애플리케이션 배포 및 운영 · 심화 과정

애플리케이션 배포 개요 Deployment Overview

내가 만든 프로그램을 어떻게 세상에 내보내고, 살아 있게 유지하는가

리눅스 기초(명령어 · 프로세스 · 리다이렉션 · 파이프라인 · 권한 · 쉘 스크립트)를 마친 학습자를 위한 **배포 · 운영 모듈 도입 강의자료**입니다. 네트워크 기초부터 배포 전 과정, 그리고 운영의 첫걸음까지 한 번에 정리합니다.

이번 모듈에서 우리가 갈 길

개념 이해 → 네트워크 기초 → 빌드/컴파일 → 어디에 올릴까 → 배포 전체 흐름 → 운영(살아 있게 유지) → 자동화·컨테이너 예고

1 배포(Deployment)란 무엇인가

배포란, 내가 만든 애플리케이션(웹 페이지·서버·프로그램)을 다른 사람들이 네트워크(주로 인터넷)를 통해 사용할 수 있도록, 항상 켜져 있는 서버에 올려 서비스 상태로 만드는 일을 말합니다.

아무리 잘 만든 웹 페이지나 서버라도 내 컴퓨터 안에만 있으면 다른 사람은 쓸 수 없습니다. 인터넷 상에서 누구나 찾아올 수 있는 **고유한 주소(IP·도메인)**를 부여받고, 그 주소로 접속하면 동작하도록 만드는 것 — 이것이 배포입니다.

> 🔍 **비유로 이해하기**
>
> 개발은 **집에서 요리를 연습**하는 단계입니다. 맛있게 만들어도 우리 집 식구만 먹을 수 있죠. 배포는 **가게를 열어 영업**하는 것입니다. 간판(도메인)을 달고, 주소(IP)를 공개하고, 영업시간 내내(서버 상시 가동) 손님(사용자)을 받는 상태가 되어야 비로소 "서비스"가 됩니다.

개발 환경 vs 운영 환경

프로젝트 초기에는 내 PC에서 `localhost`(127.0.0.1) 주소로 테스트하며 개발합니다. 하지만 **localhost는 다른 컴퓨터에서는 접근할 수 없는 "내 컴퓨터 자신"을 가리키는 주소**입니다. 실제 서비스를 하려면 외부에서 접근 가능한 환경, 즉 **운영 환경(Production)**에 올려야 합니다.

구분	개발 환경 (Development)	운영 환경 (Production)
접속 주소	`localhost` / 127.0.0.1	공인 IP · 도메인
사용자	나(개발자)	실제 사용자 전체
데이터	테스트용 가짜 데이터	실제 데이터
중요 가치	빠른 수정·실험	안정성 · 보안 · 성능

실무에서는 보통 **개발(dev) → 스테이징(staging, 운영과 똑같이 맞춘 최종 점검용) → 운영(production)**의 3단계로 환경을 나누어, 사용자에게 영향을 주기 전에 충분히 검증합니다.

localhost와 127.0.0.1 (루프백)

`localhost`는 `www.naver.com` 같은 "도메인 이름"이고, 그 이름과 짝지어진 IP 주소가 바로 **127.0.0.1**입니다. 이 주소를 **루프백(loopback) 주소**라고 부릅니다. 패킷이 외부 네트워크로 나가지 않고 **곧장 나 자신에게 되돌아오기** 때문입니다.

🔗 이미 배운 것과 연결

리눅스에서 `/etc/hosts` 파일을 열어 보면 `127.0.0.1 localhost` 한 줄이 있습니다. 바로 이 파일이 "localhost라는 이름 → 127.0.0.1" 매핑을 정의합니다. 권한·파일 다루기에서 배운 내용이 여기서 쓰입니다.

2 네트워크 기초 — 주소·이름·문(門)

배포를 이해하려면 "주소(IP), 이름(도메인), 문(포트)" 세 가지를 먼저 잡아야 합니다.

2-1. IP 주소 — 인터넷 상의 물리적 주소

IP 주소는 네트워크에 연결된 모든 장치에 할당되는 **고유 식별 번호**입니다. 컴퓨터들이 서로를 알아보고 데이터를 주고받기 위해 사용하는 주소죠. 가장 널리 쓰이는 IPv4는 `192.168.0.10` 처럼 점으로 구분된 네 개의 숫자로 표현합니다.

> 📦 **비유로 이해하기**
>
> 택배를 보낼 때 "어디로 보낼지" 주소가 있어야 물건이 수취인에게 도착합니다. 인터넷도 똑같습니다. **IP 주소 = 건물의 물리적 주소, 도메인 이름 = 그 건물의 이름(아파트 단지명)**처럼 작동합니다.

