# **In‑Class Assessment 1 — Answers**

## **(a) Cloud service models: IaaS, PaaS, SaaS**

****Infrastructure as a Service (IaaS)****

* ****What it is:**** Raw compute, storage, and networking delivered as on‑demand, metered infrastructure. You manage OS and above; provider manages physical hardware and virtualization.
* ****Dev use case:**** Stand up a custom CI build fleet or GPU runners using Amazon EC2 / Google Compute Engine / Azure VMs. You choose the base image, install compilers, SDKs, and agents.
* ****Examples:****
  1. Compute: AWS EC2, Azure Virtual Machines, Google Compute Engine
  2. Storage/Network: AWS S3/EBS/VPC, Azure Blob/Disks/VNet, GCP Cloud Storage/Persistent Disk/VPC
* ****Why devs pick it:**** Maximum control over OS, runtime, and network topology; lift‑and‑shift of legacy apps; custom AMIs; self‑managed Kubernetes on raw VMs.

****Platform as a Service (PaaS)****

* ****What it is:**** A managed runtime/platform that abstracts servers and OS. You push code/artifacts; the platform handles buildpacks, scaling, health checks, logs.
* ****Dev use case:**** Deploy a Node/Java/Python web API by pushing to a PaaS (e.g., Heroku, Render, Azure App Service, Google App Engine). The platform builds from package.json/pom.xml, provisions HTTPS, autoscaling, and rollbacks.
* ****Examples:**** Heroku, Google App Engine, Azure App Service, AWS Elastic Beanstalk, Fly.io, Cloud Run (serverless containers).
* ****Why devs pick it:**** Faster delivery, low ops burden, opinionated best practices, built‑in CI/CD hooks and observability.

****Software as a Service (SaaS)****

* ****What it is:**** Fully managed applications delivered over the web. Vendor manages everything; you configure and consume.
* ****Dev use case:**** Use GitHub/GitLab for source control and CI; Jira/Linear for planning; Datadog/New Relic for monitoring; Auth0/Okta for auth instead of building your own.
* ****Examples:**** GitHub, Atlassian Cloud (Jira/Confluence), Figma, Notion, Salesforce, Auth0.
* ****Why devs pick it:**** Zero infrastructure to maintain; pay as you go; enterprise features (RBAC, SSO, audit) out of the box.

****Rule of thumb:**** IaaS = maximum control; PaaS = balanced control/speed; SaaS = maximum speed, minimum control.

## **(b) What is Docker? When to use containerization?**

****Docker**** is a container platform that packages an application and all its dependencies into a single, portable image. When you run an image, Docker creates an isolated container with its own filesystem, process namespace, and network, sharing the host kernel (lighter than a VM).

****Scenario:**** Microservice with API + worker + scheduler

* You containerize each component (e.g., api, worker, scheduler) and define them in docker-compose.yml with a shared network and dependencies (e.g., Postgres, Redis).
* ****How it helps:****
  1. ****Reproducible builds**** — same image runs in dev, CI, and prod.
  2. ****Environment parity**** — no “works on my machine”; dependencies are in the image.
  3. ****Fast spin‑up**** — docker compose up starts the full stack locally.
  4. ****Scalability**** — scale services with --scale or in Kubernetes.
  5. ****Isolation**** — conflicting library versions do not collide across projects.

## **(c) Deploy n8n with Docker + screenshot**

### **Option 1 — Docker Compose (recommended)**

1. Copy .env.example to .env (optional).
2. Run:
   * ****macOS/Linux:**** bash run.sh up
   * ****Windows PowerShell:**** ./run.ps1 up
3. Open **[http://127.0.0.1:5678](http://127.0.0.1:5678/)**.
4. Take a screenshot and save it as screenshot.png in the repo root.
5. Stop: bash run.sh down or ./run.ps1 down.

**docker-compose.yml**explained (line by line):****

* services: — defines runnable containers in the app.
* n8n: — our n8n service using the official image.
* image: n8nio/n8n:latest — pulls the latest published n8n image.
* ports: - "5678:5678" — maps host port 5678 → container 5678, so you can visit [http://127.0.0.1:5678](http://127.0.0.1:5678/).
* env\_file: - .env — loads environment variables from .env.
* environment: — inline env overrides; N8N\_HOST/N8N\_PORT configure URLs; N8N\_SECURE\_COOKIE=false keeps local login simple.
* volumes: - n8n\_data:/home/node/.n8n — persists workflows and credentials across restarts.
* restart: unless-stopped — auto‑restart if the container exits/crashes.
* volumes: n8n\_data: — named volume definition.

### **Option 2 — One‑liner docker run**

docker run -it --name n8n -p 5678:5678 -v n8n\_data:/home/node/.n8n -e N8N\_HOST=127.0.0.1 -e N8N\_PORT=5678 -e N8N\_SECURE\_COOKIE=false --restart unless-stopped n8nio/n8n:latest

****Flag breakdown:****

* docker run — create and start a container.
* -it — interactive TTY (useful to see logs/stop with Ctrl+C).
* --name n8n — human‑readable container name.
* -p 5678:5678 — publish host:container port.
* -v n8n\_data:/home/node/.n8n — named volume for persistent data.
* -e KEY=VALUE — set environment variables inside the container.
* --restart unless-stopped — restart policy on daemon start/crash.
* n8nio/n8n:latest — image to run.

****Troubleshooting tips:**** If port 5678 is busy, change the left side (e.g., 8080:5678) and browse [http://127.0.0.1:8080](http://127.0.0.1:8080/). If you update the image, run docker pull n8nio/n8n:latest before starting.

