**PROJECT MANAGEMENT**

**L’obiettivo del project managemement** è gestire un progetto e le persone che ci lavorano, identificare l’obiettivo del progetto, stimare la quantità di lavoro e crearne una schedulazione.

**I progetti sono spesso in ritardo per**: - pessima pianificazione

- poco monitoraggio dello sviluppo

- cattiva comunicazione tra i componenti ecc...

**Il project management serve a**: - Definire e pianificare (sapere cosa si deve fare, si identificano i task)

- Schedulare (sapere l’ordine di schedulazione dei task)

- Controllare che i piani vadano come pianificati

**SPIKE** = Demo funzionanti della tecnologia che ci serve da cui partirà poi il progetto

**FASI DEL PROJECT MANAGEMENT** Definire – Pianificare – Schedulare – Controllare il progetto

**Il planning** Permette di stabilire obiettivi concreti e stimare la portata del progetto (ANTICIPA I PROBLEMI)

E’ necessario determinare le priorità, preparandosi a risolvere conflitti e a modificare i piani con revisioni e controlli.

Cause del fallimento del planning: Non considerare ogni persona coinvolta, non avere piani B, …

**Gestire un progetto** La gestione del progetto è legata a task, risorse, tempo e costi

Un progetto è un insieme di TASK eseguiti in una certa SEQUENZA per ottenere un

OBIETTIVO, con una SCHEDULAZIONE e una quantità di RISORSE

**Il modello delle 4 fasi -** Definizione del progetto

si definiscono obiettivi e fattori limitanti (risorse, tempo…)

* Creazione e definizione di un project plan

Viene utilizzato per predire e controllare il progetto, e fare Scheduling

* Controllo (traking) del progetto e revisione del project plan
* Chiusura del progetto

Analisi post mortem: si analizza il progetto a progetto finito per capire i problemi e migliorare in futuro

Il successo del progetto dipende da: task, risorse, tempo e cosri

**Il progect management tradizionale** C’è una lista di task in ordine sequenziale

**Milestone** Specificare milestone e checkpoint

**Linee guida per identificare i task** Identificare i task in modo preciso, il livello di dettaglio deve essere

coerente con la quantità di pianificazione e controllo desiderata.

**I GANT CHART** Rappresentano graficamente le dipendenze dei task mostrando la durata di ciascun task

**I PERT CHART** Sono una alternativa ai gantt ma hanno un focus sulle dipendenze piuttosto che alla calendariz.

**Limiti del planning tradizionale – le 3 bestie** Incertezza rispetto alle aspettative del cliente e all’evoluzione

Complessità Il planning è essenziale ma difficile

Irreversibilità

**PEOPLE MANAGEMENT**

L’obiettivo del people management è raggiungere un’alta produttività imparando a gestire un gruppo.

Sono necessari: energia e impegno tra i collaboratori, team unito, imparare a risolvere problemi…

**Esempi di management**  Activity-based management

Si gestiscono le attività tipo “catena di montaggio”

Learn management

Si basa su 5 principi: VALUE il valore è definito dal punto di vista cliente

VALUE STREAM

FLOW le attività devono fluire senza interruzioni

PULL fare qualcosa solo se richiesto da un processo

PERFECTION continui miglioramenti

**Il project manager** Deve avere delle caratteristiche fondamentali

* Challenging the process: cercare sempre opportunità per migliorare,

sperimentare e accettare il rischio

* Ispirare una visione condivisa: Coinvolgere e ispirare
* Rendere gli altri capaci di agire: incoraggiare
* Modellare la strada: dare l’esempio

**Linee guida per il people management**

Gli obiettivi devono essere: specifici, realistici, Realizzabili, Misurabili, Con deadline e con commitment

**Cercare di essere dei buoni ascoltatori**

Active listening: Verbalizzare i problemi aiuta ad alleggerirli (effetto confortante), migliora i rapporti tra

dipendente e responsabile, aiuta a pensare prima di parlare

**Come misurare un ascoltatore** Ascolta prima di parlare?

**Motivare non manipolare**  - Motivare: fare leva verso obiettivi comuni attraverso riconoscimento del

lavoro, opportunità di crescita ecc.

- Manipolare: Direzionare in modo scaltro e sleale per i propri tornaconti

**STILI DI MANAGEMENT**

* Manager come tecnico: Funziona se il manager ha una maggiore conoscenza

tecnica del team.

