**DOMANDE CLOUD COMPUTING**

**Vallati:**

**#### Multi tenancy**

* Spiega
* Vantaggi

Risposta:

La multi-tenancy risolve le limitazioni introdotte dal modello multi-user nel quale più utenti condividono la stessa macchina fisica e ne sono consapevoli; infatti, in questo sistema c’è un amministratore che ha privilegi maggiori rispetto ai tenant che hanno un account limitato.

Nella multi-tenancy, ogni tenant opera con una porzione virtualmente separata dell’applicazione, ma condivide l’hardware con altri tenant sullo stesso server o cluster di server. In altre parole, un tenant vede l’intero sistema come se fosse interamente suo.

Tra i vantaggi abbiamo:

**Condivisione delle risorse**: Le risorse hardware e software sono condivise tra più tenant, aumentando l'efficienza e riducendo i costi operativi.

**Isolamento e sicurezza**: Anche se i tenant condividono le stesse risorse fisiche, i loro dati e configurazioni sono mantenuti separati.

**Scalabilità e gestione centralizzata**: Le applicazioni multi-tenant possono scalare facilmente per supportare un numero crescente di tenant senza richiedere modifiche significative all'infrastruttura.

**Personalizzazione limitata**: Sebbene i tenant condividano la stessa applicazione, possono avere alcune opzioni di configurazione personalizzabili. Tuttavia, queste personalizzazioni sono generalmente limitate rispetto a un'architettura a tenant singolo (svantaggio).

**Economia di scala**: La condivisione delle risorse e la gestione centralizzata riducono i costi operativi e di manutenzione.

**### Principali vantaggi di adottare cloud computing**

* principali vantaggi di adottare cloud computing (da punto di vista startup e grande azienda e aziende che hanno bisogno di potenza computazionale con spike e periodi morti)

Risposta:

**Minore responsabilità gestionale**: Le macchine sono dal cloud provider, quindi è sua responsabilità gestirle, ripararle, aggiornare, potenziare etc.

**Minore responsabilità legale**: riguardo la gestione dei dati dei clienti, poiché “*scarico*” la responsabilità sul fornitore dell’infrastruttura cloud.

**Qualità del servizio più elevata**: Il cloud provider ha il compito di creare e mantenere una grande e costosa infrastruttura; quindi, la qualità dell'infrastruttura (*hardware e software*) e la qualità delle persone che ci lavorano sono una priorità primaria.

**Affidabilità maggiore**: I cloud provider hanno capacità gestionale infinitamente superiore rispetto ad un server “*in casa*”. Il sistema continuerà a funzionare correttamente anche in caso di guasti, disastri o altro.

**Disponibilità maggiore**: I cloud provider sono attivi 24x7, quindi un servizio può rimanere attivo sempre.

**Gestione software semplificata:** L'acquisto di licenze per software complessi (*e costosi*) è obbligatorio nella creazione di un’infrastruttura cloud. Usare un servizio cloud ti permette di non comprare queste licenze perché fornite in primo luogo dal cloud provider.

**Indipendente dalla locazione:** Non mi importa dove siano localizzati i server, molto probabilmente non saprò mai dove sia il data center.

**#### Cloudification**

* Vantaggi e svantaggi

Risposta:

Vantaggi sono quelli elencati nella risposta precedente.

Svantaggi

**Connessione ad Internet obbligatoria:** Un'azienda con servizi in cloud deve sempre essere connessa; quindi, si aggiunge un costo fisso dovuto alla connessione ad Internet (*non è un grande contro, la connessione ad Internet l'avrebbe avuta comunque, però con una tariffa meno costosa*).

* 1. **Richiesta di grande banda:** Serve un servizio di connettività potente in grado di trasportare un grande volume di dati. È possibile mitigare questo problema, tramite una strategia molto particolare: Se voglio spostare petabyte verso il mio Cloud provider, mi conviene anzi consegnare fisicamente un Hard Drive che contiene tutto.
  2. **Portabilità limitata:** Il processo di standardizzazione delle tecnologie cloud è ancora indietro. Diversi cloud provider potrebbero (*molto spesso*) usare tecnologie molto diverse e spesso proprietarie e incompatibili tra loro; ciò rende difficoltoso il cambiare cloud provider dopo averne scelto uno, questo problema è detto “*lock-in*”.
  3. **Problemi legali:** I data center vengono costruiti in posti convenienti (*elettricità poco costosa, lavoro poco costoso, etc*), ma ogni nazione ha le sue normative che possono essere incompatibili con le normative del paese in cui voglio fornire il mio servizio, specialmente quando si parla di dati sensibili. Ad esempio, le cartelle mediche devono essere memorizzate nella stessa nazione del paziente.
  4. **Sicurezza:** Una delle ragioni primarie per cui un'azienda non vorrebbe eseguire una cloudification. L’azienda vorrebbe tutto sotto il suo controllo, quali sono i rischi nel trasportare i dati verso il cloud? Il canale di trasmissione è insicuro: I dati possono essere intercettati; quindi, ho bisogno di meccanismi di crittografia per proteggere la confidenzialità dei dati trasmessi.

**#### Modello NIST**

* tipi di servizio che un cloud provider puo fornire ai clienti
* differenza public cloud e private cloud (e altri tipi)
* Supponiamo che tu sia il progettista di una nuova app per smartphone dove utenti inseriscono ricette di cucina e gli altri le ritrovano. Supponiamo invece che tu sia il progettista di una multinazionale che opera nel campo biomedico perché produce dispositivi biomedici per il monitoraggio dei pazienti. Quale tipo di infrastruttura e servizio suggeriresti nei due casi?

Risposta: per quanto riguarda una startup come infrastruttura andrei su un cloud public che risulta più economico, accessibile da rete pubblica da molti utenti contemporaneamente. Per quanto riguarda il servizio userei il PaaS, in modo tale che la startup si concentri unicamente sulla logica della applicazione senza dover configurare le macchine virtuali e installare altri software come nell’IaaS, o ancora meglio SaaS, cosicche debbano solamente utilizzare la applicazione. Invece la multinazionale potrebbe andare su un private cloud avendo maggiore disponibilità economica e dovendo probabilmente gestire i dati internamente nella propria azienda. E poi su IaaS, in modo tale da avere la massima personalizzazione possibile sulle macchine virtuali e permettergli di installare i software necessari allo sviluppo di servizi.

