ARCHITETTURE SOFTWARE

Principi di scalabilità, resilienza, monoliti e microservizi, microfrontend

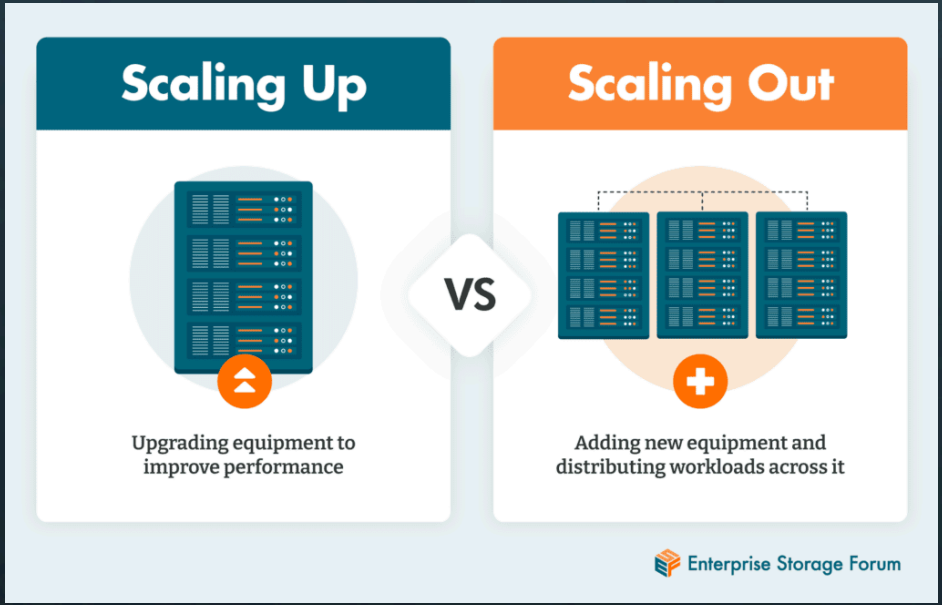
> *"L'architettura è troppo importante per essere lasciata agli architetti"* Giancarlo De Carlo

Voi chi siete? 🤔

- Studenti del **secondo anno** front-end
- Conoscenze acquisite:
  - JavaScript ✅
  - React ✅
  - Sviluppo web di base ✅

Finalità del Corso

Acquisire i principi di base per la progettazione di architetture software basate su tecnologie tipiche di Internet, scalabili e robuste



Cosa Significa "Scalabile"?

- **Scalabilità verticale**: Aggiungere potenza (CPU, RAM)
- **Scalabilità orizzontale**: Aggiungere più server

> *"Scale up until you can't, then scale out"* — Motto degli architetti software

Cosa Significa "Robuste"?

- **Resilienza**: Capacità di riprendersi da errori
- **Fault tolerance**: Continuare a funzionare anche con componenti guasti
- **High availability**: Essere sempre disponibili

Il Panorama Attuale

Da dove veniamo

- **Monoliti**: Un’applicazione, un server
- **Three-tier**: Presentation, Business, Data

Dove andiamo

- **Microservizi**: Tanti piccoli servizi
- **Cloud-native**: Pensato per il cloud
- **Event-driven**: Guidato da eventi

HTTP: Il Linguaggio del Web

- Caratteristiche del protocollo HTTP
- Stateless vs Stateful
- Headers, methods, status codes
- HTTP/1.1, HTTP/2, HTTP/3

Load Balancers & Reverse Proxy

- Distribuzione del carico
- Alta disponibilità
- Configurazione pratica
- Algoritmi di bilanciamento

Content Delivery Networks

- Distribuzione geografica dei contenuti
- Edge computing
- Caching strategies
- Performance optimization

Microservizi: La Rivoluzione

- Teoria alla base dei microservizi
- Potenzialità e criticità
- Domain-driven design
- Service boundaries

Scalabilità e Ridondanza

- Scalabilità orizzontale vs verticale
- Patterns di ridondanza
- Auto-scaling
- Performance monitoring

Team e Microservizi

- Conway’s Law
- Team topologies
- DevOps culture
- Ownership dei servizi

Disaccoppiamento Architetturale

- Message queues
- Event-driven architecture
- Publish/Subscribe patterns
- Async communication

