services in kubernetes are an abstract way to expose an application running on a set of Pods as a network service. They provide a stable IP address and DNS name for your application, even as Pods are created and destroyed

- ClusterIP (Default): Exposes the service on an internal IP address within the cluster. This makes the service only reachable from within the cluster Useful For internal communication between microservices.

- NodePort: Exposes the Service on a static port on each Node in the cluster. You can access the service by connecting to NodeIP:NodePort from outside the cluster. Generally not recommended for production due to port conflicts and limited load balancing.

- LoadBalancer: Exposes the Service externally using a cloud provider's load balancer. This type automatically provisions an external load balancer (if your kubernetes cluster is running on a cloud provider like AWS, GCP or Azure)

and provides an external IP address. This is the most common way to expose Internet-facing applications.

* Jenkins

* Introduction to CI/CD

- CI/CD is about automating s/w delivery. It helps teams ship code faster, more reliably, and with higher quality

• Continuous Integration (CI):
- Developers frequently merge code into a central repository. Automated builds and tests run on every commit to catch issues early.
- Reduce integration problems, get fast feedback ensure consistent builds.
-

* Continuous Delivery (CD):
- Extends CI by ensuring s/w is always in a deployable state. It automates packaging and preparing the appl^n for release
- make releases reliable and repeatable

• Continuous Deployment (CD):-
- Every change that passes all automated tests is automatically deployed to

production without human intervation
- fastest time to market, minimal human error in deployment.

* Jenkins to Build a CI/CD pipeline

- Jenkins is an open-source automation server widely used for orchestrating CI/CD pipelines. It connects your code changes to build, test, and deployment actions.

1. Source code management (SCM) Mac Integration
- Jenkins monitors your code ~~m~~ repository. When a change is detected, it triggers the pipeline.

2. Pipeline Definition
- You define your entire CI/CD workflow in a file named Jenkinsfile, stored directly in your projects repo. This makes your pipeline version controlled and reproducible.

3. Build Automation
- Jenkins ~~B~~ compiles your code, runs dependency installations and creates a deployable artifact (e.g. a TAR file, Docker Image)

4. Automated Testing
- After building, Jenkins runs various automated tests to verify functionality and quality.

5. Artifact Management
- Successful build's artifacts are stored in a central repository (e.g. Docker Registry for images)

6. Deployment Automation
- Jenkins deploys the validated artifact to various environments. This can involve direct server access, Kubernetes commands, or cloud provider APIs.

* Cloud Computing *

- Cloud computing is the on-demand delivery of IT resources and applications over the internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining your own compute servers, storage, and databases, you can access them from a cloud provider like Amazon Web service (AWS), Microsoft Azure, or Google cloud Platform (GCP).

* 5 Essential characteristics of cloud computing.
- On-Demand-Self-Service : Get computing resources (servers, storage) instantly, without needing a person to approve it.

- Broad N/w Access : Access cloud services from anywhere, on any device (phone,

laptop, etc) over the internet.

- Resource Pooling : Providers share large pools of resources (CPU, memory, storage) among many users, allocating them dynamically as needed.

- Rapid Elasticity : Quickly scale resources up or down automatically to match changing demand; resources appear unlimited.

- Measured Service : cloud usage is tracked and reported, allowing for billing based on actual consumption

* Cloud service Models:

1. Infrastructure as a service (IaaS):
- provides fundamental computing resources (virtual machines, storage, n/w) over the internet. You manage the os, applicatn and data.

exe. AWS EC2, Azure virtual machines

2. platform as a service (PaaS):
- Provides a platform allowing customers to develop, run and manage applications without the complexity of building and maintaining the underlying infrastructure. It includes the os, programming language execution environment, database and webserver

3. Software as a service (SaaS):
- Delivers complete, ready to use applications over the internet. Users simply access the software via a web browser or client application. The cloud provider manages all underlying infrastructure, platform and s/w.

* cloud deployment models:

1) Public cloud :- services offered over the internet by a third-party provider, shared among many user (e.g. AWS, Azure)

2) Private cloud :- Dedicated cloud infrastructure for a single organization, either on-premises or hosted by a third party

3) Hybrid cloud :- Combines public and private clouds, allowing data/apps to move bet'n them for flexibility.

4) Community Cloud : Shared cloud infrastructure For several organizations with common concerns (e.g. specific regulations).

5) Multi-cloud :- Using services from multiple diff public cloud providers to avoid lock-in or leverage specific strengths.

* Administering and Monitoring cloud services

- Administering : Managing cloud resources (provisioning, security/IAM, cost, compliance, automation).

