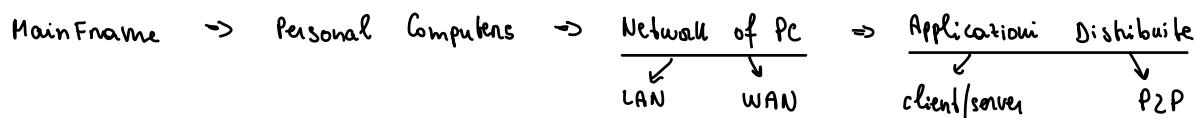


## Evolution of technology

### PC evolution



### Parallel Processing

Multiple processes che lavorano insieme per risolvere un singolo task (grande)

L'unione di questi due concetti ci ha portato ai: Distributed Computing System

Distributed Computing System = Esegue applicazioni distribuite su più computer.

→ Cluster Computing: Gruppi di computer che svolgono op. per completare un certo task comune.

Problema: Un computer gestisce la distribuzione del lavoro.

→ Grid Computing: Non c'è un singolo punto di fallimento

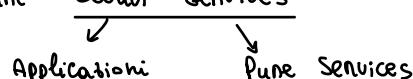
Problema: Non è possibile cambiare l'architettura in modo semplice e non era fault tolerant

→ Hardware Virtualization: Permette di gestire più so sulla stessa macchina, virtualizzando l'hardware

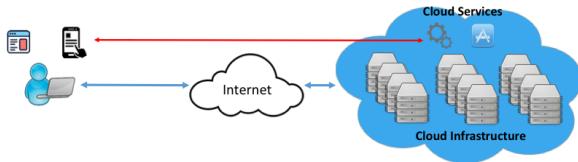
## Cloud Computing

Con Cloud computing si intende un distinct IT environment designed con lo scopo di fornire IT resources (RAM, CPU, HD) in modo scalabile e misurato, accessibili tramite l'internet.