Configurare Strumenti

- **Load balancers**
- **Reverse proxy** configurazioni
- **SSL/TLS** certificati
- **Monitoring** e logging

Gestire Sessioni

- **Stateless** authentication
- **JWT** tokens
- **Session stores** distribuiti
- **OAuth** e SSO

> *"Il miglior modo di gestire le sessioni è... non gestirle!"*

Articolo

Microservizi: Pro e Contro

Pro ✅

- Scalabilità indipendente
- Team autonomi
- Tecnologie diverse per servizio
- Fault isolation

Contro ❌

- Complessità di rete
- Data consistency
- Testing distribuito
- Monitoring complesso

Coordinamento dei Servizi

- Service discovery
- Circuit breakers
- Retry patterns
- Timeout strategies

Scambio di Messaggi

- **Synchronous** vs **Asynchronous**
- **REST** APIs
- **GraphQL**
- **Message brokers** (RabbitMQ, Kafka)

tutto molto bello, ma noi frontendisti...?

Il Vostro Background: JavaScript

Vantaggi per questo corso

- **Node.js**: Runtime server-side
- **Express**: Framework web
- **Async/Await**: Programmazione asincrona
- **JSON**: Formato di scambio dati

React: Più Rilevante di Quanto Pensiate

Concetti che si applicano alle architetture

- **Component isolation** → Service isolation
- **Props down, events up** → API design
- **State management** → Data consistency
- **Hooks** → Service lifecycle

L’Evoluzione delle Architetture

```
Mainframe (1960s)
    ↓
Client-Server (1980s)
    ↓
Three-Tier (1990s)
    ↓
Service-Oriented (2000s)
    ↓
Microservices (2010s)
    ↓
Serverless (2020s)
```

Caso di Studio: Netflix 🎬

Architettura a microservizi

- **700+ microservizi**
- **1+ miliardo** di ore di video al giorno
- **190+ paesi** serviti
- **99.99%** di uptime

Caso di Studio: Monolite vs Microservizi

Amazon nel 2001

- **Un monolite** gigantesco
- **Deploy** ogni 11.6 secondi oggi
- **Migliaia** di team indipendenti

La trasformazione

- Da 1 applicazione a **centinaia** di servizi
- Da 1 team a **migliaia** di team
- Da deploy mensili a deploy **ogni secondo**

Le Sfide Reali

> *"There are only two hard things in Computer Science: cache invalidation and naming things"* — Phil Karlton

E nel nostro caso aggiungiamo:

- **Network partitions**
- **Distributed transactions**
- **Service versioning**

Conway’s Law 👥

> *"Organizations which design systems... are constrained to ooduce designs which are copies of the communication structures of these organizations"* — Melvin Conway (1967)

In parole semplici:

Se avete 4 team, farete un sistema con 4 componenti!

Fallacies of Distributed Computing 🌐

Le **8 false credenze** sui sistemi distribuiti:

1. La **rete** è affidabile
2. La **latency** è zero
3. La **bandwidth** è infinita
4. La rete è **sicura**
5. La **topologia** non cambia
6. C'è **un amministratore**
7. Il **costo** di trasporto è zero
8. La rete è **omogenea**

CAP Theorem

Puoi averne solo 2 su 3:

- **C**onsistency (Consistenza)
- **A**vailability (Disponibilità)
- **P**artition tolerance (Tolleranza alle partizioni)

> *"In the presence of a network partition, you must choose between consistency and availability"* — Eric Brewer

BASE vs ACID

ACID (Database tradizionali)

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

BASE (Sistemi distribuiti)

- **B**asically **A**vailable
- **S**oft state
- **E**ventual consistency

Microservices vs SOA 🔄

Service-Oriented Architecture (SOA)

- **Enterprise Service Bus** (ESB)
- **SOAP** / **WSDL**
- **Heavy governance**

Microservices

- **Smart endpoints**, dumb pipes
- **RESTful** APIs
- **Decentralized governance**

The Twelve-Factor App 📋

Metodologia per app **cloud-native**:

1. **Codebase**: One codebase, many deploys
2. **Dependencies**: Explicitly declare dependencies
3. **Config**: Store config in environment
4. **Backing services**: Treat as attached resources
5. **Build, release, run**: Separate stages
6. **Processes**: Stateless and share-nothing
7. **Port binding**: Export services via port binding
8. **Concurrency**: Scale out via process model
9. **Disposability**: Fast startup and shutdown
10. **Dev/prod parity**: Keep environments similar
11. **Logs**: Treat logs as event streams
12. **Admin processes**: Run as one-off processes