Svantaggi: Dirige troppo da vicino, c’è un bypass del

planning funzionale, focus sulla tecnologia

e non sul fattore umano

* Manager come conduttore: Funziona se i membri del team sono esperti e i task

semplici

Ha un ampio e costante controllo

Responsabilità unica non condivisa dal team

Dirige senza fare il lavoro personalmente

Svantaggi: serve troppa coordinazione

* Manager come sviluppatore: Permette di avere un ottimo risultato (eccellenza)

Ha impatto senza esercitare il controllo

**Gestione del team** Il focus è sul continuo sviluppo delle capacità individuali, bisogna saper lavorare in collaborazione, comunicare in modo aperto e diretto, ottenere il massimo da tutti

**Meeting** Group meeting: Stabilire l’agenda, comunicare ciò che avviene in ditta, sentire da ciascuno come

va il progetto, discutere

One meeting: Discutere i possibili problemi, ascoltare

Stand up meeting: stando in piedi per evitare che duri troppo

**Performance review** Sono valutazioni molto dettagliate e puntuali, necessario focalizzarsi sul problema di

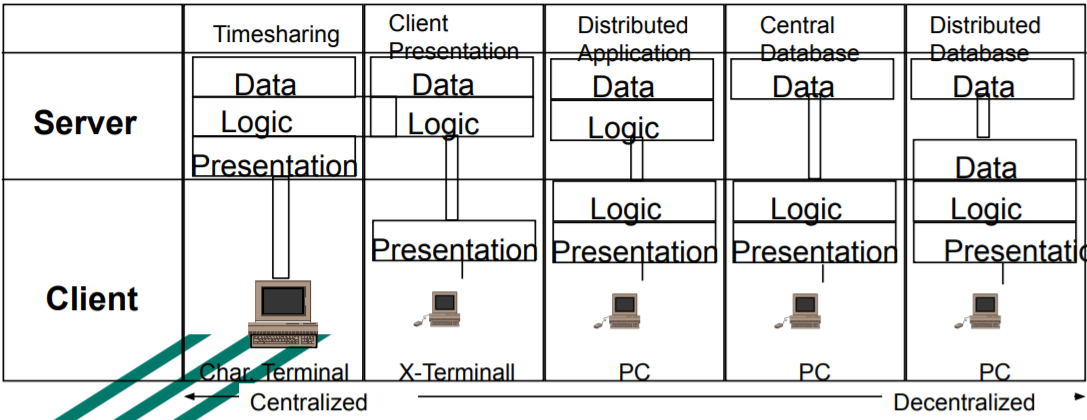
performance e non sulla persona, MIGLIORARE LA PERFORMANCE MEDIA

**SOFTWARE ARCHITECTURE**

ARCHITETTURE CLIENT SERVER

Architettura software per la quale un computer client si connette al server per usufruire di un servizio

La classificazione delle architetture avviene in base a dove si trovano DATA, LOGIC e PRESENTATION



**THIN CLIENT** Time sharing

Client presentation

**FAT CLIENT** Central database

Distributed database

**Time sharing** Data, Logic e Presentation tutti sul server, il client manda comandi al server tramite un

Char terminal e stampa solo l’output, NO PARALLELISMO viene seguito un terminal per volta

**Client presentation** La presentazione si sposta sul client e si usa X-terminal con le prime interfacce grafiche

**Distrubuted Application** la logica si spalma tra client e server

**Central Database** Sul server troviamo soltanto il database, tutta la logica di controllo è sul client

Two Phase Commit (2PC) I commit avvengono in 2 fasi, si manda un PREPARE, se tutti

rispondono positivamente allora si manda il COMMIT

Stored procedures Sono procedure effettuate all’interno del DB come pre-operazioni

sui dati

**Distributed Database** Dati parzialmente duplicati, potenziale problema di inconsistenza

Replicazioni asincrone Per garantire maggior scalabilità è possibile usare architetture

distribuite ma bisogna risolvere il problema della replicazione

dei dati

ARCHITETTURE MIDDLEWARE

È un software che permette l’interazione tra programmi in un ambiente distribuito