Il modello NIST spiega i tre pilastri fondamentali di una architettura cloud:

1. Le **caratteristiche fondamentali** di una architettura cloud.
2. **Essere sempre accessibile** via rete internet.
3. **Avere una alta elasticità** riguardo l’allocazione e de allocazione delle risorse.
4. **Avere risorse misurabili:** L’utilizzo delle risorse deve essere misurato.
5. **Allocazione di risorse On-Demand:** Le risorse devono essere allocate on-demand, quindi subito dopo la richiesta.
6. **Gestione automatizzata delle risorse:** Le risorse computazionali dell’infrastruttura cloud devono essere tante, e gestite in modo remoto in modo da essere in grado di soddisfare tutti i bisogni dei clienti contemporaneamente.
7. Le tre **tipologie di servizio** che può offrire.

* IaaS: questo servizio consiste nel fornire delle macchine virtuali “*vergini*” al cliente, il quale potrà configurarle da remoto secondo i suoi bisogni come una normalissima macchina. Inoltre, il cliente deve configurare la sua macchina virtuale installando (*sopra il sistema operativo scelto*) i software necessari per creare il suo servizio. Il cloud manager invece ha il compito di installare l’hypervisor (*VMM, nell’immagine sotto è il “Layer of Virtualization”*) sulle macchine fisiche, gestire le macchine virtuali e ovviamente gestire l’hardware su cui girerà la macchina virtuale.
* PaaS: non offre l’accesso alle machine virtuali ma a piattaforma di sviluppo installate sopra le macchine virtuali.

Su questa piattaforma gli sviluppatori software possono creare le loro applicazioni usando le API fornite dalla piattaforma stessa.

Il cloud manager si occupa di installare il sistema operativo sulle macchine virtuali e di gestire il runtime.

Il customer si occupa solo di scrivere la logica dell’applicazione. Ha il problema del vendor lock-in.

* SaaS: il cloud provider gestisce tutto, logica dell’applicazione e dati. I clienti cosa fanno? Usano le applicazioni fornite (*ovviamente con pochissima possibilità di customizzazione da parte del consumatore del servizio*).
* Tra i modelli sono presenti anche storage as a service, database as a service, backup as a service e desktop as a service.

1. I quattro **modelli di deployment** (*distribuzione*) dell’infrastruttura cloud.

* Public cloud: in questo modello promuove una alta multi-Tenancy ai suoi più alti livelli; infatti, è studiato per essere usato da moltissimi utenti contemporaneamente. È accessibile tramite rete pubblica e le sue risorse sono condivise tra molteplici consumatori.
* Private cloud: La gestione è eseguita da un'unica organizzazione per il suo uso interno. Di solito l’infrastruttura è localizzata all’interno della compagnia (*dentro l’edificio*), infatti questa soluzione è usata quando la compagnia ha dei bisogni davvero molto specifici; soprattutto riguardo la sicurezza per cui richiede il controllo completo sull’infrastruttura.

L’infrastruttura è costruita con tutte le funzionalità di un cloud computing system, ma l’accesso è ristretto solo ai membri dell’organizzazione

Rispetto al cloud pubblico è molto più costoso e impegnativo: l’organizzazione deve pagare tutto di propria tasca per creare e gestire l’infrastruttura.

* Community cloud: un cloud privato ma condiviso tra queste organizzazioni.

I costi fissi di creazione sono divisi tra gli utilizzatori (*quindi minori di un cloud privato*) e i costi variabili di gestione sono stabiliti tramite tariffe pay-per-use.

* Hybrid cloud: I cloud ibridi hanno una parte pubblica (*situato in una infrastruttura pubblica*) e una parte privata.

Ognuna delle due parti può offrire servizi diversi: Ad esempio la parte privata gestirà dati sensibili e quindi attuerà protocolli e tecnologie più indicate a gestire attacchi esterni.

I due cloud ovviamente comunicano tra loro attraverso una certa modalità. Di solito la maggior parte della logica e dai dati è schierata sul cloud pubblico in modo da abbattere i costi; Il cloud privato viene usato solo per i dati molto sensibili.

**#### Hypervisor**

* Cos’è e tipologie
* Quale tipo di hypervisor scegliere se hai 50000 vm tutte uguali?

Risposta: Bare metal approach perché è più prestante ottimizzando meglio le risorse ed essendo più scalabile, avendo accesso direttamente all’HW e non dovendo chiedere risorse HW al sistema operativo guest, si evita overhead inutile, ovviamente è meno compatibile e più complesso da sviluppare.

**#### Virtualizzazione**

* Vantaggi
* Metodologia di virtualizzazione della CPU

Risposta: Ogni processo ha una VCPU, ovvero una struttura dati che rappresenta lo stato del processore quando sta eseguendo quel processo (intende questo?)

* Quali tecnologie usa la CPU per virtualizzare la stessa CPU e RAM?

Risposta: CPU come precedente domanda e RAM assegnando ai processi lo spazio di memoria virtuale, i quali credono di avere accesso all’intera memoria partendo dall’indirizzo 0 (intende questo?)

