SOA:

È un modo di Programmare, Veicolare attraverso HTTP la Chiamata a funzioni Remote (servizi) Noti come Web Services

Usare SOAP (xml)

Metodologia Rest uso 4 Metodi HTTP (+ c'è codifica JSON)

JSON ⇒ Nato per SERIALIZZARE i dati

↳ Quando devo Trasferire dei DATI dal chiamante al Chiamato attraverso la Rete

↳ Lo STUB Serializza e lo SKELETON deserializza

↳ Nel Caso ci fosse un RETURN Allora SKELETON Serializza e STUB DeSerializza

↳ Non ci si Accorge Quasi di Nulla se non che ci Potrebbero Essere RITARDI

STUB Stessa Interfaccia del Oggetto Chiamato

↳ fa da PROXY.

Json è formato Testuale Quindi 1 Byte per Carattere

Si Spreca per le Parentesi ma è per Semplificare il PARSING

Sul BACKEND posso Avere più funzioni, una Sorta di libreria Back-End.

SOA e MICROSERVIZI:

funzione Pesante in Back-End:

- Occupa CPU
- fa aspettare il BACK-END

funzione è un UNITÀ ATOMICA, se Nessuno mi Intercompe eseguo Quella dall' Inizio Alla fine

Si fanno funzioni in BACKEND in DATACENTER e finirà in un BLADE all' Interno di un RACK.
Macchine Sono VIRTUALIZZATE dove viene Eseguita Quella funzione

È Così INEFFICIENTE, si Vuole avere un MULTITASKING, Voglio che la mia funzione venga SPEZZETTATA il più possibile

BACKEND MONOLITICO → BACKEND MICROSERVIZI

(1 cuoco per Cliente) (5 cuochi che fanno tutto)

Applicazione con una Mega funzione

Applicazione con Tanti Micro Servizi che Collaborano

Posso:

1. Chiamare la funzione più volte con Parametri diversi
2. Costruire proprio la funzione in Maniera diversa

ARCHITETTURA MICROSERVIZI:

VANTAGGI A RUNTIME:

- Facilità il LOAD BALANCING (distribuzione del lavoro)
- Scalabilità (Si può adattare in BASE alla RICHIESTA) per Allocazione dinamica delle RISORSE

VANTAGGI A DEV:

- Più facili da Sviluppare

Perdo un po di Tempo Nel Passare da un MODULO all' Altro.

Nei DB Relazionali è Bene che le Tabelle siano Sullo Stesso Database perchè Altrimenti NON posso fare le JOIN, o meglio NON posso Sfruttare Gli Algoritmi OTTIMIZZATI per farlo.

Ma Se NON Spacco il DB NON è Microservizi.

Devo SPACCARE il DataBase in Maniera furba per Evitare di dover fare CHIAMATE COSTOSE.

Esempio ECommerce: Prodotto e descript. Prodotto saranno collegati e NON Voglio Separarli

Mai Quindi SPEZZARE le Tabelle che uso Nei JOIN su 2 Istanze di POSTGRES ≠ perchè dovrei fare io a mano la JOIN (CHE È MOLTO MENO EFFICIENTE RISPETTO AL DBMS)

Ci devono ESSERE Moduli che però NON Comunicano Troppo con gli Altri, se no dovrei fare il MERGE per Migliorare l'Efficienza

Architettura orientata ai Servizi mi Permette di Mescolare i linguaggi (cosa che con i MONOLITICI era difficile)

Stare però ATTENTO ai TIPI Base Nativi che potrebbero Essere diversi.

Back-End:

- Progettarlo
- Implementarlo
- Istanziarlo
- Ottimizzare il Load Balancing

LOAD BALANCING:

Devo Evitare il Sovraccarico di 1 macchine e Le Altre che NON fanno Nulla.

Chiaramente Processo Server in Ascolto consuma Comunque Energia (ANCHE SE MAGARI POCA)

Vorrei Che il Carico Si Spalmasse UNIFORMEMENTE (per Questo devo Avece dei MICROSERVIZI) altrimenti ho una Granularità Troppo Grande

VIRTUALIZZAZIONE ⇒ Mi distacca Macchina dalla fisica, od è utile Quando ci Sono differenze (come IL. S.O.) ma anche Architettura

Macchina Virtuale farà da TRADUTTORE... Tutto ciò Costa Molto e nel DATACENTER non ne ho Bisogno di Questo
↳ Voglio INCAPSULARE la mia App.

CONTAINERIZZAZIONE ⇒ Virtualizzazione più leggera, Voglio DUPLICARE il S.O.

Quindi Magari Riuscire ad avere da 1 linux hardware a 5 S.O. Linux.

Mi Serve un modo per Isolare alcuni file/Librerie sulla Stessa Macchina.

Creo un CONTAINER per Ogni Configurazione che mi Serve, come ad Esempio il fatto di Avere 2 Istanze di POSTGRES con Versioni diversa

CONTAINER ho Tutto DUPLICATO (ambiente) e Quindi OCCUPO MOLTO SPAZIO

KUBERNETES Sarebbe l'ORCHESTRATORE dei CONTAINER

da Pensare come un SUPER PROCESSO

↳ He sì il PROCESSO ma anche tutto l'Ambiente

Come si fa:

- Servizi ≠ Vanno su Istanze ≠ (un Po FAKE)
- Server Smista e Basta Le RICHIESTE (server con IP unico)
- Sfrutto DNS, associo al Nome logico più Indirizzi IP, c'è però il Problema che i DNS hanno la CACHE e Quindi NON funziona Così Bene

Applicazione deve Essere SCRITTA in maniera STATELESS, Quindi un MICROSERVIZIO non deve Avere BISOGNO dei RISULTATI dei Servizi Invocati Prima altrimenti devo far Girare tutti i Dati e Quindi potrebbe Essere che Aspetto che finisca di Produrli

Quindi NON dovrei Avere: dipendenza dei dati
Problemi di Precedenza

Cloud Computing novità Economica (NON TECNOLOGICA), dove il POSSESSORE dell' HW NON è il POSSESSORE dell' Applicazione