(a)

IaaS gives users virtualized computing resources such as servers, storage, and networks. Developers don't need to buy or maintain physical hardware. For example, when building a new web application, a team can use **Amazon EC2** or **Microsoft Azure Virtual Machines** to create and scale virtual servers on demand. They have full control over the operating system and runtime, which makes it easier to configure environments for testing or production.

PaaS provides a ready-to-use platform that includes operating systems, databases, and development tools. It removes the need to manage the underlying infrastructure so developers can focus on writing code. For instance, a group creating a mobile app might use **Google App Engine** or **Heroku**, which handle server management, automatic scaling, and updates while the team just deploys the code.

SaaS delivers complete software applications over the internet, usually on a subscription basis. Instead of installing programs locally, users simply access the service through a browser. In a development setting, a team might collaborate using **GitHub** or **Atlassian Jira** as SaaS tools. These services handle all maintenance and updates while giving developers features like issue tracking and version control.

(b)

Docker is a popular open-source platform that packages applications and all their dependencies into lightweight units called **containers**. A container runs the same way on any machine that has the Docker engine installed, which solves the "it works on my computer" problem.

A typical scenario is a development team building a microservices-based web application. Each microservice can be placed in its own Docker container with the exact runtime, libraries, and configuration it needs. Developers can run these containers locally for testing, and later deploy the same containers to a cloud environment such as AWS or Azure.

Using containers helps the development and deployment process in several ways:

Consistency: The environment inside the container is identical across developers' laptops, testing servers, and production servers.

Isolation: Each service runs in its own container, so updates or crashes in one service don't break others.

Scalability: Containers start quickly and can be replicated easily, making it simpler to scale the application when demand grows.

(c)

docker run -it --rm -p 5678:5678 -v "$env:USERPROFILE\.n8n:/home/node/.n8n" n8nio/n8n

This Docker command starts an n8n container and sets up everything needed to run it locally. The `docker run` part tells Docker to create and start a new container, while the `-it` flag keeps the terminal interactive so you can see logs in real time. The `--rm` option makes Docker automatically remove the container when it stops so you don't leave unused containers behind. The `-p 5678:5678` mapping exposes port 5678 inside the container to port 5678 on your Windows host, which is why you can open n8n in a browser at `http://127.0.0.1:5678`. The `-v "$env:USERPROFILE\.n8n:/home/node/.n8n"` option mounts a folder named `.n8n` from your Windows user directory into the container, allowing your workflows and settings to persist even if the container is deleted. Finally, `n8nio/n8n` is the official image from Docker Hub that contains the n8n application itself, so Docker downloads it if it's not already on your machine and runs it with the settings you provided.