Contents

Container - Kubernetes

- **Affidabilità** ed **isolamento** data dalla virtualizzazione di una macchina.
- **Container Orchestrator**: K8 è un **orchestratore**, cerca quindi di mantenere uno stato spegnendo o accendendo specifici container.
- **Kublet**: Processo del nodo worker che permette la comunicazione con l'API di K8's Cluster Services.
- **Desired State**: File di deployment (file yaml) che descrive lo stato desiderato, solitamente in numero e caratteristiche dei pod.
- **Proprietà Garantite da Kubernetes**:
  - **Distribution**: Viene fornita un interfaccia che permette l'interazione con tante macchine diverse.
  - **Declarativeness**: Si dichiara nel file di deployment uno stato desiderato ideale, in modo tale da poterlo fixare nel caso in cui lo stato corrente non fosse coerente con quello dichiarato come desiderato.
  - **Decoupling**: K8s permette lo sviluppo di container indipendenti, ciascuno con il proprio scopo, gestendo così un architettura a microservizi.
  - **Immutable Infrastructure**: L'aggiornamento dell'architettura deve seguire lo schema utilizzato durante il primo deploy, non si va mai ad esempio ad aggiornare il singolo container, si modifica infatti lo stato desiderato per quel container, e si sostituisce il vecchio con il nuovo deployato.
- **Oggetti di K8s**:
  - **Pod**: Unità di K8s, contiene:
    - Uno o pochi più container correlati.
    - Volumi relativi a questi container.
    - Un layer dedicato all'interfaccia di rete.
  - **Deployment**: Anche lo stato desiderato è oggettificato, layer creato dagli statement del file yaml.
  - **Service**: Ciascun Pod può andare giù ed essere sostituito da uno identico, di conseguenza risulta necessario utilizzare un ulteriore livello di indirizzamento ip che wrappi i Pod in modo tale da indirizzare anche in caso di sostituizione di Pod. Solitamente **wrappa** quindi l'**oggetto** di **Deployment**.
  - **Ingress**: Oggetto che ci permette di **esporre** degli **oggetti** di **tipo Service** anche all'**esterno**. Possiamo scegliere anche di esporne solo alcuni e non tutti.
- **Descrizione dell'Architettura di K8s**:
  - **Master Node**: Nodo di gestione, solitamente singolo, è composto da:
    - **API Server**: Accetta richieste su aggiornamenti di stato del cluster, richiede periodicamente lo stato del cluster.
    - **Controller Manager**: Confronta stato desiderato e stato corrente.
    - **Scheduler**: Si occupa dell'effettiva assegnazione degli oggetti in

caso di cambio di stato.
  - **Worker Node**: Nodo effettivo, è composto da:
    * **Kubelet**: Processo che permette la comunicazione con l'API Server del Master Node, aggiorna il master sullo stato del node a cui appartiene.
    * **Kube Proxy**: Permette la comunicazione tra Worker Nodes allo stesso livello.