**TP MONITOR**  Middleware che fornisce supporto alle TRANSAZIONI

Permettono che un insieme di transazioni venga eseguito completamente o annullato

Garantisce le proprietà Atomic Consistent Isolata Durevole

Fornisce scalabilità e funneling per sistemi di grandi dimensioni

Fornisce Load balancing

FUNNELING: Se i client richiedono una risorsa ma non la usano tutti insieme il middleware

alterna l’assegnazione ai soli client che la utilizzano

**OBJECT REQUEST BROKER (ORB)**

Si basa sulla programmazione ad oggetti

ORB è un middleware che permette di effettuare chiamate tra computer diversi in una rete

gestendo la trasformazione di strutture dati in byte e viceversa

un esempio è   
 **CORBA**

Corba è uno standard per architetture distribuite ad oggetti, rende possibile la

comunicazione tra software scritti in linguaggi ad oggetti diversi e distribuiti

**IDL** Interface Definition Language È un’astrazione usata in CORBA per separare le

interfacce degli oggetti dalle loro implementazioni

**MESSAGE ORIENTED MIDDLEWARE (MOM)**

È una infrastruttura client-server che distribuendo una app tra più piattaforme eterogenee

ne incrementa interoperabilità, portabilità e flessibilità

In sostanza il programmatore può ignorare i dettagli sui protocolli di rete, sistemi operativi

* Il middleware sta sia su client che su server e consente l’esecuzione di chiamate asincrone

tra i due

Fire and forget: Il mittente può mandare msg e poi continuare l’esecuzione

Publish and subscribe

**AGILE SOFTWARE ENGINEERING**

**Lifecycle** Analisi dei requisiti

Design

Codifica

Testing

Manutenzione

I modelli agili si contrappongono al modello a cascata e lavorano in modo iterativo e incrementale

**Modello a cascata** è il modello tradizionale, la sequenzialità è difficile da rispettare, incertezza e consegna

solo alla fine

**Modello incrementale** Il sistema viene consegnato a pezzi, prima vengono sviluppati gli elementi ad alta

priorità

**Modello a spirale** E’ un modello iterativo la cui caratteristica principale è l’analisi del rischio

EXTREME PROGRAMMING (XP)

È un processo di sviluppo software EVOLUTIVO basato sulle ITERAZIONI

È incentrato sul CLIENTE e sui requisiti software da lui richiesti

VALORI: Semplicità, comunicazione, feedback, abbracciare il cambiamento

**Principi base** Feedback rapido, Semplicità, Cambiamento incrementale, …

Si svolge prima la priorità alta e poi si itera per quelle più basse

Pair programming: due sviluppatori alla stessa postazione

**Release corte:** È più facile effettuare testing e modifiche se il cliente non rimane spaesato

Major Release: Release di una nuova funzionalità

Minor Release: fix di bug

Integrazione continua: le modifiche apportate al software vengono integrate molto spesso

REAFACTORING FONDAMENTALE in quanto non c’è un’accurata fase di progettazione si modifica durante lo

sviluppo

**Fasi di XP**

User stories + Spike architetturali

Release plan

Iteration -> Acceptance test (con il cliente) -> Small release

KANBAN E’ una lavagna con i todo, riduce lo stress e aumenta la produttività

**AGILE PLANNING**

Se il progetto è complesso è necessario gestire i cicli

PIANIFICAZIONE A BREVE TERMINE E IN PIU FASI -> CICLI

Time of commitment -> E’ la frequenza con la quale si mantengono le “promesse” fatte al cliente

Il committente è fortemente coinvolto

Standup meeting

Test PRIMA della codifica

Per evitare SIDE EFFECTS rifare tutti i test ad ogni funzionalità aggiunta, nel caso REGRESSION TEST

**OO ANALISYS AND DESIGN**

Prima della fase di definizione (Design) c’è una fase di progettazione in cui non si ha ancora idea delle entità che compariranno nel dominio (Analisi)

Nell’Analisi si identificano le entità principali

Verificare i requisiti è importante, lo si fa tramite le User stories, ovvero requisiti scritti dal cliente in maniera discorsiva e senza termini tecnici, devono essere goal oriented e con approccio SCUTTERGUN ovvero generate dall’aspettativa del cliente

