In-Class Assessment 1 202283910012 张景橦

(a) Describe the three primary cloud service models in cloud computing infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Provide specific examples of how each model can be applied in the context of software development.

Infrastructure as a Service (IaaS)

IaaS provides virtualized computing infrastructure over the internet, including servers, storage, networking, and virtual machines. Users have full control over the operating systems, middleware, and applications running on the infrastructure while the cloud provider manages the physical hardware.

Example in software development: A development team uses AWS EC2 instances to set up a temporary testing environment. They configure the instances with specific CPU, memory, and storage to match production specifications, install a Linux OS, and deploy a Node.js runtime along with test databases. This allows them to simulate production conditions for integration testing without investing in physical servers.

Platform as a Service (PaaS)

PaaS offers a complete development and deployment platform, including operating systems, databases, development tools, and hosting environments. It abstracts away underlying infrastructure management, enabling developers to focus solely on writing and deploying code.

Example in software development: A team building a Python web application uses Heroku. They push their Django code to Heroku, which automatically handles server configuration, dependency installation (via requirements.txt), and load balancing. The platform also provides built-in CI/CD tools, allowing the team to automate testing and deployment workflows without managing servers.

Software as a Service (SaaS)

SaaS delivers ready-to-use software applications over the internet on a subscription basis. Users access the software via web browsers or APIs, with no need for local installation or maintenance.

Example in software development: A development team uses GitHub (a SaaS tool) for version control. They host their code repositories, collaborate on pull requests, and track changes through GitHub’s web interface. Additionally, they use Jira (another SaaS tool) to manage project tasks, track bugs, and coordinate sprints—both tools operate entirely in the cloud, requiring no local setup.

(b) What is Docker? Describe a scenario where you would use containerization technologies such as Docker in software development. How does containerization contribute to the development and deployment process of software in this scenario?

Docker is an open-source containerization platform that packages an application and all its dependencies (such as libraries, runtime environments, and configuration files) into a standardized unit called a "container." Containers are lightweight, portable, and self-sufficient, ensuring that applications run consistently across different environments (e.g., development machines, testing servers, and production systems) regardless of the underlying infrastructure.

Scenario for Using Docker in Software Development

Imagine a cross-functional development team working on a microservices-based e-commerce application. The application consists of multiple components: a Node.js-based API gateway, a Python-based product catalog service, a PostgreSQL database, and a Redis cache. Team members use diverse operating systems (Windows 10, macOS Ventura, and Ubuntu 22.04), and the application requires specific versions of dependencies (e.g., Node.js v18, Python 3.9, PostgreSQL 14). Historically, the team faced frequent issues like "it works on my machine but fails in testing" due to environment mismatches, and setting up local development environments for new members took 2–3 days.

How Containerization Contributes

Environment Consistency: By defining each service (API gateway, product catalog, database) in a Dockerfile and orchestrating them with docker-compose.yml, the team ensures that every member uses identical environments. For example, the Node.js service’s Dockerfile specifies FROM node:18-alpine and installs exact npm package versions, eliminating "version mismatch" errors.

Rapid Onboarding: New team members can simply run docker-compose up to spin up all services (API, database, cache) in minutes, instead of manually installing and configuring each dependency.

Isolation: Each service runs in its own container, so updates to the Python service (e.g., upgrading a library) won’t affect the Node.js service or the host machine, preventing "pollution" of local environments.

Streamlined Deployment: The same containers used in development can be pushed to a container registry (e.g., Docker Hub) and deployed to staging/production environments. This avoids "deployment drift"—issues arising from manual configuration differences between development and production.

Scalability Testing: The team can easily replicate production-like conditions using Docker commands, facilitating load testing early in development.

(c) Deploy n8n (n8n.io) with Docker and capture a screenshot of http://127.0.0.1:5678. Please explain the docker command in detail.

图形用户界面, 应用程序

AI 生成的内容可能不正确。

1. Environment Check Command

# Check Docker version

docker --version

# Check Docker service status

docker info

# List existing images

docker images

# List running containers

docker ps

1. Image Pull Command

# Test network connection (pull test image)

docker pull hello-world

# Pull official n8n image

docker pull n8nio/n8n:latest

1. Container Deployment Command

docker run -d --name n8n -p 5678:5678 -v ~/.n8n:/home/node/.n8n \

-e N8N\_HOST=127.0.0.1 \

-e N8N\_PORT=5678 \

-e N8N\_PROTOCOL=http \

-e N8N\_SECURE\_COOKIE=false \

n8nio/n8n:latest