* Tecniche di virtualizzazione delle varie risorse fisiche? CPU, RAM, I/O, MMU ecc

Risposta: CPU e RAM come nella precedente domanda, I/O attraverso i device model e MMU attraverso virtual MMU

* Page tables on virtualizations

Risposta:

*Virtualizzazione della RAM:*

Ogni VM crede di avere tutta la RAM a disposizione grazie all’intervento del VMM che effettua le traduzioni necessarie e grazie all’utilizzo della memoria virtuale. Nel VMM hosted, una porzione della RAM viene donata al VMM, che la suddivide e distribuisce per le VM in esecuzione. Nel caso bare metal, il VMM ha accesso a tutta la RAM. Il sistema operativo guest utilizza anch’esso il meccanismo della memoria virtuale, dunque si ha un “guest virtual address” e un “guest physical address”, poi quest’ultimo deve essere tradotto in “host physical address”. Per fare questo abbiamo bisogno di modificare la MMU: la funzione G (guest virtual 🡪 guest physical) viene implementata in una page table nella VM, la funzione H (guest physical 🡪 host physical) viene implementata nella VMM. Queste due funzioni fanno parte della cosidetta MMU virtuale, una per ogni VM, che utilizza quella fisica. Questo in generale, nel metodo “brute force” si utilizza la shadow page table, la quale implementa la H nella guest page table in modo trasparente al sistema operativo guest, che crede di aver solo applicato G. In caso di normale traduzione il VMM non interviene. Se non fosse una normale traduzione perche il sistema operativo guest ha modificato la page table di G allora il VMM deve modificare la shadow page table, altrimenti se il SO guest ha cambiato tabella della pagine, il VMM deve creare una nuova shadow page table. La traduzione in ogni caso viene effettuata in HW dalla MMU.

*Virtualizzazione dei dispositivi I/O:*

Per dare l’illusione che le VM possano accedere ai dispositivi di IO, il vero HW viene emulato creando delle rappresentazioni virtuali chiamate device models. Il VMM si comporterà da controller, ovvero si occuperà di tradurre le istruzioni inviate ai device models in quelle per le vere periferiche. Quando la VM accede ai device models si parla di istruzioni privilegiate, dunque interviene il VMM.

Quando arriva una richiesta di interruzione da una periferica, la VM viene interrotta e il controllo passa al VMM (primo cambio di contesto), in questo momento la CPU utilizza la host IDT per gestirla che manderà in esecuzione l’handler indicato (fase di trap & emulate). Quest’ultimo cambierà l’IP della VM inserendo l’indirizzo che punta ad una istruzione “INT”, in questo modo ha iniettato nella VM una interruzione (virtuale) che, quando la VM riprenderà il controllo, triggererà un intervento da parte del kernel del guest OS che provvederà ad eseguire il codice dell’handler presente nella guest IDT. Il comportamento del guest OS risulta uguale ad un normale OS non virtuale.

**#### Full Virtualization**

* How to handle interrupt, schema of the IDT

Risposta: data in precedenza

* KVM e Networking in KVM

Cos'è libvirt o kvm?

Come funziona l'istanziazione di una macchina virtuale utilizzando libvirt/kvm? Che cosa devo fare?

Dal punto di vista della rete come funziona? Le macchine virtuali su quali reti vengono inserite? Come accedono le macchine virtuali alla rete? Come fanno le macchine virtuali create attraverso libvirt kvm a connettersi ad internet? Quali sono le opzioni che abbiamo visto? Passaggi per creare una macchina virtuale con libvirt

Risposta: KVM è un hypervisor mentre libvirt è una libreria software che include virsh che consente di gestire le VM sopra KVM. Per creare una VM ci sono vari metodi, tramite un comando in cui specifichiamo le caratteristiche della macchina, oppure crearla tramite immagini preconfigurate che troviamo in rete. Le VM in virsh sono connesse ad una rete virtuale creata all’interno delle VM, questa rete è creata utilizzando un **Linux bridge**. Per creare una configurazione di rete bisogna scrivere prima la configurazione su un file XML, poi definire la nuova rete partendo da questo file con un comando (virsh net-define nomefile), si imposta l’avvio automatico e si avvia la rete con virsh net-start, si crea una nuova NIC virtuale e si collega alla rete. Infine su configura la NIC nella VM.

* Quando alcune istruzioni non possono essere eseguite in una VM?
* Differenze con la para

Risposta: la paravirtualization ha meno overhead, abbiamo una porzione dei task di gestione della virtualizzazione che è demandata al sistema operativo guest. Si ha una riduzione dei cambi di contesto rispetto alla full grazie alle Hypercalls, funzioni presenti in una API esposta dall’hypervisor verso il sistema operativo guest, richiamandole si ha un solo cambio di contesto verso il VMM (rispetto ai numerosi del modello trap & emulate), per fare ciò il sistema operativo guest è modificato. Tramite le hypercalls il sistema operativo guest accedere direttamente all’HW virtualizzato. Ha meno sicurezza rispetto alla full perche il guest OS ha piu controllo sull’HW ed è meno compabile, solo i sistemi aperti permettono modifiche dell OS. Non richiede HW specifico in entrambi i casi.

**#### Trap and emulate**

* è ancora utilizzato frequentemente?

Risposta: boh si? Però è velocizzata dalla HAV

* Cos’è?

É una tecnica utilizzata nella full virtualization per gestire l’esecuzione di istruzioni privilegiate da parte delle VM. Le istruzioni privilegiate vengono intercettate dal VMM che prende il controllo del flusso di esecuzione e lui emula il comportamento dell’istruzione privilegiata (ovvero emula l’handler) e restituisce il controllo alla VM.

Esempio della INT: viene generata una interruzione dal sistema operativo guest, il VMM prende il controllo ed esegue l’handler indicato nella host IDT emulandolo, il quale manda in esecuzione l’handler della guest IDT nel kernel del sistema operativo guest. Se questo handler prevede altre istruzioni privilegiate viene chiamato nuovamente il VMM ad intervenire. Ci sono numerosi cambi di contesto.

* e come è stata superata?

Risposta: con la root-non root mode

* IDT
* Chi fa il salvataggio del guest operating system?

Risposta: la CPU quando avviene una VMEXIT

* Qual è secondo te il meccanismo più costoso in termini di overhead nel trap and emulate?

Risposta: Emulazione se presente altrimenti cambio di contesto

**#### HW assisted virtualization**

* How memory access is handled, what if we have 64Gb or RAM, but VM only has 2? Is it continuous in host memory?

