UNICA
UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

sAIfer Lab
Joint lab on Safety and Security of AI

Deployment

Maura Pintor
maura.pintor@unica.it

Cos’è il Deployment

Il deployment è il processo che porta un’applicazione dal codice sorgente all’ambiente di produzione, rendendola disponibile agli utenti.

Obiettivi:

- Disponibilità online – il servizio viene reso disponibile
- Sicurezza – prima di rilasciare il servizio, si deve verificare bene che sia robusto ad attacchi
- Scalabilità – bisogna capire quante risorse serviranno al servizio quando sarà in uso

Ambienti di Esecuzione

È bene avere ambienti separati per evitare errori catastrofici

- per esempio, lo sviluppo di nuove funzionalità deve essere ben isolato dal servizio che è attivo e usato dagli utenti al momento
- Gli utenti saranno avvisati quando si aggiornerà il codice sul servizio attivo, ma questo viene fatto solo dopo aver verificato che funzioni tutto in un ambiente di pre-deployment

Gli ambienti vengono solitamente suddivisi in:

- **Development:** ambiente locale degli sviluppatori
- **Testing / Staging:** Identico all'ambiente di produzione, ma serve per testare in sicurezza, usando dati fittizi o replicati
- **Production**: Ambiente usato dagli utenti reali, dev'essere caratterizzato da alta disponibilità e sicurezza

Metodi di deployment

Deployment Manuale (FTP/SSH)

- Copia manuale dei file sul server via FTP, SFTP o SSH
- Esecuzione diretta di comandi (scp, rsync, nano, vi, systemctl, ecc.)
- Solo per prototipi locali o ambienti temporanei (errore umano, nessuna tracciabilità)

Deployment con Script Automatizzati

- Scrittura di script (bash, Makefile, Fabric, Ansible) che automatizzano le fasi del deploy: copia file, riavvio servizi, migrazioni DB…
- Richiede manutenzione continua
- Poco flessibile per ambienti dinamici o con molte variabili

Metodi di deployment

CI/CD (Continuous Integration / Deployment)

- Workflow automatizzato che esegue test e deploy dopo ogni push
- Strumenti: GitHub Actions, GitLab CI, Jenkins
- Configurazione iniziale più complessa, ma esegue test automatici e deployment tracciabili
- Integrazione diretta con Git (si può fare deployment si possono eseguire i test *on push*)

Deployment con Docker / Docker Compose

- Si crea un container con l'app e le sue dipendenze
- Con Docker Compose si gestiscono più container (es. app + DB)
- Facile da replicare/scalare, coerenza tra ambienti (dev/prod)
- Richiede familiarità con Docker e non gestisce da solo il deploy (richiede CI)

Metodi di deployment

Deployment su Platform As A Service - PaaS (Heroku, Render, Railway)

- Hosting gestito: basta caricare il codice (via Git o UI)
- Build automatico dell'app + deploy
- Meno controllo su configurazioni avanzate

Deployment su Internet As A Service -IaaS (es. AWS EC2, GCP Compute, DigitalOcean)

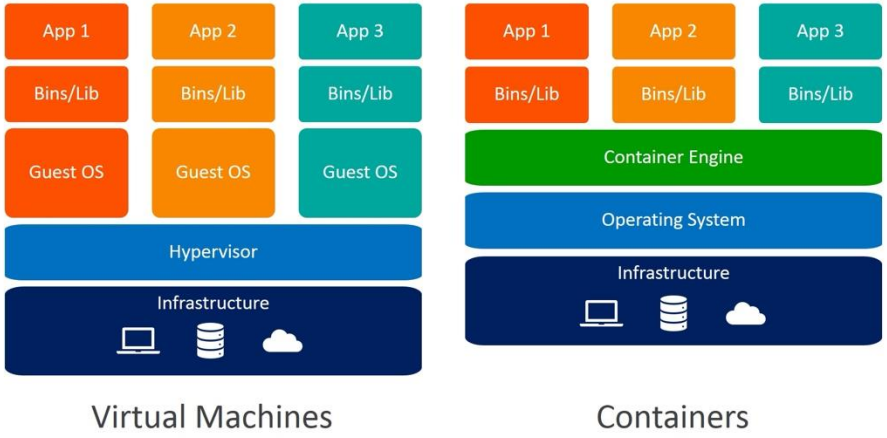
- Un server virtuale completamente gestibile (come avere un PC remoto)
- Puoi configurare tutto: OS, firewall, software, script
- Massimo controllo, alta scalabilità
- Tutta la responsabilità è lato sviluppo (sicurezza, aggiornamenti, logging), e richiede conoscenze avanzate di sysadmin e rete

Metodi di deployment

Metodo	Pro	Contro
Manuale (FTP, SSH)	Facile da iniziare	Rischi alti, poco scalabile
Script automatici	Ripetibile, più sicuro	Va mantenuto
CI/CD (GitHub Actions, GitLab)	Completamente automatizzato	Richiede setup iniziale
PaaS (Heroku, Render)	Velocissimo per iniziare	Meno controllo
IaaS (AWS, GCP, VPS)	Flessibilità massima	Più complesso