**Acceptance test** Le user stories sono la base per scrivere i test cases.

**Analisis ->** I concetti del sistema tratti dalle user stories vengono analizzati e si definiscono tramite le CRC

**Design ->** L’output dell’analisi viene usato per definire classi e oggetti

**Analisis**  
l’approccio object oriented permette di concentrarsi sui concetti importanti del progetto invece che sulle funzionalità tecniche

OOA Model: descrive gli oggetti principali del dominio e le operazioni da effettuare su di essi

Gli oggetti hanno uno STATO (cosa sono) e un COMPORTAMENTO (cosa fanno)

Gli oggetti sono istanze di CLASSI

**CRC Cards**  Class entità che compongono il dominio

Responsability Attributi e metodi delle classi

Collaborations Relazioni tra le classi

Whalktrhoug si analizzano gli use cases per vedere se le CRC cards sono sufficienti

**Design**

Diagrammi statici: Indicano le relazioni tra gli oggetti (class diagram)

Diagrammi dinamici: indicano ciò che succede in fase di esecuzione (sequence diagram)

**PATTERN ARCHITETTURALI (vedere slide)**

**Pipe and filters** Un output di un programma è l’input di un altro

**Data Abstraction and OO Organization** Gli oggetti collaborano tra loro tramite scambi di msg

**Layered system** Architettura a layer tipo quella dei SO

**Repositories** Il sistema usa una blackboard, gli agenti per comunicare scrivono sulla blackboard

**Table driven interpreters**

**SERVIZI WEB**

**SERVLET**

Programmazione server side

Le applicazioni web sono applicazioni accessibili in rete ed eseguite su un server remoto

Per far girare app web sono necessari WEB SERVER che catturano le richieste http

WEB CONTAINER che ospitano l’app sul server

Le servlet sono strumenti di base per sviluppare app web serverside

Ogni applicazione web è identificata dal contesto dell’applicazione

MODELLO AD EVENTI: Un programma basato sul modello a eventi consiste in un insieme di procedure

detti event handler che vengono scatenate da un certo evento (richieste http)

**Sessione utente** Protocollo stateless: ogni richiesta è gestita in una nuova sessione

Protocollo stateful: si mantiene traccia dell’interazione tra client e server

La sessione utente serve a riconoscere la sequenza di azioni effettuate e mantenere una memoria associata all’interazione

**JSP** È un linguaggio di markup che permette di generare pagine web dinamiche, formata da parte statica (html) e parte dinamica che permette di gestire codice lato server (script)

**MAVEN**

Maven è un tool java che permette di avere librerie sempre aggiornate e a disposizione senza doverle sempre installare.

Nel pom.xml vengono inserite le dependency che vanno a prendere le librerie da dei repository online.

Maven features: è un sistema per gestire dipendenze, aiuta a costruire multicore project e a mantenere

i progetti consistenti

È PLUGIN ORIENTED

È uno strumento per fare build, compilare, deploy e test.

Permette inoltre di fare management e gestire le versioni.

Si utilizza perché la CONTINUOUS INTEGRATION permette di integrare lo sviluppo di nuove funzionalità

Il project name (GAV) groupId identificatore univoco

artifactId nome arbitrario del progetto

version

**JAVA ENTERPRISE e ARCHITETTURE THREE-TIER**

CLIENT TIER: componenti sulla macchina client

WEB-TIER: componenti sul JEE server

BUSINESS-TIER: Componenti sul JEE server

Quando è consigliato usare enterprise bean? Quando l’applicazione deve essere scalabile, l’app avrà più client e le transazioni devono assicurare data integrity.

ENTERPRISE = per l’impresa

**CARATTERISTICHE DI J2EE Java connectors** API che consentono di creare dei wrapper per facilitare la

comunicazione con sistemi legacy

**Java messaging service** API per la gestione di messaggi asincroni e

publish/subscribe

**Java transaction service** API che supportano la gestione di transazioni

(CORBA)

**ENTERPRISE JAVA BEAN EJB** Modello componenti lato server per gestire

sessione e logica di business, offrono

servizi alla parte frontend

Singleton

JDBC

Application client: programmi java che interagiscono con la business logic del server per offrire servizi

Session beans: vengono usati per gestire la sessione utente

**Persistenza @Entity**

Deve avere un costruttore senza parametri

L’entità è un oggetto persistente nel dominio

Le variabili di istanza presenti devono essere private, protected o package protected

