Purwadhika
Digital Technology School

Module 06

Cloud Platforms and Application Deployment

Job Connector Program

Outline

Cloud Platforms Overview
Introduction to server and cloud environments used to run modern web applications in production.

VPS vs Cloud Computing (GCP)
Understanding the differences, trade-offs, and use cases between VPS and cloud platforms like Google Cloud.

Deployment Architecture
How frontend and backend applications communicate through servers, reverse proxies, and networks.

Production Deployment Tools
Key tools used in real deployments, including Linux servers, Docker, Nginx, and HTTPS configuration.

Introduction

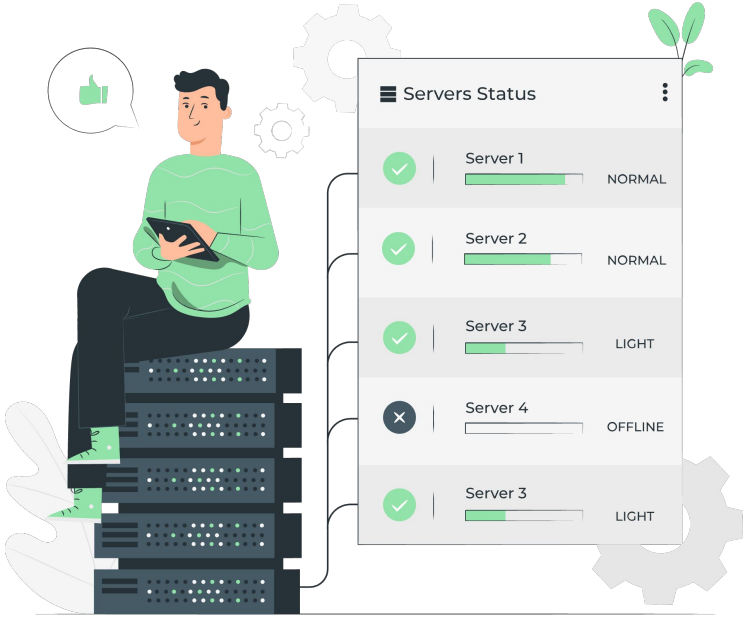
Application deployment is a critical phase in modern software development. Developers are expected not only to build applications, but also to understand **where and how applications run in production.** This topic introduces various **cloud platforms, server models, and deployment approaches**, enabling students to choose the right infrastructure for their projects.

Cloud Platform Overview

Cloud platforms provide on-demand computing resources such as servers, storage, networking, and managed services. Instead of managing physical servers, developers can provision infrastructure programmatically and scale as needed.

Key benefits of cloud platforms:

- Scalability and elasticity
- High availability and reliability
- Global infrastructure
- Pay-as-you-go pricing model



Types of Hosting and Deployment Platforms

Traditional Web Hosting

- **Shared Hosting:** Websites are hosted on servers shared with other websites. It's cost-effective but may have limitations on performance and customization.
- **Virtual Private Server (VPS):** Provides more control and resources than shared hosting. Users get a virtualized instance on a server.

Cloud Hosting

Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP): Leading cloud providers that offer scalable and flexible hosting solutions. Developers can choose specific services like EC2 instances, S3 storage, or App Engine to deploy applications.

Deployment Platforms

Platform as a Service (PaaS)

Heroku, Google App Engine, Microsoft Azure App Service: PaaS platforms abstract away infrastructure management, allowing developers to focus on code. They often provide built-in tools for deployment, scaling, and monitoring.

Containerization

Docker: Containers package applications and their dependencies, making it easy to deploy consistently across different environments. Containers can run on various platforms, such as Kubernetes.

Serverless Computing

AWS Lambda, Azure Functions, Google Cloud Functions: Serverless platforms allow developers to run code without managing servers. Code is executed in response to events (e.g., HTTP requests, database changes) on a pay-as-you-go basis.

Different Deployment Strategies

Rolling Deployment

- In a rolling deployment, the new version is gradually rolled out to a subset of servers or instances while the old version continues to handle the remaining traffic.
- Advantages: This strategy minimizes downtime and allows for easy rollback if issues are detected during the deployment.
- Use Cases: Well-suited for applications that can handle mixed-version traffic, ensuring a smooth transition between versions.

Blue-Green Deployment

- Blue-Green deployment involves maintaining two identical production environments, labeled "blue" and "green." Traffic is routed to one environment while the other is updated, and the switch is made once the update is successful.
- Advantages: Offers near-zero downtime, as the switch between environments is instantaneous. It provides a reliable way to roll back in case of issues.
- Use Cases: Suitable for applications where downtime is critical or for frequent and iterative releases.