Risposta: Non è continuo nella memoria host, vedere figura 48 a pagina 68

* HW passthrough

Risposta: per velocizzare l’accesso ai dispositivi viene introdotto questo meccanismo che permette alla VM di accedere al dispositivo senza effettuare cambi di contesto e dunque senza l’intervento del VMM, riducendo l’overhead. Facendo ciò, solo una VM può accedere contemporaneamente al dispositivo. Ovviamente questo metodo non viene implementato per tutte le periferiche ma solo quelle che hanno necessità di alta velocità. Esso viene implementato tramite: mapping diretto dell’IO fisico nella memoria virtuale della VM, cosicche la VM scriva e legga direttamente dai registri della periferica, e all’assegnamento diretto delle interruzioni alla VM, gestita senza cambio di contesto. Il mapping diretto consiste nell’inserimento di una struttura chiamata IO Bitmap nella VMCS. IO Bitmap ha un bit per ogni possibile indirizzo dello spazio di IO e server per regolarne l’accesso (1 significa che è possibile l’accesso senza cambio di contesto. Dunque quando verrà eseguita una operazione di IO verrà controllata questa struttura e può esserci o meno una VMEXIT.

* Vmcs

Risposta: viene introdotta una nuova struttura per gestire le informazioni della modalità non root, in particolare essa contiene le istruzioni che possono essere gestite senza il cambio di contesto verso il VMM. Abbiamo una VMCS per ogni processore di ogni macchina virtuale. Il processore fisico ha un registro apposito per salvare l’indirizzo di questa struttura. In generale la struttura include: guest state (stato corrente della CPU della VM), host state (stato corrente della CPU prima del lancio della VM) e VM Execution control (specifica le istruzioni permesse in modalità non root).

* Dma device

Risposta: è uno di quei dispositivi che può leggere direttamente dalla RAM senza interpellare la CPU, grazie al supporto HW chiamato “controllore DMA”. Per accedere a questi indirizzi fisici IO il dispositivo usa indirizzi virtuali IO che poi verranno tradotti dalla IOMMU, che li traduce per conto del DMA.

* Che meccanismi sono stati introdotti nella virtualizzazione HW?

Risposta: sono stati introdotti dei meccanismi per velocizzare la full virtualization, in particolare sono state introdotte due nuove modalità: root e non root mode. Esse generano 4 stati, combinandosi con livelli di privilegio kernel e user. La root mode è utilizzata quando lavora il VMM e il sistema operativo host per eseguire le istruzioni privilegiate del guest OS ed effettuare le loro operazioni di gestione della virtualizzazione. La non root mode invece per eseguire il codice delle VM. L’obiettivo finale è minimizzare il numero di cambi di contesto ed effettuare un controllo di sicurezza dopo ogni esecuzione di istruzioni virtuali. Per fare queste cose verrà definito un set specifico di istruzioni privilegiate che permetterà alla VM di evitare il cambio di contesto nella loro esecuzione (evita il t&e). La CPU in modalità root permette di avere a disposizione tutte le risorse del sistema, il VMM deve gestirle. Quando invece è in non root non sono necessari cambi di contesto, le macchine virtuali operano come se accedessero direttamente all’ HW.

IRT: La IRT viene introdotta per permettere all’utente di gestire direttamente le interruzioni senza effettuare il cambio di contesto, senza un VMEXIT. Le interruzioni HW (che provengono dai dispositivi fisici per esempio) vengono inoltrate alla VM e con ciò si riduce il numero di cambi di contesto. Ogni entry della IRT ha un puntatore verso la guest IDT di una VM nel caso in cui si voglia far eseguire direttamente l’handler selezionato dalla VM. Altrimenti la entry della IRT punta alla host IDT del VMM per svolgere il normale flusso di esecuzione con il suo intervento. La IRT è gestita direttamente dal VMM.

**#### Nested Virtualization**

**#### Paravirtualization**

* Cos’è e qual è il vantaggio di questa struttura?

Risposta: spiegato prima

* come si ovvia al problema di dover fare trap e emulate?
* Qual è il vantaggio rispetto alla full?

Risposta: spiegato prima

**#### Lightweight virtualization**

* Cosa sono I container?
* Qual è la differenza tra un approccio lightweight e full?

Risposta: sono completamente diverse, la lightweight riduce al minimo l’overhead dovuto alla gestione delle macchine virtuali. La lightweight permette di eseguire le applicazioni in porzioni virtuali indipendenti e isolate ma non in vere e proprie macchine virtuali, esse vengono gestite dallo stesso kernel. Inoltre permette di gestire molti piu container rispetto alle VM.

Inoltre, nell’approccio full il guest OS non è modificato e non è a conoscenza di essere eseguito in un sistema virtualizzato, in questo approccio viene utilizzato il metodo trap & emulate. Nella lightweight virtualization, il guest OS è a conoscenza di essere in esecuzione su un sistema virtualizzato nel caso di paravirtualization

**Docker**

* what is docker Hub
* What is in the Docker file

Risposta: Ci sono le direttive per assemblare (build) le immagini

* Comandi docker

Risposta: creo il docker file indicando con FROM quale immagine ho scelto, uso “docker build” e nome immagine per creare l’immagine docker dal docker file, infine creo ed eseguo il container con “docker run” nome immagine. Posso usare i seguenti comandi sul container: “docker start”, “docker stop” …

* Come funziona l'esposizione di un container verso l'esterno su Docker? Cosa vuol dire esporre un container verso l'esterno e cosa comporta questa operazione?

Risposta: se si vuole esporre un container, nel docker file va inserita la dicitura “EXPOSE 80”. Esporre significa rendere il docker accessibile tramite internet.

**#### Cloud application**

* Secondo te quali sono i costi e i benefici dell’aumentare il numero di tier?

Risposta: migliora la disponibilità e velocità ma richiedono sincronizzazione, manutenzione e costo di installazione e gestione infrastruttura.

* per quale motivo un app non cloud migrata sul cloud non puo sfruttare al massimo i vantaggi di essere sul cloud?

Risposta: perche deve rispettare i principi di interoperabilità, composability ma soprattutto scalabilità orizzontale che è meno costosa di quella verticale, essa permette di aggiungere e togliere componenti senza incidere sulla funzionalità dell’applicazione.

* I vantaggi delle cloud applications con casi d’uso (start up vs. grandi aziende)

Risposta: immagino siano i vantaggi generici di una applicazione cloud (cioè del cloud computing)

**#### SOA**

* perchè offre un packaging dei messaggi standardizzato?