Cloud Native - Architettura a Micro Servizi

- **REST**: Architettura basata sul concetto di transizione di stato, la quale sarà trasferita tramite rappresentazione all'utente. Si basa su questi principi:
  - Ogni **risorsa** è **identificata** tramite **URI**.
  - **Interfaccia uniforme**, definite delle **azioni standard** (GET, POST, DELETE, PUT) con specifici protocolli.
  - Messaggi auto descrittivi.
  - Ogni **interazione con una risorsa** risulta essere **stateless**, lo stato è handlato solo lato client, non lato server.
- **OpenAPI**: Descrizione d'Interfaccia per HTTP standard, disinteressata dalle specificità di linguaggio.
- **Vantaggi delle Architetture a Micro Servizi**:
  - **Lead time più corto** (da quando parte lo sviluppo di una feature fino a quando non arriva all'utente).
  - **Scaling più semplice**.
- **Caratteristiche delle Architetture a Micro Servizi**:
  - **Sviluppo di insiemi di servizi stateless**, ciascuno runnato su un container proprio, aventi la possibilità di comunicare con meccanismi leggeri ed indipendenti dal linguaggio di ognuno.
  - Organizzazione più dinamica in team per set di microservizi.
    * **Coupling**: Dipendenza di un componente dagli altri.
    * **Cohesion**: Correlazione degli elementi di un componente tra loro.
    * **Equilibrio**: Non devono essere ne troppi ne pochi, un microservizio dovrebbe essere gestito da massimo 8-10 persone.
  - **Database Decentralizzato**: Ogni servizio gestisce il proprio database, facendo affidamento alla **eventual consistency**. Netflix ad esempio utilizza **Cassandra** di **Apache** per la consistenza dei dati distribuiti, si deve raggiungere almeno un quorum per dichiarare effettuata con successo una transizione.
  - **Servizi Modulari**: Più semplice deployare in prod delle singole feature, senza dover redeployare tutte le altre.
  - **Scalabilità Orizzontale**: Sarà possibile scalare solo i servizi che lo richiedono e non tutta l'applicazione.
  - **Resilienza ai Fallimenti**: Una gestione modulare causa anche

maggiore resistenza. **Chaos Monkey** di **AWS** è un servizio che viene utilizzato per testare la resilienza di un applicazione a microservizi buttandone giù i container in maniera casuale.
- **DevOps**: Gestione dello sviluppatore dell'ambiente d'esecuzione del proprio prodotto.

Docker Swarm Lab - Un Paio di Comandi Comuni

- docker **swarm init** ci permette di avviare uno swarm (**un nodo Manager**), questo esporrà ip, porta e token per le connessioni ai worker
- docker **join** [TOKEN] [IP:PORTA] ci permette di **connettere** nuovi **nodi worker** allo **swarm** del manager creato prima
- docker **service update –replicas 3** sleep-app ci permette di **replicare 3 volte il servizio passato come secondo argomento**, in questo caso "sleep-app"
- docker **node update –availability drain** NODE_ID ci permette di **svuotare** il **nodo** contrassegnato da NODE_ID **dai suoi container**
- **docker stack services** [STACK] permette di leggere tutti i servizi presenti su uno stack
- **docker service ps** [SERVIZIO] permette di visualizzare tutti i task di un servizio

Data Centers

Un paio di punti dalla lettura del file sui data center: - Nascevano negli anni 40 per scopi militari - Dagli anni 2000 è nato lo schema enterprise, dove delle compagnie offrono dei servizi di computazione occupandosi pienamente della gestione dei dati. - CSP (Cloud Services Provider) offrono hyperscale data centers, come AWS, Google, IBM Cloud. - Contengono migliaia di server. - Edge Data Centers, messi più vicini agli utenti per utilizzi reattivi più specifici. - Calcolo PUE (Power Usage Effectiveness): Quanta energia viene utilizzata oltre a quella necessaria per l'alimentazione dei calcolatori.

Business Models

- **Valore Generato**:
  - **Value Propositions**: Descrive l'insieme di prodotti e servizi che creano valore per uno specifico segmento di utenza.
- **Consegna**:
  - **Customer Segments**: Descrive il target da servire.
  - **Customer Relationships**: Tipi di relazioni instaurate tra azienda e specifico insieme d'utenza.
  - **Channels**: Canali di comunicazione tra azienda ed utenza.
- **Creazione**:

    - **Key Resources**: Asset necessario al funzionamento del business model.
    - **Key Activities**: Attività che l'azienda deve effettuare per rendere funzionante il business model.
    - **Key Partners**: Descrive il network di fornitori alle spalle del business model.
- **Costi/Guadagni**:
    - **Revenue Streams**: Flusso di entrata che produce uno specifico insieme d'utenza.
    - **Cost Structure**: Costi causati dal business model.