Different Deployment Strategies

Canary Deployment

- Canary releases involve deploying a new version to a small subset of users before gradually rolling it out to the entire user base. This allows for monitoring and testing the new version in a real-world environment.
- Advantages: Enables early detection of issues and provides a controlled way to test new features with a limited audience.
- Use Cases: Useful for minimizing the impact of potential issues by exposing them to a small group of users first.

Feature Toggles (Feature Flags)

- Feature toggles involve using conditional statements in the code to enable or disable specific features. This allows for deploying new features that can be activated or deactivated dynamically.
- Advantages: Provides flexibility to enable or disable features without redeploying the application. Useful for A/B testing and progressive feature rollouts.
- Use Cases: Effective for releasing partially implemented features or conducting controlled experiments.

Subscribe VPS / Cloud Computer

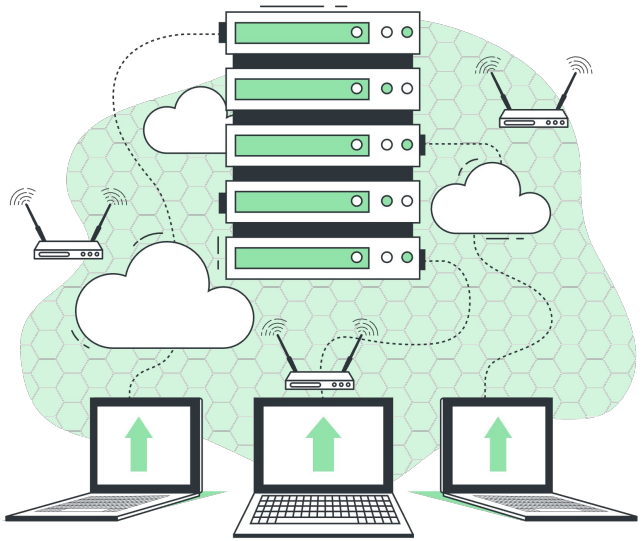
VPS or cloud computers can be purchased from local or global providers. Choose a provider based on budget, region, and learning goals.

Local Providers (Indonesia)

- DewaVPS
- BizNet Gio
- Hostinger
- Jagoanhosting

Global Providers

- Google Cloud Platform (Compute Engine)
- AWS (EC2)
- Microsoft Azure (Virtual Machines)
- DigitalOcean
- Linode



Recommended Server Specifications

Use Case	CPU	RAM	Storage	Operating System	Notes
Learning / Basic Deployment	1 vCPU	1–2 GB	20–40 GB SSD	Ubuntu Server (Latest LTS)	Suitable for learning React TS & Express TS deployment
Small Production / Low Traffic	2 vCPU	2–4 GB	40–80 GB SSD	Ubuntu Server (Latest LTS)	Good for MVPs and small applications
Medium Production	4 vCPU	4–8 GB	80–160 GB SSD	Ubuntu Server (Latest LTS)	Handles active users and moderate traffic
High Traffic / Scaling Ready	8+ vCPU	16+ GB	160+ GB SSD	Ubuntu Server (Latest LTS)	Designed for high-traffic and scalable systems

Key Notes:

- SSD storage is recommended for performance
- Start with small resources and scale as needed
- Suitable for deployment using Nginx, Docker, PM2, and HTTPS

Initial Server Preparation & Firewall Setup

Access VPS via Provider Dashboard
After purchasing a VPS or cloud server, access the server terminal from the provider dashboard (web console).

Update System & Install Essential Tools
Run the following commands to prepare the Ubuntu server:

sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential libssl-dev curl wget nano htop ufw

Purpose:

- Prepare the Ubuntu system for development
- Install essential tools required for server setup

Configure Firewall (UFW) for SSH Access
UFW is used to control incoming network access on the server.
⚠️ Important: Always allow SSH before enabling the firewall.

sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable

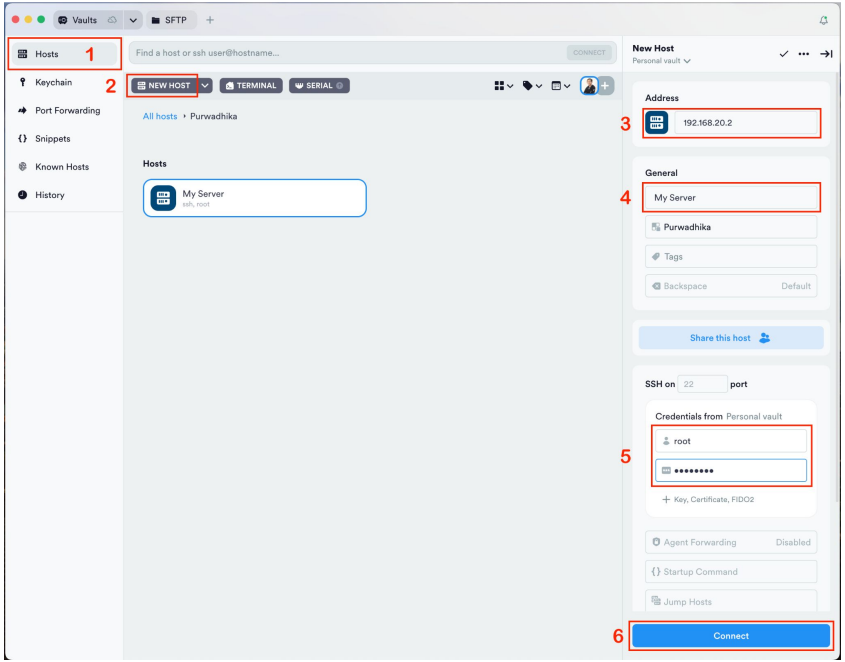
Verify firewall rules:
sudo ufw status

After UFW is configured, the server is ready to be accessed using: ***Termius or OpenSSH (Terminal)***

Access VPS with Termius

To use a VPS, we need to access it via an SSH connection. One of the applications you can use is Termius. Visit the following link : https://termius.com/download/windows

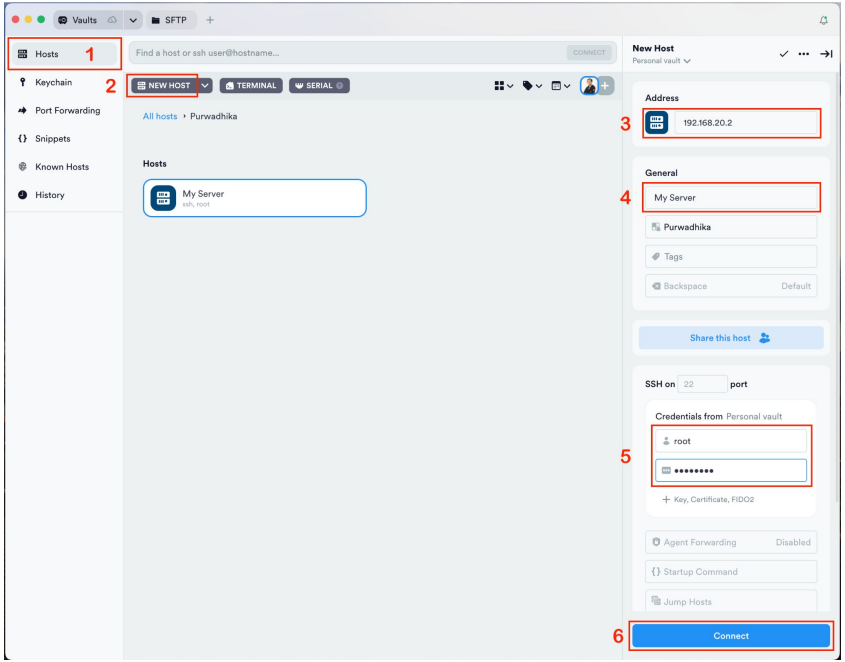
After download and install, you can config vps connection.



Access VPS with Termius

To use a VPS, we need to access it via an SSH connection. One of the applications you can use is Termius. Visit the following link : https://termius.com/download/windows

After download and install, you can config vps connection.



VPS & Cloud Server Deployment (Hands-on Focus)

Timezone & Git Configuration

Set timezone:

timedatectl list-timezones

sudo timedatectl set-timezone Asia/Jakarta

timedatectl

Install & configure Git:

sudo apt install -y git

git config --global user.name "Your Name"

git config --global user.email "youremail@domain.com"

VPS & Cloud Server Deployment (Hands-on Focus)

Node.js Installation (NVM-Based)
Install NVM:
```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
# or
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
```

Reload shell:
```
source ~/.bashrc
```

Check available Node versions:
```
nvm ls-remote
```

Install and use version:
```
nvm install xx.xx.xx
nvm use xx.xx.xx
```

Optional symlink (for system-wide usage):
```
sudo ln -s "$NVM_DIR/versions/node/$(nvm version)/bin/node" /usr/local/bin/node
sudo ln -s "$NVM_DIR/versions/node/$(nvm version)/bin/npm" /usr/local/bin/npm
```

Nginx & Reverse Proxy

Nginx is a high-performance web server commonly used to:

- Serve static files
- Act as a reverse proxy
- Handle incoming HTTP/HTTPS traffic

Why Use Nginx

- Lightweight and efficient
- Handles high traffic with low resource usage
- Commonly used in production environments
- Works well with Node.js, Docker, and SSL (Certbot)

Alternatives to Nginx

- **Apache** – traditional web server, easier for beginners
- **Caddy** – simple configuration with automatic HTTPS
- **Traefik** – dynamic reverse proxy, often used with Docker

Nginx is used because it is widely adopted in industry and production-ready.