**SPA**

I siti web tradizionali hanno svariate pagine web

Il single page application permette di avere solo una pagina html velocizzando i tempi di risposta

I render in javascript sono solo di alcune parti della pagina

**CMS e GIT**

Configuration Management System

Scenario: C’è un bug e vuoi tornare indietro di versione in versione per capire dove è stato inserito e risolvere

Se si fa parte si un grosso team che lavora sullo stesso progetto è rischioso fare delle sovrascritture, voglio isolare il lavoro e fare il merge solo quando sono sicuro

CMS Roles **Configuration manager** È responsabile di identificare le varie release, gestisce il gruppo

**Change control board member** È il responsabile di approvare o meno le change request

**Developer** propone le change request

**Auditor**

Il change management è il modo in cui vengono gestite le richieste di change request

Controlling change **PROMOTION** È uno stato interno che dice che c’è una modifica

**RELEASE** È committata

Record keeping: possibilità di tornare alle versioni precedenti

Working copy, revisione, repository ecc ecc

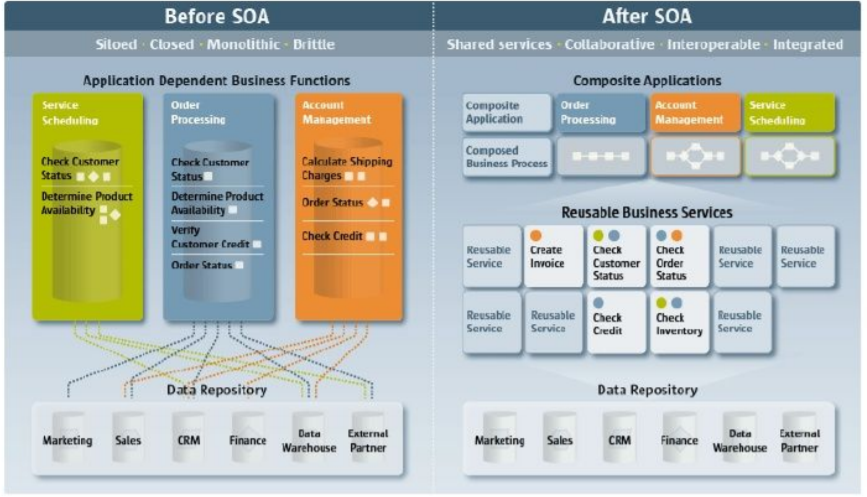
**Service Oriented Architecture SOA**

I web services sono componenti software con interfacce scritte in CML con sintassi WSDL

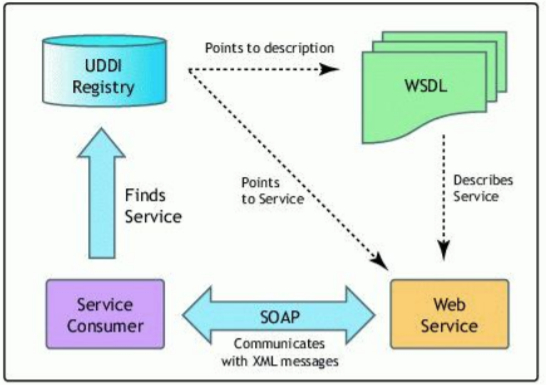
Ci sono molti servizi indipendenti e riusabili  
I vari servizi comunicano tra di loro a prescindere dall’utente

Un web service è accessibile in rete e indipendente dall’utente

LOOSLEY COOPLED: i servizi sono indipendenti tra di loro  
ENCAPSULATED: la logica è nascosta all’utente, che vede solo l’interfaccia



La forza dei web service è che hanno XML come linguaggio di interazione tra servizi

**WEB SERVICES SOAP (Antecedenti ai REST)**

**Simple Object Association Protocol**

**SOAP** è il protocollo per richiamare procedure remote

**UDDI Registry** è il registro in cui sono rubricati i

Servizi e le descrizioni scritte in

**WSDL** ovvero il linguaggio di descrizione

**TCP/IP, http, XML (Soap,wsdl,uddi)**

* **Le comunicazioni SOAP chiamano direttamente le funzioni in modo specifico usando XML**

Da ricordare: dall’architettura a silos ad avere vari servizi che comunicano tramite SOAP o rest