Virtualizzazione e container

- Container = ambiente isolato con tutto ciò che serve per eseguire l'app
- Differenze da VM:
  - Leggero
  - Veloce da avviare
  - Più portabile



Esempio Dockerfile (file di configurazione):

```
FROM node:20-alpine
WORKDIR /app
COPY . .
RUN npm install
CMD ["npm", "start"]
```

Ambienti multi-container: docker-compose

Docker-compose è uno strumento per avviare applicazioni multi-container
Facilita la gestione dei servizi, la scalabilità, e le connessioni grazie a file di configurazione YAML

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

Continuous Integration / Cont. Deployment e Delivery (CI/CD)

- Continuous Integration (CI): build, test, e integrazione di modifiche al codice in un repository condiviso
- Continuous Delivery (CD): consegna automaticamente le modifiche del codice all'ambiente di produzione per essere approvate e messe in funzione
- Continuous Deployment (CD): invia il codice automaticamente ai clienti

Messi insieme, formano una pipeline di CI/CD, ovvero un flusso di lavoro che permette agli sviluppatori di risparmiare tempo, automatizzando i task ripetitivi

```
on: [push]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: docker build -t myapp .
      - run: docker run myapp
```

Configurazione e .env

Tra le best practice di sviluppo e configurazione per deployment, uno degli aspetti più importanti è quello di **non scrivere mai credenziali nel codice**

Si usano file di configurazione, di solito per convenzione chiamati .env (dot-env), per variabili sensibili

- impostano variabili di ambiente, ovvero relative solo all'ambiente in cui saranno usate (non sono condivise con altri)

Questi file non vengono messi nel version control e non sono pubblici, ma vengono configurati e modificati in locale (o creati in fase di setup dell'applicazione)

Ci sono librerie per interagire direttamente con questo tipo di file:

- python-dotenv, dotenv in Node.js

```
DATABASE_URL=postgres://user:pass@host/db SECRET_KEY=abc123
```

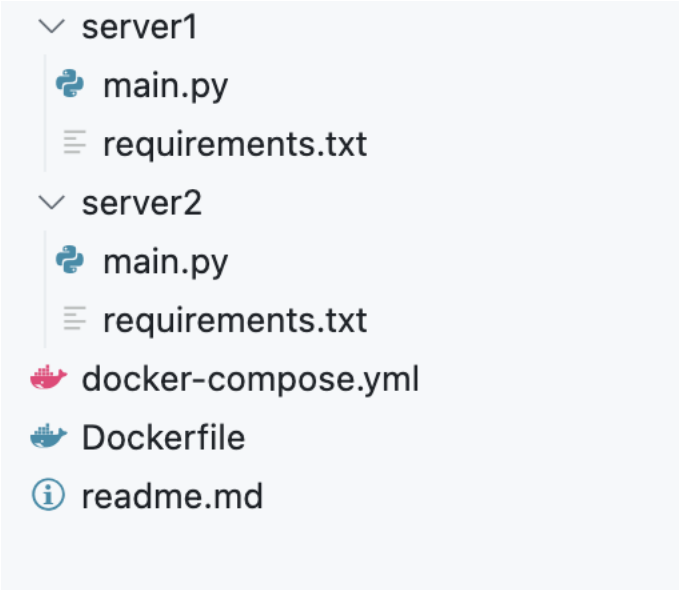
Sicurezza in produzione

- HTTPS sempre (Let's Encrypt)
- Rate limiting e protezione da DoS
- Validazione input lato server
- Accessi e permessi ben definiti
- Backup regolari del DB

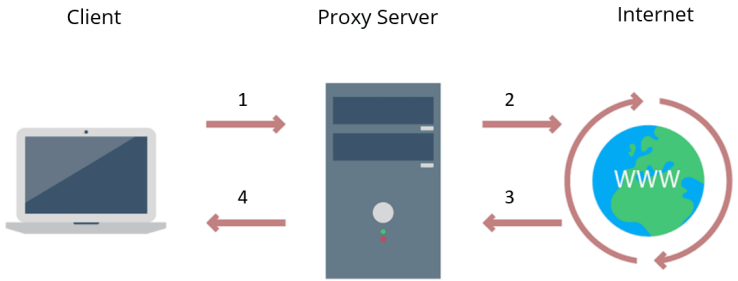
Il logging è importante per capire cosa succede in produzione
- Output di errore, eventi, performance, monitoraggio
- Rivelare tentativi di attacco

Evitare errori comuni:
- Deploy manuali non tracciati
- Dipendenze non bloccate (niente requirements.txt)
- Password nel codice
- Mancanza di logging e rollback

Esercitazione con docker-compose



Implementeremo un proxy server (un server intermediario che fa da interfaccia tra l'utente e un altro server)



https://github.com/maurapintor/docker-compose-tutorial