Reverse Proxy Concept

What is a Reverse Proxy

A reverse proxy sits in front of backend applications and forwards client requests to the correct service.

Purpose:

- Expose applications via standard ports (80 / 443)
- Hide internal application ports
- Enable multiple applications on one server

Install Nginx:
sudo apt install -y nginx
sudo systemctl enable nginx
sudo systemctl start nginx

Create Reverse Proxy Configuration

Create a configuration file inside /etc/nginx/conf.d/
sudo nano /etc/nginx/conf.d/app.conf

Notes:

- The filename can be any name (e.g. app.conf, api.conf, subdomain.conf)
- Use different config files for different projects on the same server

Reverse Proxy Concept

Example Configuration

```
server {
    server_name subdomain.domain.com;

    location / {
        proxy_pass http://127.0.0.1:{PROJECT_PORT};
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Important Notes:

- **{PROJECT_PORT}** must match the port used by your application
- Each project must run on a different port
- One server can host multiple projects using different subdomains

Test & Reload Nginx

sudo nginx -t

sudo systemctl restart nginx

Key Takeaways

- *Nginx routes traffic from the internet to internal applications*
- *Reverse proxy enables multiple apps on one server*
- *Configuration files can be separated per project*
- *Project ports must be unique*

Process Manager (PM2)

PM2 is a process manager for Node.js applications that keeps applications running in the background and automatically restarts them if they crash.

Why Use PM2:

- Keeps Node.js applications running continuously
- Automatically restarts apps on failure
- Provides monitoring and log management
- Commonly used in production environments

Project Location on Server

Before running PM2, make sure your project is placed in a proper directory. Common practice:

/var/www/project-name

Notes:

- Each project should have its own directory
- Nginx routes traffic to the project port, not the directory

Running Application with PM2

Important: Always run PM2 inside the project directory.
Install PM2 Globally
sudo npm install -g pm2

Start Application
pm2 start dist/index.js --name api

Notes:

- ***dist/index.js*** is the compiled production entry file
- The app must already be running on a specific port
- That port must match the Nginx reverse proxy configuration

Common PM2 Commands
pm2 restart api
pm2 stop api
pm2 monit

Enable PM2 on Server Startup
pm2 save
pm2 startup

Purpose:
Automatically start applications after server reboot

Key Reminders

- *PM2 manages the application process, not HTTP routing*
- *Nginx handles incoming traffic and forwards it to the PM2-managed app*
- *Each application must use a unique port*
- *Always execute PM2 commands from within the project directory*

Domain & DNS Basics

A domain is a human-readable address (e.g. example.com) that points users to a server's IP address.

Purpose:

- Makes applications accessible without using IP addresses
- Required for HTTPS and SSL certificates
- Used to separate frontend, backend, or multiple projects

Domain & Subdomain Example

- ***domain.com*** → main website
- ***api.domain.com*** → backend service
- ***app.domain.com*** → frontend application

How Domains Work (DNS)

- Domain names are mapped to server IPs using DNS records
- Common DNS record:
  - ***A Record → points domain to server IP address***

HTTPS & SSL with Certbot

Certbot is a tool used to automatically generate and manage SSL/TLS certificates from Let's Encrypt.

Purpose:

- Enable HTTPS for web applications
- Encrypt data between client and server
- Improve security and trust
- Required for production-ready applications

Why Use Certbot

- Free and widely trusted (Let's Encrypt)
- Automatic Nginx configuration
- Automatic certificate renewal
- Industry-standard HTTPS solution

Prerequisites

- Nginx is already installed and running
- Domain or subdomain is pointing to the server IP
- Nginx reverse proxy configuration is active

Install Certbot

sudo apt install -y certbot python3-certbot-nginx

Generate SSL Certificate

sudo certbot --nginx -d subdomain.domain.com

What happens:

- *Certbot verifies domain ownership*
- *SSL certificate is generated*
- *Nginx configuration is updated automatically*
- *HTTP traffic is redirected to HTTPS*

Key Notes

- *Run Certbot for each domain or subdomain*
- *Certbot modifies the existing Nginx config*
- *SSL certificates are renewed automatically*

Docker Installation on Ubuntu VPS

Install Docker on Ubuntu VPS

- Login to VPS via SSH (dashboard terminal / SSH client)
- Update package lists
  sudo apt update
- Install required packages
  sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
- Add Docker official GPG key
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | ***sudo apt-key add -***
- Add Docker repository
  sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
- Update APT & install Docker
  sudo apt update
  sudo apt install docker-ce -y
- Verify Docker service
  sudo systemctl status docker
- Expected output: Docker service is active (running).

Thank you