Risposta: O perché ogni servizio è implementato in un modulo che nasconde l’implementazione interna, oppure intende SOAP e negli appunti dice che definisce come il produttore ed il consumatore si scoprono a vicenda. Penso che sia la prima. Immagino sia anche per avere una comunicazione efficiente nella quale si conoscono i precisi formati dei messaggi.

* vantaggi applicazioni basate su scambio messaggi (SOA)

Risposta: implementazione interna nascosta, scalabilità orizzonatle, loose coupling.

* Caratteristiche e tecnologie del backend?

Risposta: Il livello di **backend** invece è quello che effettivamente implementa la logica dell’applicazione; esso riceve le richieste dal modulo di frontend e invia le risposte al modulo di frontend. Non so se intende altro.

* opzioni che sviluppatore ha per implementare comunicazione tra moduli di una cloud application

Risposta: paradigma di comunicazione->produttore e consumatore che garantiscono insieme al broker scalabilità e flessibilità, implementazione di broker che abbiamo visto->RabbitMQ)

* paradigmi di comunicazione verso esterno o verso applicazione (rest o soap)
* Principali differenze tra SOAP e REST (WSDL)

WSDL è un’altra cosa rispetto a REST

* Quale protocollo utilizzare per un’applicazione tipo candy crush, o di un servizio bancario

Risposta: per candy crush è meglio REST perche non prevede una struttura ben definita dunque ha meno overhead ed è piu flessibilie mentre il SOAP ha una struttura ben definita, è piu formale, specifico e resistente ad errori

**#### Compute Cluster**

* Struttura generale architettura (applicazione) cloud? Quali sono le strutture cloud che si adottano in base al tipo di obiettivo che si vuole raggiungere?

Risposta: HA, LB e CI

* LBC: Does a Load Balancing cluster provide availability and redundancy?

Risposta: No, serve per smistare le richieste per garantire scalabilità orizzontale e migliore utilizzazione delle risorse.

* Why do we use High Availability clusters instead, then?

Risposta: Because LB clusters are less efficient to duplicate data, since every node in the tier has to communicate with all the other nodes of the tier.

**#### Modulo di frontend e backend**

* Modo con cui si creano le interfacce verso gli utenti di un interfaccia cloud (Soap, rest e indirect

communication)

* Indirect Communication

Sistema a coda di messaggi

* RabbitMQ**:** Quali sono i modi per scambiare i messaggi? Do you remember what we did in the RabbitMQ Lab?

Risposta: RabbitMQ implementa un message broker utilizzando il protocollo AMQP, facilita la comunicazione tra applicazioni distribuite, in particolare tra produttore e consumatore. Il produttore genera un messaggio, lo invia all’exchange di rabbitMQ, esso lo inoltre ad una delle code secondo una certa politica (attraverso una struttura detta binding avviene questa associazione) poi il messaggio rimane nella coda finche non viene estratto dal consumatore secondo una politica FIFO. Gli exchange possono essere di 3 tipi: direct, fanout, topic. Direct prevede che i messaggi vengono inoltrati nelle code con la stessa routing key del messaggio. Fanout invece prevede l’invio verso tutte le code a cui è collegato e Topic prevede che si estraggano parole dalla routing key per instradare il messaggio. Codice: si creano dockerfile di consumatore e produttore, si lanciano e il produttore invierà un messaggio all’IP del container dell’exchange rabbit con una certa routing key. In quest’ultimo si può modificare il binding tra la routing key e le code e il tipo di exchange scelto.

**#### Replication mechanism**

* Tipi di replicazione in un datacenter (senza network failure)

Risposta: Primary-backup replication, active replication, gossip architecture.

* Strategie nelle cloud applications (active, passive, gossip)
* Architettura gossip**:** Su che assunzione si basa? Cosa che non viene assunta in Bayou?

Risposta:Che non può avvenire partizionamento della rete, mentre Bayou le gestisce? Oppure che il primo è sequential consistency ed il secondo eventual consistency?

* Cos'è la Sequential Consistency?

Risposta: Un client potrebbe ricevere una versione vecchia dei dati, ma comunque consistente. Due client diversi possono ottenere valori differenti per lo stesso dato, però prima o poi avranno l’ultima versione.

**#### Bilanciamento globale del carico**

* How to handle global load balancing
* Come funziona il reindirizzamento delle richieste?

Risposta: Le richieste vengono indirizzate al server più vicino, in caso si voglia ridurre la latenza, è il caso di un motore di ricerca, mentre le richieste vengono inoltrate al server meno sovraccarico nel caso vogliamo aumentare il throughput, è il caso dello streaming.

* Come si realizzano delle applicazioni su scala globale (ad es. motore di ricerca)?

Risposta: boh, global e local load balancers?

* GRE tunneling? dove abbiamo visto la tecnologia gre oltre a qui e come veniva utilizzata?

Risposta: neutron controller neutron agent, permetteva isolamento delle vm

**#### Sistemi geograficamente distribuiti**

* Geographically distributed systems (talking about the problems of network partitons and how to solve them, bayou architectures and operations)
* Bayeu architecture
  + Applicazioni eventually consistent, com’è che viene implementato il sistema (Bayou architecture) e perchè (applicazioni su larga scala, gestione delle partizioni di rete, performance)
  + Eventual Consistency

**#### OpenStack**

* Definizione di cloud platform
* Tipi di nodi
* General introduction
* Network handling and virtualization
* SDN
* Nova component
* Neutron: Come virtualizzare una rete fisica?
* Dashboard (GUI)

Risposta: tramite la gui si possono creare le VM, funzione principale di OpenStack. Con i flavor posso definire le risorse assegnate ad ogni VM, sono creati dagli admin. Nella sezione overview ci sono tutte le risorse allocate. Creato il flavor bisogna aggiungere l’immagine importandola (puo essere messa pubblica o privata). Con questi due si puo creare una VM (Project 🡪compute🡪instances). Si possono creare volumi dall’immagine creata in precedenza che non sono altro che HD virtuali, ospiteranno il SO. Bisogna selezionare anche la rete alla quale collegare la VM. (creabili da Admin 🡪network). Alle VM puo essere assegnato anche un IP pubblico chiamato Floating IP per accedere ad una rete esterna

* Workflow per creare una nuova VM in Openstack
* Tecnologie di virtualizzazione di rete
* Come avviene il processo di installazione di OpenStack? Come si installano le componenti di OpenStack sulle varie macchine una volta aggiunte? Che macchine ci sono e che ruoli hanno quelle che avete installato voi? Dove si esegue "juju" e a cosa serve? Qual è il flusso di lavoro con cui viene utilizzato? Cos'è juju?