**WEB SERVICE REST**

E’ uno stile di architettura software per sistemi distrubuiti

* **AL POSTO DI FARE LE INVOCAZIONI CON SOAP USANO LE PRIMITIVE HTTP PASSANDO JSON**

E’ più semplice ma bisogna far entrare tutto nell’URI

SONO STATELESS quindi bisogna gestire la sessione in altro modo

**ARCHITETTURA A MICROSERVIZI**

L’architettura a microservizi è legata ai servizi REST e basata sulla Continuous integration (CMS e GIT)

Un servizio offre gli api per gestire appunto un servizio, è una parte della logica applicativa più piccolo e indipendente.

Ogni servizio usa il suo Database e può essere scritto in linguaggi diversi.

**Benefici** Sono più piccoli e più gestibili

Possibilità di scelta di diverse tecnologie

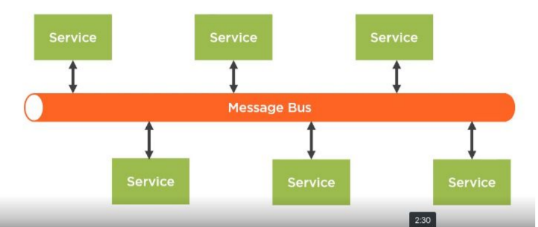
Scalabilità

Agility

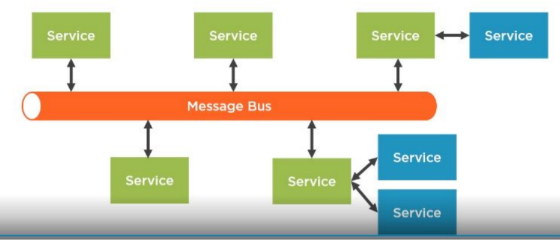
DUPLICAZIONE DELLE TABELLE NEL DB -> MANTENERE LA CONSISTENZA

API GATEWAY (con load balancing e sicurezza in un solo servizio) //ZUUL

**Event driven architecture**



**Hybrid: event driven + RPC**

****

**RABBIT MQ** message broker tra servizi

**EUREKA** E’ il registration service permette di non scrivere l’indirizzo direttamente nel client ma averli insieme per poi usare il gateway (zuul)  
Service Discovery

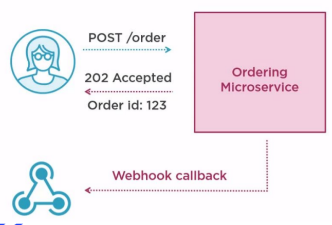
Duplicazione DB -> mantenere la consistenza

**Differenza tra SOA e MICROSERVICES**

**SOA** stateful **MICROSERVIZI** stateless

**SOA** usa enterprise javabean **MICROSERVIZI** usano scambio msg (+ semplice)

**SOA** grossi  **MICROSERVICE** piccoli

Comunicazione ASINCRONA (rabbit)

**MICROSERVICE DESIGN PATTERN**

**Greenfield:** Sviluppo da 0

**Brownfield:** Sviluppo su una app già esistente (slide)

**Domain Driven Design**

È ciò che si fa quando si definiscono le classi

Domain

Context

Service

**Microservizi sincroni: comunicano direttamente tramite rest**

**Microservizi asincroni: Comunicano tramite msg broker, che poi andrà a comunicare con il servizio richiesto**

**Design pattern dei mircoservizi**

FACADE è una interfaccia che sta tra la business logic e le API

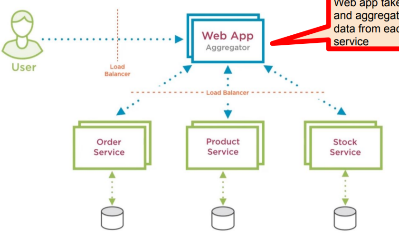
PROXY design pattern c’è un proxy tra business logic e API

Stateless Service pattern è normale, perché i rest sono stateless

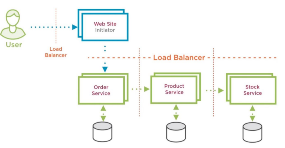
**Composition pattern**

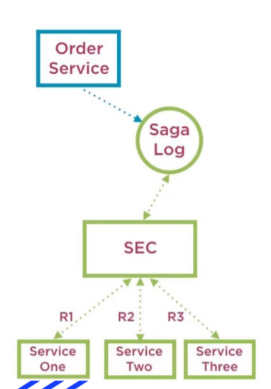
**Broker composition pattern** quello con il msg broker