공인 IP vs 사설 IP — 배포에서 특히 중요한 구분입니다.

구분	사설 IP (Private)	공인 IP (Public)
범위 예	`10.x` , `172.16~31.x` , `192.168.x`	그 외, ISP·클라우드가 부여
식별 범위	집·회사 내부 네트워크 안에서만	인터넷 전체에서 직접 식별
용도	공유기에 연결된 내부 기기	외부 공개 서버(배포 대상)

`ipconfig` (윈도우) / `ip addr` (리눅스)에서 보이는 `192.168.x.x` 는 공유기가 내부 기기에 나눠준 **사설 IP**입니다. 여러 내부 기기가 하나의 공인 IP로 인터넷에 나가는 방식을 **NAT(Network Address Translation)**라고 합니다. **외부에 서비스를 공개하려면 공인 IP가 필요**합니다.

2-2. 포트(Port) — 한 컴퓨터 안의 여러 문

한 대의 컴퓨터(=하나의 IP)에서는 웹 서버, 데이터베이스 등 **여러 프로그램이 동시에 실행**됩니다. 들어온 요청을 "어느 프로그램에게 줄지" 구분하는 번호가 **포트**입니다(0~65535).

> 🚢 **비유로 이해하기**
>
> 포트(port)는 본래 항구·공항을 뜻합니다. IP가 **건물 주소**라면 포트는 그 건물의 **호실 번호**입니다. "주소(IP) + 호실(포트)"가 합쳐져야 정확히 어느 프로그램에 닿을지 정해집니다. 이 조합을 **소켓(socket)**이라 부릅니다.

자주 쓰는 **well-known 포트**는 외워 두면 실습이 편합니다.

포트	서비스	포트	서비스
`22`	SSH (원격 접속)	`3306`	MySQL / MariaDB
`80`	HTTP (웹)	`5432`	PostgreSQL
`443`	HTTPS (암호화 웹)	`6379`	Redis
`8080`	HTTP 대체(개발·WAS)	`3000`	개발 서버(Node 등) 관례

2-3. 도메인과 DNS — 이름을 주소로 바꾸기

`www.naver.com` 같은 문자열이 **도메인 이름**입니다. 사람은 이름을 기억하기 쉽지만, 컴퓨터는 결국 IP 주소로 통신합니다. 그래서 도메인을 IP로 바꿔 주는 서비스가 필요한데, 이것이 **DNS(Domain Name System)**입니다 — 인터넷의 전화번호부인 셈이죠.

도메인은 보통 다음과 같은 계층 구조를 가집니다.

www
서브도메인
(호스트/용도 구분)

naver
2차 도메인
(기관·서비스 이름)

.com
최상위 도메인(TLD)
(.com .net .kr …)

브라우저에 도메인을 입력하면 내부적으로 다음 순서가 일어납니다.

1. **도메인 입력**
   사용자가 `www.naver.com` 입력
2. **DNS 질의**
   브라우저가 DNS 서버에 "이 이름의 IP가 뭐야?" 라고 물음
3. **IP 응답**
   DNS가 짝지어진 IP 주소를 돌려줌
4. **서버 접속**
   받은 IP의 해당 포트로 접속 → 페이지 표시

참고로 도메인을 IP에 연결하는 설정은 **DNS 레코드**로 합니다. 대표적으로 **A 레코드**(도메인→IPv4), **AAAA**(→IPv6), **CNAME**(도메인→다른 도메인), **MX**(메일 서버)가 있습니다. 배포 시 "내 서버의 공인 IP를 A 레코드에 등록"하는 작업을 하게 됩니다.

3 클라이언트-서버와 요청/응답

네트워크에 연결된 각 장치를 **호스트(Host)**라고 합니다. 그중 서비스를 요청하는 쪽이 **클라이언트(Client)**, 요청을 받아 처리하고 결과를 돌려주는 쪽이 **서버(Server)**입니다. 배포한다는 것은 결국 **"서버를 띄워 둔다"**는 뜻입니다.

→ **요청 (Request)**
클라이언트(브라우저)가 서버에 "이 페이지/데이터 주세요" 요청

← **응답 (Response)**
서버가 결과(HTML·데이터)를 만들어 돌려줌

HTTP와 HTTPS

웹에서 이 요청/응답을 주고받는 약속(프로토콜)이 **HTTP**입니다. 여기에 **암호화(SSL/TLS)**를 더해 도청·위변조를 막은 것이 **HTTPS**입니다. HTTPS를 쓰려면 신뢰된 기관이 발급한 **인증서(Certificate)**가 필요하며, 오늘날 실서비스는 사실상 HTTPS가 필수입니다.