Risposta: ci sono due tipi: una con JuJu e un’altra piu onerosa. Nella seconda si installa snapd utilizzato per installare microstack, si esegue il comando per l’inizializzazione, si usa un comando per recuperare la password dell’admin. L’installazione prevede 4 macchine fisiche: 1 per controller, 3 per i compute node ma noi necessitiamo di piu controller per non aver SPOF. In generale si utilizzano procedure di installazione automatizzate, durante l’installazione l’unico host che viene contattato è l’architecture manager. La configurazione scelta è quella con 5 VM: una per l’architecture manager, una per il controller e 3 per il compute node. Invece l’installazione con il framework JuJu è la seguente: esso puo connettersi ad una macchina remota senza usare una password ogni volta, gestisce il development. Le configurazioni avvengono nella macchina remota tramite SSH. JuJu è installato nell’architecture manager, si configura quest’ultimo con il comando “add cloud” . Dopo si aggiungono le macchine con il comando “juju add-machine”. Si effettua la configurazione di rete per i container. Poi utilizzando il comando “juju deploy” avviene il deployment.

**#### DevOps**

* Cos'è il concetto di DevOps? Pro/Contro?

**#### Kubernetes**

* Struttura a grandi linee
* cos’è un pod
* Come le app vengono esposte all’esterno
* Cosa c'è dentro il descrittore di app?
* Explain how autoscaling works in Kubernetes
* Flusso di lavoro in kubernetes, come si instanzia una applicazione nella pratica? Qual è il comando? Ovvero, uno sviluppatore che sviluppa un'applicazione e vuole farla girare su Kubernetes, che deve fare? Come abbiamo installato kubernetes?

Risposta: lo installiamo tramite juju, una volta installato, installiamo Minikube, un tool di test in cui un’applicazione Kubernetes viene eseguita.

Per instanziare un’applicazione creiamo un pod (un file .yaml) che contiene la lista di container (*kubectl apply -f deployment.yaml*), se vogliamo replicare l’applicazione è possibile indicare il numero di repliche del pod. Per poter accedere ad un’applicazione dall’esterno bisogna esporre il servizio, è possibile assegnare un indirizzo IP virtuale pubblico, oppure è possibile utilizzare l’indirizzo IP pubblico del controller Kubernetes.

* Una delle caratteristiche di Kubernetes è quella di gestire il ciclo di vita delle applicazioni e in particolar modo dei container. Tutto in maniera automatica. Ad esempio, c'è la gestione della replica delle istanze di un container in maniera automatica. Come funziona questo meccanismo?
* Differenze tra Kubernetes e Docker?

**#### Cloud Storage**

* Storage per il cloud (quindi SAN e accenno Ceph, a quel punto chiede cosa c’era prima di Ceph)
* **Raid**
  + Cos’è?
  + Raid 5 structure
  + Secondo te, RAID è utilizzabile/consigliato nel realizzare cloud storage economico stile Ceph?

Risposta: No, non è necessario, gestisce la replicazione e la distribuzione Ceph, è possibile utilizzare RAID con Ceph ma in generale non è consigliato, si possono generare singoli volumi RAID sui nodi di storage per ridurre il numero di OSD attivi (un volume che contiene 5 dischi fisici con un OSD e i file sono distribuiti tra i dischi). Se magari si hanno tanti dischi, RAID puo evitare l’overhead relativo ai tanti OSD (uno per disco).

* + RAID è infallibile come tecnologia?

Risposta: No, per esempio RAID 0 non introduce nulla di nuovo rispetto ad un normale paradigma e non ha replicazione, dunque, soggetto a perdita di dati in caso di guasti. Inoltre, il grado di replicazione anche negli altri modelli è limitato, quindi la perdita di dati è possibile.

* **Ceph**
* What is Ceph, comment its structure
* What are the monitors
* Processo di scrittura (crush algorithm)
* Qual’è il vantaggio di Ceph rispetto alle soluzioni che offrono un approccio diverso?

Risposta: è presente una forte replicazione a dispetto degli altri DFS, tranne il Gluster FS con Replicated volumes, ma esso è meno veloce ed ottimizzato rispetto a Ceph. Inoltre, Ceph non utilizza Lookup tables grazie all’introduzione dell’algoritmo Crush. Non ha single point of failure e non necessita di HW potente come le SAN ad esempio.

* Supponiamo di avere 3 macchine con CEPH installato sopra ed ogni macchina ha 5 dischi. Abiliteresti RAID in aggiunta a Ceph e se sì quale?

Risposta: Raid 0 utile per spartire dati su piu dischi, perché avendo 5 dischi sulla macchina, se metto raid li faccio diventare una unica unità logica (perche? Forse perché l’utente non si accorge che accede ai dati da più dischi), e me li gestisce lui in modo veloce e trasparente. Inoltre levo overhead id avere 5 OSD deamons su una singola macchina.

**#### Swagger**

* Cos’è?
* Qual è il flusso di lavoro che abbiamo fatto in laboratorio? (non ne ho idea)

Risposta: Abbiamo un file employees.yaml che descrive degli impiegati, lo incolliamo nello swagger editor per creare le scheletro dell’applicazione, mettiamo l’indirizzo IP della VM, generiamo il server selezionandolo nell’editor (python-flask), cerchiamo l’installazione nella VM, unzippiamo il file e creiamo nella VM il container rest-server, eseguiamo build e run del container, implementiamo la funzionalità associata alla richiesta REST ed infine creiamo il database per il backend.