le risorse fornite vengono usate per implementare Cloud Services

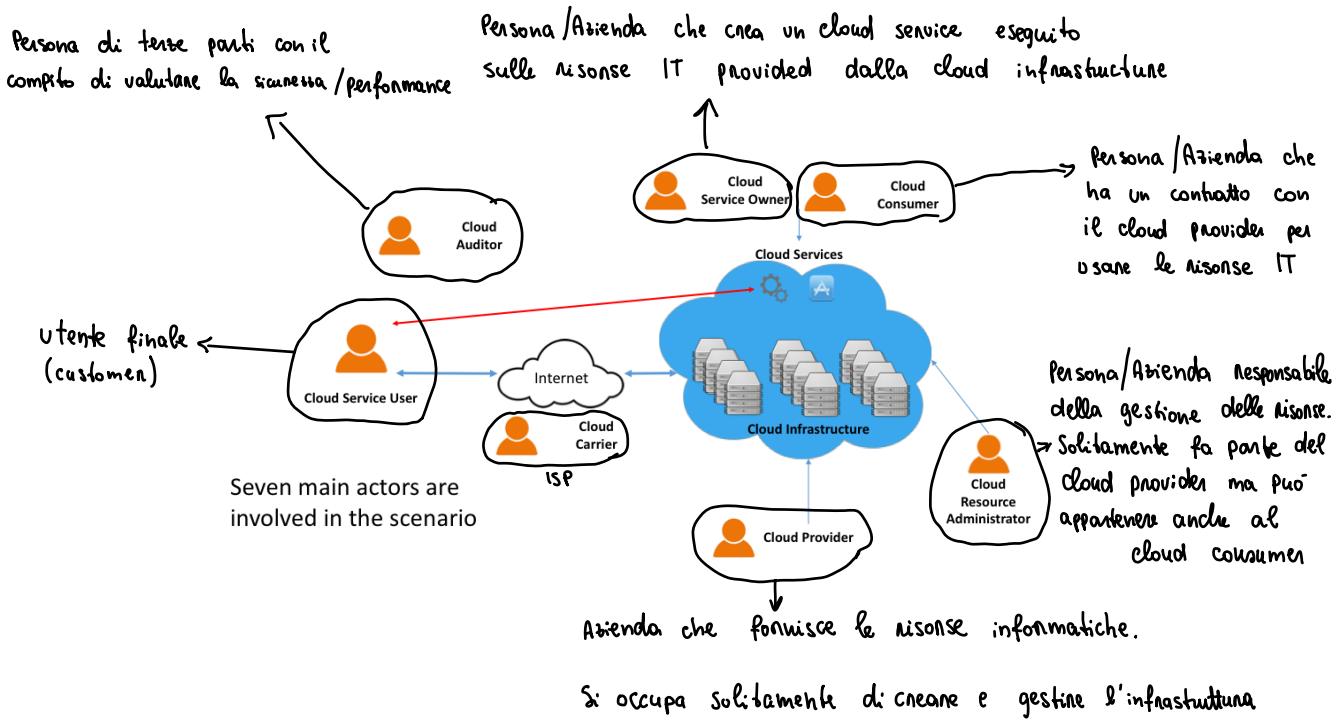


## Cloud Services Architecture



Seguendo questa architettura è possibile fornire qualsiasi cosa come un servizio.

**Roles :** Nel fornire i cloud services ci sono diverse interazioni tra le

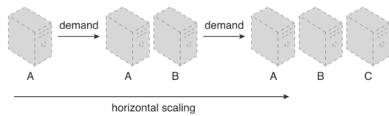


## Utility Based Model and Dynamically Provisioned Resources

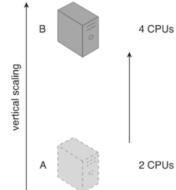
I cloud providers adottano un utility based approach e Dynamically Provisioned Resources :

- 1) L'utilizzo delle risorse viene misurato e i cloud consumers pagano per quanto consumano. **Utility Based Model**
- 2) Le risorse vengono allocate / deallocate AUTOMATICAMENTE a seconda dell'utilizzo. **Dynamically Provisioned Resources**

→ Scaling Orizzontale: Allocare o rilasciare lo stesso tipo di risorse



→ Scaling Verticale: Rimpiattare le risorse con risorse più/meno potenti



## Cloud Infrastructure

La cloud Infrastructure è sviluppata sui Datacenter

↓  
spazio dove vengono collocati server e cani.

I server del Datacenter, per far sì che vengano sempre utilizzati, vengono assegnati a più clienti contemporaneamente. Naturalmente, ogni cliente vuole gestire le proprie risorse (S.O., applicazioni ecc...). Abbiamo quindi bisogno di una infrastruttura multi-tenant, cioè che sia in grado di gestire più amministrazioni (dare l'impressione ai cloud consumer di avere il completo controllo delle risorse assegnate). Per questo viene usata la virtualizzazione.

## Virtualization

La virtualizzazione permette di creare copie multiple virtuali delle risorse reali (Virtual Machines).

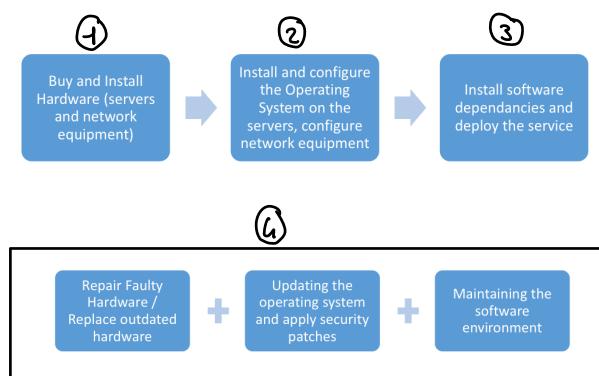
Questo ci permette di assegnare ad ogni customer una VM differente sulla quale un S.O. completo può essere fatto girare.

Il server viene chiamato host.

Le varie VM vengono gestite da un Hypervisor, un software che ricrea l'hardware virtualizzato nel quale il "guest S.O." viene eseguito.



## Conventional Computing Paradigm



Inoltre il sistema può avere bisogno di scalare:

- Comprare nuovo hardware
- Ridurre l'hardware → Perdita di risorse

## Cloud Computing

Il cloud computing permette di non fare nessun investimento iniziale, pagare per le risorse utilizzate e scalare in modo semplice.