구분	HTTP	HTTPS
기본 포트	`80`	`443`
암호화	없음(평문)	SSL/TLS로 암호화
인증서	불필요	필요(예: Let's Encrypt 무료 발급)

4 빌드(Build)와 컴파일(Compile)

소스코드는 그대로 서비스되지 않습니다. "실행 가능한 형태"로 가공하는 과정이 빌드·컴파일입니다.

컴파일 (Compile)

사람이 작성한 소스코드를 **컴퓨터가 이해할 수 있는 기계어(혹은 중간 코드)로 변환**하는 과정입니다. 여기서 언어는 크게 두 갈래로 나뉘며, **배포 방식이 달라지므로** 구분이 중요합니다.

구분	컴파일 언어	인터프리터(스크립트) 언어
예	C, C++, Java, Go, Rust	Python, JavaScript, PHP, Ruby
실행 방식	미리 컴파일 → 산출물 실행	실행기(런타임)가 코드를 읽으며 바로 실행
배포 시	빌드 산출물(실행파일·jar 등)을 올림	소스 + 런타임·라이브러리를 함께 갖춤

빌드 (Build)

컴파일을 포함해, 의존 라이브러리 설치·리소스 묶기·압축 등을 거쳐 "바로 실행할 수 있는 산출물(아티팩트, Artifact)"을 만드는 전체 작업을 빌드라고 합니다. 컴파일이 "번역"이라면, 빌드는 "번역 + 포장까지 끝낸 완제품 만들기"에 가깝습니다.

> 🔗 **이미 배운 것과 연결**
>
> 빌드 명령은 결국 여러 단계를 순서대로 실행하는 일입니다. 그동안 배운 **쉘 스크립트**로 "의존성 설치 → 컴파일 → 압축 → 서버 전송"을 한 번에 자동화할 수 있습니다. 이것이 뒤에 나올 배포 자동화의 출발점입니다.

대표적인 빌드 산출물(아티팩트) 예: `.jar` / `.war` (Java), 실행 파일(C/Go), `dist/` 폴더(프론트엔드), 그리고 **도커 이미지** (8장 예고).

5 어디에, 무엇으로 배포하나

5-1. 어디에 — 배포 위치

구분	온프레미스 (On-premise)	클라우드 (Cloud)
의미	직접 서버 장비를 구매·운영	AWS·GCP·Azure 등에서 서버를 빌려 사용
장점	완전한 통제, 내부 데이터 보안	빠른 시작, 필요 만큼 확장(탄력성), 관리 편의
단점	초기 비용·관리 부담 큼	사용량 기반 비용, 외부 의존

국비 실습에서는 보통 클라우드의 가상 서버(리눅스)를 받아, 그 위에 SSH로 접속해 배포·운영하는 흐름을 다룹니다.

5-2. 무엇으로 — 서버를 구성하는 요소

- **웹 서버 (Web Server)** — `Nginx`, `Apache`. 정적 파일(이미지·HTML·CSS) 처리와 외부 요청의 첫 관문 역할을 합니다.
- **WAS (Web Application Server)** — `Tomcat`, `Node.js`, `gunicorn` 등. 로그인·DB 조회처럼 매번 결과가 달라지는 **동적 처리**를 담당합니다.
- **리버스 프록시 (Reverse Proxy)** — 외부 요청을 웹 서버가 먼저 받아 내부 애플리케이션으로 전달하는 구조. 포트를 80/443으로 통일하고, HTTPS 처리·부하 분산(로드 밸런싱)을 맡깁니다.

> 🏨 **비유로 이해하기**
>
> 웹 서버(리버스 프록시)는 호텔 **프런트 데스크**입니다. 손님(요청)은 일단 프런트로 오고, 프런트가 용건에 따라 알맞은 부서(WAS·애플리케이션)로 안내합니다. 손님이 객실 번호(내부 포트)를 직접 알 필요가 없죠.

6 배포 전체 흐름 — 큰 그림

개별 개념을 하나의 줄기로 꿰어 봅니다. 이 7단계가 이번 모듈의 뼈대입니다.