* Che interfaccia esponiamo? Risposta: a regola interfaccia REST

Risposta: Swagger è un framework per la progettazione e manutenzione di API RESTful. Data una descrizione della API, Swagger genera lo scheletro del codice con un set di server REST pronti per eseguire lo scheletro. Ha una serie di strumenti per gestire le API come OAS che è un file con tutte le specifiche dell’API che può essere modificato con l’editor online Swagger Editor. Nel cloud computing è utile per fornire una descrizione standard delle API facilitando l’interoperabilità tra servizi, utile per generare documentazione API e testing delle stesse per i servizi cloud

*Non contestualizzate: (qua mi devi aiutare te a capire a cosa si riferiscono)*

* Vantaggi e svantaggi dei container comparato a full virtualization (questa lo è ma magari la seguente si riferisce a questa)
* Come avviene l’invio dei pacchetti in questi contesti? (incapsulamento, tunnel) Non dovrebbe essere legata a quella prima, non è certo a cosa è riferita
* Perché abbiamo bisogno del tunnel in questo caso? Il load balancer non potrebbe mandare i pacchetti direttamente senza tunnel? (Per avere come destinazione il local load balancer)
* Come integrare i paradigmi dei messaggi nell’architettura cloud modulare

Risposta: ogni componente software è un modulo che nasconde l’implementazione interna e comunica con messaggi in un formato noto.

**Puliafito:**

* Parlami del Task Interaction Graph
* Descrivi l’immagine del workflow di MapReduce (con l’immagine davanti)
* dove vedi high order function

**Hadoop**

* 3 principi di hadoop
* Quali caratteristiche aggiunge Hadoop a quelle del functional programming

Risposta: credo i 3 principi di hadoop (Scale out non up, move processing to data, right level of abstraction

* Cosa sono i metodi di setup e clenup? Esempio di dove abbiamo usato setup?

Risposta: usati nell’in mapper combining stateful

* In-Mapper combining. Dove viene effettuata l'operazione di "emit" nello stateful in-mapper combiner
* Pairs and Stripes
* Lettura/scrittura hdfs con schema (come funziona l’albero nella read)
* Hadoop 1.0 e differenze con gli altri
* Node Manager e resource manager come funzionano (spiegare nella slide)
* Come funziona l’esecuzione di una applicazione in Hadoop con YARN
* Yarn scheduler
* Fault Tolerance YARN
* Delayed scheduling
* Descrivi la slide di shuffle and sort
* Speculative execution

**Spark**

* Spark transformation e action

Risposta: la trasformazione permette di costruire un piano logico mentre l’azione permette di attivare la computazione. Le Transformations nel concreto creano un nuovo RDD da un RDD esistente (dai dati originali si possono ricostruire quelli finali tramite il DAG che indica le trasformazioni, dunque le trasformation permettono allo spark context di capire il DAG che rappresenta l’applicazione). Non vanno ad effettuare la computazione/esecuzione del risultato poiché sono "lazy"; la computazione avviene nel momento in cui una Action necessita di dover ritornare un risultato al program driver. Alcuni esempi di Transformations sono: distinct e filter. Le Actions danno istruzione a spark di calcolare un risultato da una serie di trasformations . Ne abbiamo di tre tipi e sono relative a: view data, collect data e write to output. Alcuni esempi di Actions sono: collect, max e min, first … La collect converte l’RDD in una struttura dati locale nello spark driver.

* Parlami di task, job, stage e application in spark

Risposta:

* Applicazione: in spark è un programma completo che viene eseguito su un cluster. Ogni applicazione è gestita da un driver program che contiene il codice principale e utilizza l’API spark per creare e gestire RDD. È composta da uno o più job.
* Job: è una unità di lavoro che viene generata ogni volta che viene chiamata una azione su un RDD, quindi per ogni azione abbiamo un job, il job non fa altro che applicare una sequenza di trasformazioni.
* Stage: il job viene diviso in uno o più stage, rappresenta il lavoro necessario per eseguire una singola azione su un dataset, è un insieme di task in cui non serve lo shuffle. Il numero di stage dipende dal tipo di trasformazioni. Lo stage inizia quando nel job c’è una wide transformation (che richiede uno shuffle), dopo ci sono solo narrow. Lo stage finisce quando è necessaria una operazione di shuffling che richiede una ridistribuzione tra le partizioni, perciò, dopo lo shuffle inizia un altro stage. In realtà se c’è bisogno di uno shuffle, prima di esso abbiamo una stage che termina prima dello shuffle, dopo lo shuffle abbiamo una nuova stage.
* Task: componente dello stage, esegue le trasformazioni, unità di lavoro più piccola, esegue la stessa operazione su una porzione di dati.
* Parlami degli RDD, qual è la sua caratteristica principale? (Immutabilità)

L’RDD è una collezione di oggetti immutabili distribuita nel cluster che possono essere manipolati con operazioni resilienti e parallele. Per distribuzione si intende che le partizioni dell’RDD vengono salvate su nodi differenti. Una volta creato l’RDD non può essere modificato ma possono esserne creati di nuovi con le trasformazione: qui si evince la caratteristica fondamentale chiamata “immutabilità”, utile per ricostruire l’RDD in caso di guasti tramite il DAG.

* Quando viene creata la lineage di un RDD?

Risposta: conoscere la lineage significa conoscere i discendenti che hanno generato la partizione scelta di cui si vuole conoscere, utile per ricalcolare una partizione in caso di perdite. Viene creata alla prima trasformazione.

* Explain distributed sorting

    - Differences between narrow and wide transformation

Risposta: le wide transformations sono quelle in cui i dati devono essere riorganizzati tra le partizioni, dunque una partizione dell’RDD in output può dipendere da molte dell’RDD in input. Lavorano su diverse partizioni RDD. Queste operazioni richiedono uno shuffle dei dati, essi vengono migrati tra i nodi del cluster e questo rende le wide molto piu costose in termini di tempo e risorse. Esempio: groupByKey. Le narrow prevedono che una partizione dell’RDD in input è utilizzata al massimo da una partizione dell’RDD in output, dunque non serve lo shuffle (nessun movimento di dati) e risultano piu efficienti. Esempio: funzione map

    - Differences between executor, task and container

Risposta: l’esecutore esegue uno o più task ed è assimilabile ad un container YARN.