Domain-Driven Design 📚

Concetti chiave

- **Bounded Context**: Confini logici del dominio
- **Aggregates**: Cluster di entità correlate
- **Domain Services**: Logica di business
- **Ubiquitous Language**: Linguaggio condiviso

> *"The heart of software is its ability to solve domain-related problems for its user"* — Eric Evans

Event Storming

Tecnica per modellare domini complessi

- **Domain Events**: Cosa è successo?
- **Commands**: Cosa vogliamo che succeda?
- **Aggregates**: Chi decide?
- **Policies**: Regole di business

Patterns Architetturali

Comuni nelle architetture moderne

- **API Gateway**
- **Circuit Breaker**
- **Bulkhead**
- **Saga Pattern**
- **CQRS** (Command Query Responsibility Segregation)
- **Event Sourcing**

Observability: I Three Pillars

Per sistemi distribuiti

1. **Metrics**: Numeri aggregati nel tempo
2. **Logs**: Eventi discreti
3. **Traces**: Richieste attraverso i servizi

> *"Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs"*

DevOps e Cultura

Non è solo tooling!

- **Collaboration** tra Dev e Ops
- **Shared responsibility**
- **Automation** first
- **Fail fast**, learn faster

> *"DevOps is not a goal, but a never-ending process of continual improvement"*

Container e Orchestrazione

Docker

- **Containerization** delle applicazioni
- **Portability** cross-platform
- **Immutable infrastructure**

Kubernetes

- **Orchestrazione** dei container
- **Auto-scaling**
- **Service discovery**

Serverless Computing

Function as a Service (FaaS)

- AWS Lambda
- Azure Functions
- Google Cloud Functions

Vantaggi

- No server management
- Automatic scaling
- Pay per execution

Frontend e Architetture 🎨

Micro-frontends

- **Independent deployment**
- **Technology diversity**
- **Team autonomy**

Jamstack

- **JavaScript**, **APIs**, **Markup**
- **Pre-built** markup
- **Serverless** functions

Security by Design

Principi fondamentali

- **Zero Trust** Architecture
- **Defense in Depth**
- **Least Privilege**
- **Security as Code**

> *"Security is not a product, but a process"* — Bruce Schneier

Performance e Ottimizzazione

Metriche chiave

- **Latency**: Tempo di risposta
- **Throughput**: Richieste al secondo
- **Availability**: Uptime percentuale
- **Error Rate**: Percentuale di errori

Testing in Architetture Distribuite

Test Pyramid

- **Unit Tests**: Singoli componenti
- **Integration Tests**: Interazioni tra servizi
- **Contract Tests**: API contracts
- **End-to-End Tests**: Flussi completi

Monitoring e Alerting

Strategie efficaci

- **Proactive monitoring**
- **SLI/SLO/SLA** (Service Level Indicators/Objectives/Agreements)
- **Error budgets**
- **On-call rotations**

Database in Architetture Distribuite

Patterns comuni

- **Database per servizio**
- **Shared databases** (anti-pattern)
- **Data synchronization**
- **Eventual consistency**

API Design Best Practices

RESTful APIs

- **Resource-based** URLs
- **HTTP methods** appropriati
- **Status codes** significativi
- **Versioning** strategy

GraphQL

- **Schema-first** approach
- **Type safety**
- **Single endpoint**

Caching Strategies 💾

Livelli di caching

- **Browser cache**
- **CDN cache**
- **Reverse proxy cache**
- **Application cache**
- **Database cache**

> *"There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors"*

Message Queues e Pub/Sub

Async Communication

- **Decoupling** dei servizi
- **Reliability** through persistence
- **Scalability** through parallelism

Popular Tools

- RabbitMQ
- Apache Kafka
- Amazon SQS
- Redis Pub/Sub

Deployment Strategies

Strategie moderne

- Blue-Green Deployment
- Canary Releases
- Rolling Updates
- Feature Flags

Infrastructure as Code 

Gestione dell’infrastruttura

- **Terraform**
- **CloudFormation**
- **Ansible**
- **Pulumi**

> *"Cattle, not pets"* - Tratta i server come bestiame, non come animali domestici