1. **코드 작성**
   로컬(localhost)에서 개발·테스트
2. **빌드**
   컴파일·패키징하여 실행 가능한 산출물(아티팩트) 생성
3. **서버 준비**
   서버 OS·런타임 설치, 필요한 포트 개방(방화벽)
4. **전송**
   산출물을 서버로 옮김 — `scp`, `git pull`, 또는 CI/CD
5. **실행**
   애플리케이션을 백그라운드 프로세스(데몬)로 상시 가동
6. **도메인 연결**
   공인 IP를 DNS(A 레코드)에 등록, HTTPS 인증서 적용
7. **확인·운영**
   정상 동작 확인 후, 살아 있게 유지·관리(7장)

7 운영(Operations) 첫걸음 — 리눅스로 돌아오다

배포는 "한 번 올리면 끝"이 아닙니다. 서비스가 **끊기지 않고 살아 있도록 유지**하는 것이 운영이며, 여기서 그동안 배운 리눅스가 본격적으로 쓰입니다.

SSH — 원격 서버에 들어가기

내 책상의 PC가 아니라 멀리 있는 서버를 다뤄야 합니다. **SSH(22번 포트)**로 원격 접속하면, 그동안 배운 `cd` · `ls` · `cat` ·권한 명령을 그대로 원격 서버에서 사용할 수 있습니다.

> 🔗 **이미 배운 것이 전부 쓰이는 구간**
>
> • **프로세스 이론** → 애플리케이션을 백그라운드에서 계속 띄우는 "데몬화", `systemd` 서비스 등록(자동 시작·자동 재시작)
> • **리다이렉션/파이프라인** → 로그를 파일로 남기고(`>`), 실시간 확인(`tail -f`), 필요한 줄만 추출(`| grep`)
> • **소유자·그룹·권한** → 서비스 전용 계정으로 최소 권한 실행(보안의 기본), 파일 접근 통제
> • **쉘 스크립트** → 배포·재시작·백업을 한 번의 명령으로 자동화

운영에서 챙겨야 할 것들

- **상시 가동** — 터미널을 닫아도 꺼지지 않도록 `nohup` · `&` , 더 나아가 `systemd` 로 서비스 등록.
- **방화벽·포트** — 꼭 필요한 포트(80/443/22)만 열고 나머지는 막기(`ufw` ·클라우드 보안그룹).
- **로그·모니터링** — 로그 확인(`tail -f`), 자원 상태 점검(`ps` , `top` , `df`).
- **환경 분리·환경변수** — DB 비밀번호 같은 민감 정보는 코드가 아닌 환경변수·설정으로 분리.

> ⚠️ **자주 하는 실수**
>
> 로컬에서는 잘 되는데 서버에서 안 되는 일은 대부분 ① 방화벽에서 **포트가 막혀** 있거나, ② 권한 부족, ③ 환경변수 누락, ④ 터미널을 닫자 **프로세스가 같이 종료**되어서입니다. "내 PC에선 됐는데"의 90%는 이 네 가지입니다.

8 한 걸음 더 (이번 모듈 예고)

- **배포 자동화** — 배운 쉘 스크립트로 "빌드→전송→재시작"을 한 번에. 사람의 실수를 줄이는 첫 단계.
- **컨테이너 (Docker)** — 애플리케이션과 그 실행 환경을 **통째로 한 상자에 포장**해, "내 PC에선 됐는데" 문제를 근본적으로 줄이는 기술. **어디서나 똑같이 실행**되는 것이 핵심.
- **CI/CD** — 코드를 올리면 **자동으로 빌드·테스트·배포**까지 이어지는 파이프라인. 협업과 빠른 배포의 표준.

★ 핵심 용어 한눈에

용어	한 줄 정의
배포	내 애플리케이션을 외부에서 쓸 수 있게 서버에 올려 서비스하는 것
localhost / 127.0.0.1	나 자신을 가리키는 루프백 주소. 외부에서는 접근 불가
IP 주소	네트워크상 장치의 고유 식별 번호(=물리적 주소)
공인/사설 IP	인터넷 전체에서 식별 / 내부 네트워크에서만 식별
포트	한 컴퓨터 안에서 프로그램을 구분하는 번호(=호실)
도메인	사람이 기억하는 이름(naver.com)
DNS	도메인을 IP로 변환해 주는 서비스(인터넷 전화번호부)
HTTP/HTTPS	웹 요청·응답 규약 / 거기에 암호화를 더한 것
컴파일	소스코드를 컴퓨터가 이해할 언어로 변환
빌드	컴파일·포장까지 마쳐 실행 가능한 산출물을 만드는 전 과정
웹 서버 / WAS	정적 처리·관문 / 동적 처리 담당
SSH	원격 서버에 안전하게 접속하는 방법(22번 포트)