Lab 9

1: Docker is a platform that **packages applications and their dependencies into lightweight containers**. These containers run consistently across various environments like local machines, staging servers, or production environments. A docker container has everything an app may need to run, like code, libraries, tools etc.

An example in game development would be if you are creating a multiplayer game, you need a dedicated server that handles match making, game states, player connections and keeps gameplay synchronised across players. Using docker in this scenario would help as you can create a backend server written in your language of choice and create a container for this using a DockerFile. This is useful as in a case where you have co-developers on different OS's, they can construct the project by just using docker. You can also create docker containers of your game that can be used as different test and production environments that you can package and compose with docker.

2: Orchestration tools like **Kubernetes** are platforms that automate the deployment, scaling, and management of containerized applications, like those made by docker. Instead of manually launching and maintaining containers, Kubernetes automates the process. It decides when and where the containers run, how many copies are there, what happens if a crash occurs and routes traffic to specific containers.

If you are running a multiplayer game, each match or game may need a dedicated server. Kubernetes aids with this.
Deployment: Deploys containers based on configurations by using pods, and ensures they are run correctly. You don't need to manually launch a game server.
Scaling: If there are more players playing your game, Kubernetes automatically notices this and scales up the number of server pods to handle the increased load. When traffic lowers, it scales down to save resources.
Management: If a game server crashes, Kubernetes can detect it and automatically replace it. If a node goes down, Kubernetes reschedules to another node. Includes load balancing to appropriately direct the traffic to the correct server. Kubernetes also allows rolling updates that gradually replaces old containers with new ones in the case of a game update.

3:
Infrastructure as a Service (IaaS):
IaaS provides virtualized computing resources over the internet (servers, storage, and networking). You manage the OS, runtime, and applications, while the provider manages the hardware.

Games dev example: You need full control over your environment to host dedicated game servers, run simulations, or build a custom backend.

Platform as a Service (PaaS)
PaaS provides a platform and environment to build, run, and manage applications without worrying about infrastructure(OS management or patching).

Games dev example: Great for backend services, leaderboards, or APIs that require scalability and ease of deployment. Example : Running your game analytics or real-time chat server.

Software as a Service (SaaS)
SaaS delivers fully functional applications over the web. You just use the software—everything else is managed by the provider.

Games dev example: You use SaaS tools to support your dev process: collaboration, asset creation, monitoring, or customer support. Example : using apps like Discord for player communities or Trello for team task tracking.

4:

Hole punching is a technique used to establish a direct connection between two peers behind NATs, even when those NATs are port-restricted. It works for UDP and sometimes TCP.

Example : **Client A** and **Client B** want to connect directly, but both are behind NATs.

Use a publicly accessible server to help them punch holes.

Both clients connect to the server via UDP.

Each NAT opens a port for outbound communication.

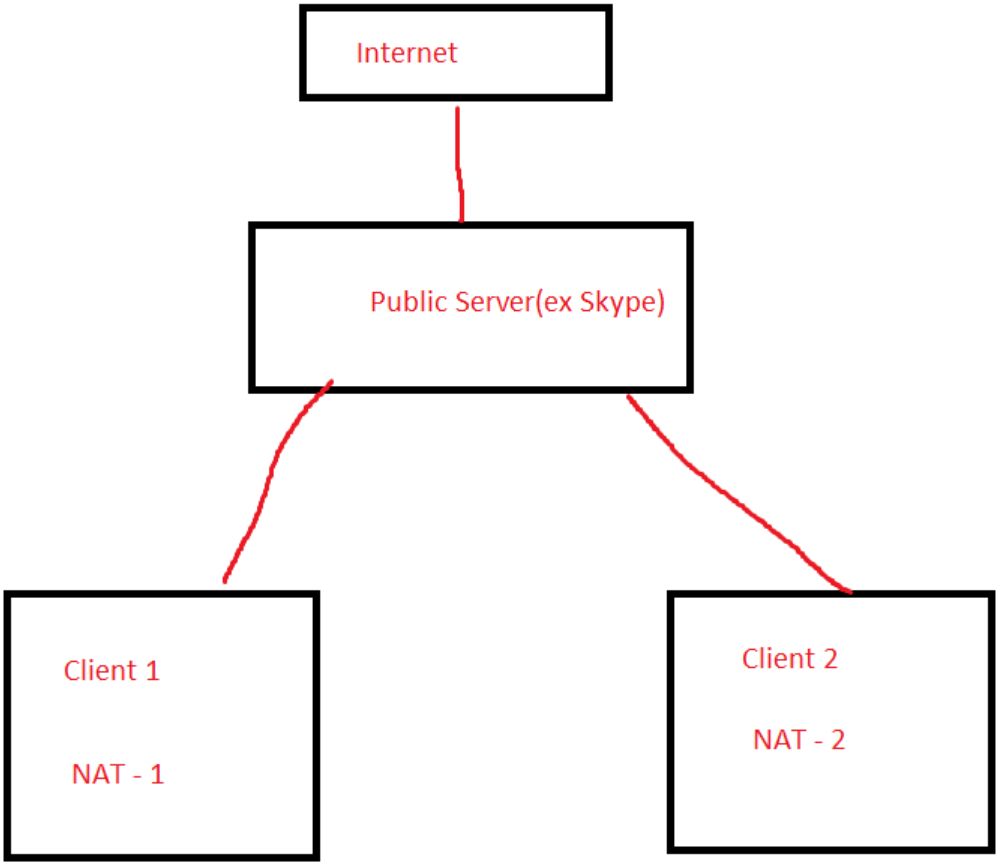
Server sees:

A's public IP:Port from NAT-A

B's public IP:Port from NAT-B

Server shares their public addresses with each other. Clients simultaneously send UDP packets to each other's public IP:Port. This punches a hole in each NAT device.

Even though the first packets may get dropped, the NATs now expect traffic from those specific external addresses. Eventually, a UDP packet gets through, and a P2P connection is established.


Internet
Public Server(ex Skype)
Client 1
NAT - 1
Client 2
NAT - 2