Cost Optimization

Nel cloud

- **Right-sizing** resources
- **Reserved instances**
- **Spot instances**
- **Auto-scaling** policies

Disaster Recovery

Strategie di backup

- **RTO** (Recovery Time Objective)
- **RPO** (Recovery Point Objective)
- **Multi-region** deployment
- **Data replication**

Compliance e Governance

Requisiti enterprise

- **GDPR** compliance
- **SOX** compliance
- **Audit trails**
- **Data governance**

Future Trends

Cosa ci aspetta

- **Edge Computing**
- **AI/ML** integration
- **Quantum computing** impact
- **WebAssembly** adoption

Tools del Mestiere

Development

- **Docker** & **Kubernetes**
- **Terraform** & **Ansible**
- **Jenkins** / **GitLab CI**

Monitoring

- **Prometheus** & **Grafana**
- **ELK Stack** (Elasticsearch, Logstash, Kibana)
- **Jaeger** (distributed tracing)

Metodologie Agili

Scrum & Kanban

- **Sprint planning**
- **Daily standups**
- **Retrospectives**
- **Continuous improvement**

DevOps Integration

- **CI/CD** pipelines
- **Infrastructure automation**
- **Monitoring integration**

Team Topologies

Secondo il libro di Matthew Skelton

- **Stream-aligned** teams
- **Enabling** teams
- **Complicated-subsystem** teams
- **Platform** teams

Cognitive Load

Gestire la complessità

- **Essential complexity**: Inerente al problema
- **Accidental complexity**: Aggiunta dalla soluzione
- **Team cognitive load**: Quanto può gestire un team

Documentation as Code

Living Documentation

- **Architecture Decision Records** (ADRs)
- **API documentation** (OpenAPI/Swagger)
- **Runbooks** e **playbooks**
- **Code comments** e **README**

Incident Management

Quando le cose vanno male

- **Incident response** procedures
- **Post-mortem** analysis
- **Blameless culture**
- **Learning from failures**

> *"Every outage is a learning opportunity"*

Chaos Engineering

Testing in Production

- Netflix's Chaos Monkey
- Fault injection
- Resilience testing
- Game days

> *"Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions"*

Recap della Lezione 1 

Cosa abbiamo visto oggi

- **Finalità** del corso e obiettivi di apprendimento
- **Panoramica** delle architetture moderne
- **Evoluzione** da monoliti a microservizi
- **Principi fondamentali** di scalabilità e resilienza
- **Sfide** dei sistemi distribuiti
- **Strumenti** e metodologie moderne

Prossime Lezioni 

Programma delle prossime 8 lezioni

- **Lezione 2**: Protocollo HTTP e fondamenti web
- **Lezione 3**: Load Balancers e Reverse Proxy
- **Lezione 4**: Content Delivery Networks (CDN)
- **Lezione 5**: Introduzione ai Microservizi
- **Lezione 6**: Scalabilità e Ridondanza
- **Lezione 7**: Gestione team e microservizi
- **Lezione 8**: Code di messaggi e disaccoppiamento
- **Lezione 9**: Sessioni, SSL e best practices

Domande?

Discussione aperta

- Dubbi sui concetti introdotti?
- Curiosità su argomenti specifici?
- Esperienze personali da condividere?

Preparazione per la Lezione 2 📖

Cosa rivedere

- **Basics del protocollo HTTP**
- **Request/Response cycle**
- **HTTP methods** (GET, POST, PUT, DELETE)
- **Status codes** comuni

Cosa installare

- **Postman** o **curl** per test HTTP
- **Node.js** per gli esercizi pratici

Risorse Aggiuntive 🔗

Letture consigliate

- **Building Microservices** - Sam Newman
- **Designing Data-Intensive Applications** - Martin Kleppmann
- **Site Reliability Engineering** - Google
- **The Phoenix Project** - Gene Kim

Website utili

- **High Scalability**
- **AWS Architecture Center**
- **Microservices.io** - Chris Richardson

Citazioni Finali 

> *"Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure"* — Conway's Law

> *"There are no solutions, only trade-offs"* — Principio architetturale fondamentale

> *"Premature optimization is the root of all evil"* — Donald Knuth

Grazie!

The End

Keep calm and architect on 🏗️