* Persistenza nel RDD Spark

Risposta: un RDD può essere ottenuto riapplicando tutte le trasformazioni che sono state effettuate fino a quel momento, può essere ricalcolato ogni volta che un’azione viene eseguita su di esso. Spark supporta il caching delle partizioni RDD in memoria RAM per velocizzarne l’accesso. “persist()” permette di memorizzare l’RDD dove viene specificato.

* Executors in Spark

Risposta: eseguono il codice assegnato dal processo driver e inviano ad esso un report sullo stato della computazione.

* Differenza fra Transformations e Stage in Spark

Risposta: lo stage permette di eseguire una serie di transformations, prima la wide che effettua uno shuffle e poi tutte le narrow. Sicuro? Non ha solo narrow transformation la stage?

* Differenze tra Hadoop e Spark

Risposta: Spark è un framework che si differenzia da Hadoop per le tecniche di memorizzazione, infatti spark utilizza la memoria RAM mentre Hadoop il disco, dunque il primo risulta piu efficiente. Spark poi usa RDD che è una collezione immutabile distribuita nel cluster che puo essere manipolata in modo resiliente e parallelo, sfrutta le operazioni in-memory dunque è molto piu veloce. Spark utilizza le trasformazioni degli RDD.

**Parte Puliafito ma chiesta da Tonellotto:**

* Come gestisce la fault-tolerance Spark?
* Metodo cache() di Spark è una trasformazione o azione? (r: azione)
* Cos'è un partitioner? Come funziona il partitioner di Hadoop?
* Perchè nei paradigmi visti a lezione per MapReduce bisogna che le funzioni che passiamo siano "stateless"? Le cossiddette higher-order function
* Ruolo del partitioner in MapReduce
* Descrivere approccio stripes vs paired
* Cos'è il Lineage? (Spark)
* Differenza tra trasformazioni wide e narrow?
* Definizione formale di paradigma MapReduce
* Cos'è HDFS? Responsabilità namenode/datanodes
* Differenze tra Trasformazioni e Azioni (Spark)
* Come implementeresti una join a tuple in MapReduce Hadoop?
* Come mai bisogna tenere d'occhio l'occupazione della memoria in Mapreduce in Hadoop?
* Cos'è lo shadow namenode?
* A cosa serve il Combiner?
* Moltiplicazione vettore x matrice in mapreduce
* Architettura di Spark
* Cos'è l'in-memory combining?
* Cos'è la rack topology?
* funzione broadcast in Spark
* accumulator in Spark
* Cosa sono gli struggler? (r: worker particolarmente lenti in hadoop mapred)
* Scheduling in YARN
* Spiegare codice progetto(getConfiguration.setInt-> set shared variable read-only, perché implements Writable,Comparable)
* resource manager
* fare map reduce per comprimere un miliardo di pagine web(dove si fa la compressione->in mapper perché si possono avere quanti mapper ci pare)
* Definizione formale di paradigma di programmazione MapReduce
* Cosa é un input split
* Cos’è HDFS? Che architettura viene utilizzata in HDFS? Parla dell’architettura Master Slave.
* Differenza trasformazioni e azioni in Spark
* Un’operazione di join tra tuple in MapReduce con Hadoop, come la implementeresti? (visto a lezione)
* Perché bisogna controllare l’occupazione della memoria nei programmi MapReduce, in particolare su Hadoop? (Evitare che il GarbageCollector sia saturo)
* Cos’è lo shadow name node? (componente di Hadoop che non ha a che fare con i job ma con HDFS)
* A cosa servono i Combiner?
* Cos’è YARN, qual è la sua architettura e quali sono i suoi componenti? Cos’è un Container?
* Come si implementa la moltiplicazione di una matrice per un vettore in MapReduce?
* Architettura del framework Spark (Architettura Master Slave, Driver ed Esecutori)
* Cosa è un partitioner
* Cos’è inmemorry mapper
* rack topology o network topology
* differenza tra pairing e qualcosaltro
* Cosa è lineage
* Perche l rdd è immutabile
* definizione di paradigna di prog mapreduce (no hadop)
* cosè input split
* caratteristiche file sistem distribuito:
* high availability e affidabilita ottenuta con meccanismo di replicazione attraverso i vari datanode
* hdsf sa dove sono i nodi
* l aprima copia gestita direttamente localmente in hdfs
* la seconda replica in unaltro rack in un nodo casuale
* la terza allocata in un altro nodo dello stesso rack il secondo
* fault tolerance il nhame node responsabile della gestione dei dati decide dove archiviare i dati, il client contatta il name che dice al client dove archiviar ei blocchi tutto gestito dal namenode è un spof quindi, deve esserci active standby in caso di fail del namenode sositturlo prontamente , le strutture vanno replicate o storage condviso o database, l stato va replicato cosi ogni macchina nel momento in cui viene richiesto di essere attiva deve essere pronta
* data lcoality si costriusce il codice
* che tipo di op si svolgono tipicamente
* accesso sequenziale
* nn ho bisogno di accesso casuale come avviene nei normali os
* diff tra trasformazione e azione
* implementazione di un operazione di join in hadoop
* come mai bisogna tenere sotto controllo l’occupazione della memoria nei programmi map reduce
* cos’è lo shadow namenode
* che succedeva se nn usavo il combiner? cosa avrei visto
* moltiplicazione tra matrici
* la matrice viene suddivisa per righe e ogni mapper fara al sua molt riga per colonna che è il vettore che è in memoria e in uscita da l indice del vettor edi uscita in cui salvare il risultato prodotto riga colonna fatto dal mapper il reducer ha l identita
* architettura di spark framework : MASTERSLAVE driver e esecutori
* l esempio del mapreduce immagini cosa prende in input
* accumulator
* a che serve il partitioner serve quando la distribuzione dei valori tra lo spazio intermedio delle chiavi
* Qual è la problematica dei grafi per gli algoritmi in map reduce risposta bisogna mantenere la struttura del grafico tra un iterazione e l’altra e quindi bisogna propagarla