**Aggregate composition pattern**

****

**Chained composition pattern**

****

**SAGA PATTERN**

Si basa sul SAGA log, il log mantiene traccia di tutte le info e gli scambi tra

Microservizi, Il SEC legge il log, e manda delle richieste di compensazione

Se ci sono errori allora attiva il ROLLBACK

**DOCKER**

Prima una app sta su un unico server, ogni microservizio aveva la sua Vm e il suo OS

Successivamente sui server si è installato un unico SO e si usano i container sopra questo SO.

I container sono una astrazione poggiata sopra il SO e permette di avere dei servizi all’interno

Sono aree isolate di un sistema operativo e permettono di avere l’app come sistema isolato

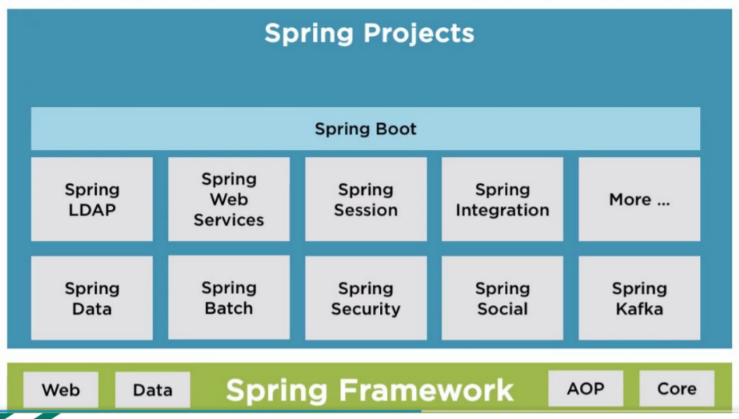
Docker machine è lo strumento che permette di installare docker

Docker compose da tutti i comandi docker

Volume

**SPRING**

E’ un framework di librerie



Svantaggi : too much magic

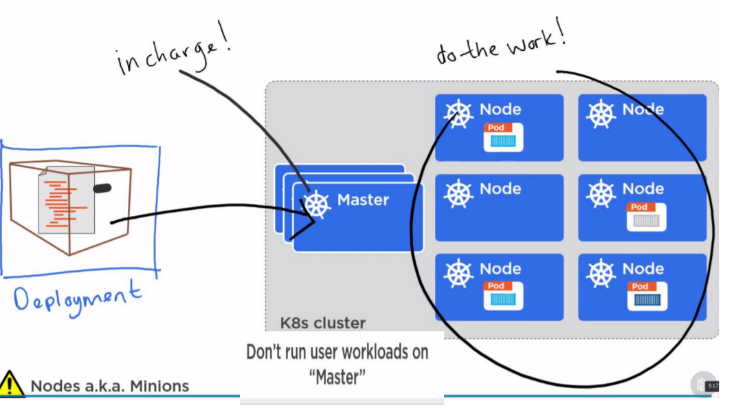
Ha le JPA che sono molto piu semplici del JDBC

**KUBERNETES**

Kubernetes è un orchestratore di containers (es se uno cade lo ritira su, o ne fa runnare un tot sempre, o schedulare i containers)

Serve ad orchestrare i microservizi

Kubernetes prende delle prescrizioni (le istruzioni di cosa deve fare) tramite un file prescriptions.yalm

Il master legge le istruzioni e gestisce

I nodi contengono uno o poi PODS

Cluster store: la memoria

Kube controller manager: gestisce i nodi (manager)

Kubes cheduler

**CLOUD COMPUTING (Heroku)**

Permette di avere potenza di calcolo o altro su server remoti  
Puo essere PRIVATO , PUBBLICO, IBRIDO

SaaS si usa del software sul proprio device con solo la UI, tutta la logica è su altri server (Google docs)

PaaS (Platform as a service) Viene fornito l’ambiente software su cui deployare proprie applicazioni (Heroku), ho gia la VM pronta

IaaS viene fornita direttamente l’infrastruttura remota su cui poter operare (potenza di calcolo, memoria)