Modelli Noti

- **Freemium**:
    - Vengono convertiti circa il 10% degli utenti
- **Epicentri di Business Innovation**:
    - **Resource Driven**: Ad esempio AWS, l'azienda già aveva delle risorse ed ha creato un business model su esse.
    - **Offer Driven**: Offerta che batte quelle già esistenti sul mercato.
    - **Customer Driven**: Modelli di business basati sulle necessità dell'utenza.
    - **Finance Driven**: Modelli di business nati dalla presenza di nuovi flussi d'entrata o costi ridotti.
- **Caso di Studio - Amazon**:
    - Nasceva come Cadabra, vendita di libri online, era stato considerato di andare in perdita per i primi 4/5 anni.
    - 574 BUSD nel 2024, provenienti da:
        * 39% vendite online
        * 25% da Fullfillment
        * 17% da AWS
        * 9% da Pubblicità

Cloud-Edge Continuum

Introduciamo dei modelli noti: - **IoT/Edge**: I dati IoT vengono elaborati in locale, sul posto, questo porta a basse latenze ma anche ad un basso potenziale di calcolo. - **IoT/Cloud**: I dati IoT vengono elaborati su cloud, questo porta ad alte latenze ma anche ad un alto potenziale di calcolo. - **Cloud/Edge Continuum**: Si gestisce il cloud a gerarchia, in modo tale da rendere il più vicino possibile l'IoT al Cloud. Questo schema è definito anche **Fog Computing**. Questo genere di schema si basa su sistemi a microservizi e containerizzati.

Proprietà del Cloud Edge

- **Requirements**:

  - **Data Gravity**: I dati possono essere molti e difficili da spostare.
  - **Security**: Dipende fortemente dall'infrastruttura.
  - **Trust**: Gestione della fiducia tra nodi, in modo che sia anche transitiva.
  - **Sustainability**: Bisogna tener conto anche dell'impatto ambientale causato da queste architetture.
- **Infrastracture**:
  - **Heterogeneous**
  - **Large**
  - **Dynamics**

Approcci al Cloud Edge

- **Monitoring**: Per garantire le proprietà sopra elencate senza l'esistenza di un algoritmo per il piazzamento risulta necessario monitorare applicazioni ed infrastrutture. Esistono vari tool che permettono questo tipo di analisi dinamica, come ad esempio FogBrainX o FogArm.

Quantum Computing 101

- **Stati del Qbit**: **Sovrapposizione** ed **Entaglement**.
  - **Sovrapposizione**: Combinazione lineare degli stati ideali 0 ed 1.
  - **Entanglement**: Due qbit possono avere stati correlati a prescindere dalla distanza che li separa.
- Una Classe di problemi attualmente non risolvibili in tempo polinomiale risulta esserlo in campo quantistico.
- Diverse categorie di tecnologie per la rappresentazione del qbit, ciascuno ha le proprie caratteristiche.
- **Era NISQ**: Attualmente è detta Era (Noisy Intermediate-Scale Quantum), data l'alta sensibilità degli errori e dei pochi qbit disponibili sulle macchine attuali (100 - 1000 qbits).
- **Cloud e Shots**: Provider cloud offrono l'esecuzione di shots (singole esecuzioni di circuiti).
- **Obiettivi e Mancanze**: Manca ancora tutta l'astrazione creata dai linguaggi, la qualità ed il riuso del software, la gestione del ciclo di vita dei progetti basati sul quantum. Attualmente si sta cercando di integrare tool di testing e debugging.

Approcci Innovativi

- **Shot Wise Distibution**: Un broker gestisce la distribuzione degli shots su più QPU, in modo tale da ottenere risultati più resilienti e personalizzati.
- **Circuit Cutting**: Tecnica di suddivisione di un circuito in sotto circuiti. Questo può portare a vantaggi di riduzione della profondità e del rumore dei

frammenti, ma allo stesso tempo può causare tempi di pre e post processing esponenziali nella dimensione del circuito e del numero dei frammenti.

- **Cut&Shoot**: Combina le sue tecniche, seguendo queste fasi:
  - Taglia il circuito in frammenti.
  - Distribuisce i frammenti su diverse QPU.
  - Aggrega i risultati parziali.
  - Costruisce la distribuzione del circuito originale.