- Monitoring : Continuously observing performance, availability, security and costs using tools for logs, alerts and metrics.

1] Administering cloud services
- Actively setting up and controlling environment.
- key tasks include:

- Setting up Resources : Creating and controlling VM, storage and databases.
- Controlling Access (IAM) :- Deciding who can do what by setting user roles and permissions.
- Managing cost : keeping an eye on spending, setting budgets, and making sure you're not overpaying for resources
- Ensuring security :- Setting up firewalls, manging encryption, and regularly updating systems to protect your data.
- Meeting Rules : Making sure your cloud setup follows industry regulations and company policies.
- Automating Tasks :- Using code to automatically setup and manage resources, saving time and reducing errors.

2] Monitoring cloud services
- involves constanty watching your cloud services to make sure everything is running smoothly.
- Checking Performance :- Tracking things like CPU usage and nlw speed to catch any slowdowns
- Ensuring Availability :- making sure your appl and services are always online and reachable
- Managing Logs :- collecting and analyzing records of activity to troubleshoot problems or spot security issues.
- setting up Alerts :- Getting immediate notifications for critical issues or when certain limits are reached
- Watching for security Threats : Detecting any suspicious activity or unauthorized access.

* Benefits and Limitations of cloud computing

- Benefits
  - Cost - Effectiveness
  - Scalability and Elasticity
  - Increased Agility and speed
  - Global Reach
  - High Availability and Reliability
  - Reduced IT overhead
  - Enhanced Security

- Limitations
  - Vendor lock-in: Migrating between cloud providers can be complex and costly, potentially leading to reliance on a single vendor's ecosystem
  - ~~Data security and privacy concern.~~
  - ~~Cost Management~~
  - Internet connectivity Dependency.

* cloud products and services

- compute: Virtual machines (eg. AWS EC2, Azure VMs, Google Compute Engine),
  - Serverless functions (eg. AWS Lambda, Azure Functions, Google cloud functions)
  - container orchestration (e.g. AWS Eks, Azure kubernetes service, Google kubernetes Engine
- storage: Object storage (e.g. Amazon S3, Azure Blob storage, Google Cloud storage).
  - Block storage (e.g. Amazon EBS, Azure Disk storage).
  - File storage (e.g. Amazon EFS, Azure Files).
- Databases: Relational Databases (e.g. Amazon RDS, Azure SQL Database, Google cloud SQL), ~~NoSQL~~
  - NoSQL databases (e.g. Amazon DynamoDB, Azure Cosmos DB, Google cloud Firestore)
- Networking: Virtual private clouds (VPCs), load Balancers, Content Delivery Nlw (CDNs), DNS servics
- Machine Learning and AI: Services for building, training and deploying AI models

(e.g. AWS SageMaker, Azure Machine learning, Google Vertex AI).

* Virtual Machines (VMs): These are virtualized instances of a computer, offering a flexible and scalable way to run operating systems and applications. You have full control over the OS and sw installed.

* Serverless Computing (Functions as a Service - FaaS): Allows developers to run code without provisioning or managing servers. The cloud provider automatically scales and manages the infrastructure and you only pay for the compute time consumed when your code is executed.

* Elastic cloud compute (EC2)

- Amazon Elastic compute Cloud (EC2) is a core IaaS offering from AWS. It provides resizable compute capacity in the cloud. EC2 allows you to:

- Launch virtual servers (instances) with various operating systems.
- Scale computing capacity up or down quickly to meet changing demands.
- Pay only for the capacity you actually use.
- Choose from various instance types optimized for different workloads (e.g. compute-optimized,

memory-optimized, storage-optimized, GPU instances)
- store data on diff storage options like Amazon EBS (Elastic Block store) volumes.
- Integrate with other AWS services for nlw, security and monitoring.

* Dashboard
- A cloud dashboard is a web-based graphical user interface (GUI) provided by cloud serivce providers. It serves as a centralized control panel for managing and monitoring all your cloud resources.

key functionalities.

(a) Resource Management: Launching, Stopping, resizing, and deleting virtual machines, storage, databases, and other services.

(b) Monitoring: Viewing real-time performance metrics (CPU usage, nlw I/O, disk activity), logs and alarms for your ~~responses~~ resources.

(c) Cost Management: Tracking spending, viewing biling details, and setting up budget alerts.

(d) Security and Identity: Managing user access, roles, and permissions (IAM), and configuring security groups and nlw access control lists.

(e) Networking: Setting up virtual nlw, subnets and routing rules.