Tutto questo a un prezzo basso, questo creando le infrastrutture:

- Grandi: Comprare molte risorse insieme pagando meno
- Shared: Share infrastrutture tra i customer

Tutto ciò può essere paragonato alle aziende che offrono energia elettrica e ti permettono di non creare l'infrastruttura.

## Altri Vantaggi del cloud computing

- 1) Minimal Management Responsibility: L'azienda customer non deve gestire l'infrastruttura.
- 2) Higher Quality of Service: La qualità è maggiore essendo gestita da esperti che si occupano solo della gestione del cloud
- 3) Reliability: Possibilità per l'azienda di offrire servizi più affidabili
- 4) Continuous Availability: Possibilità per l'azienda di offrire servizi sempre disponibili
- 5) Minimal Software Management: Alcuni cloud permettono di non gestire l'update e le licenze dei software
- 6) Location Independent: Servizi accessibili da ovunque e sempre
- 7) Le aziende sono più concentrate sul proprio business

## Cloudification

I vantaggi del cloud computing hanno portato alla "cloudification".

Per cloudification si intende il muovere le applicazioni e servizi dal local computing deployments all' infrastruttura cloud.

## Use-cases

- 1) Grossi aziende scommettono alcune delle loro attività su un cloud system per ridurre il costo fisso
- 2) Piccole aziende / startups possono trasformare le loro idee in business senza troppo costo iniziale
- 3) Aziende che richiedono alte capacità computational ma in modo sporadico, solo solo per un breve lasso di tempo senza mantenere la loro infrastruttura
- 4) Poder rendere servizi disponibili a livello globale
- 5) Poder collocare una quantità importante di dati e distribuire content a livello globale

## Rischi / Difficoltà del cloud Computing

Portabilità limitata: la standardizzazione delle tecnologie del cloud computing è ancora limitata. Differenti providers di cloud computing hanno soluzioni differenti che sono proprietarie e interoperabili tra loro.

Legal issues: I provider di cloud computing costituiscono i data centers in posti di convenienza e di conseguenza fuori dal loro stato. Potrebbero intercorrere problemi legali.

Data security: Essendo i dati trasmessi sul network potenzialmente non sicuro e non affidabile, richiede meccanismi di sicurezza.

# Cloud Computing Model

Anche se la standardizzazione è limitata, abbiamo bisogno di un modello standard per convenienza di comprensione.

## NIST MODEL

Il Nist model definisce gli aspetti e caratteristiche base del cloud computing:

- 1) **Caratteristiche essenziali:** L'insieme delle features obbligatorie che un cloud computing system deve avere.
- 2) **Service Models:** l'insieme di differenti tipi di servizi che sono forniti dai consumers
- 3) **Deployment Models:** L'insieme dei differenti modelli per lo sviluppo di infrastrutture cloud.

### Caratteristiche essenziali:

**Broad Network Access:** L'infrastruttura deve poter essere accessuta da ovunque

**Rapid elasticity:** le risorse sono allocate/deallocate rapidamente

**Measured Service:** le risorse sono misurate e fatte pagare in base all'utilizzo.

**On-Demand:** le risorse vengono allocate su richiesta

**Resource Pooling:** le risorse computazionali sono installate e disponibili su richiesta ai consumatori.

Sono mantenute in modo remoto e devono essere grandi abbastanza per soddisfare molti utenti contemporaneamente.

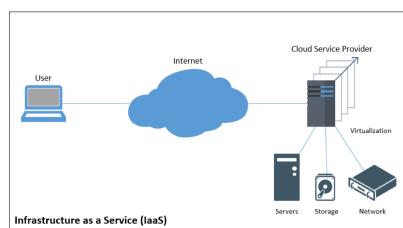
### Service Models

#### Infrastructure as a Service (IaaS)

Fornisce risorse hardware virtualizzate ai consumers.

Queste risorse possono essere usate come hardware reale per creare qualsiasi computer setup (VM)

IaaS rimuove il costo del costituire e mantenere l'infrastruttura fisica



## Platform as a Service (PaaS)

Fornisce l'accesso a una computing platforms nella quale il consumer può disegnare e sviluppare le componenti dell'applicazione.

Tutti gli aspetti hardware sono gestiti dal provider.

Problema:

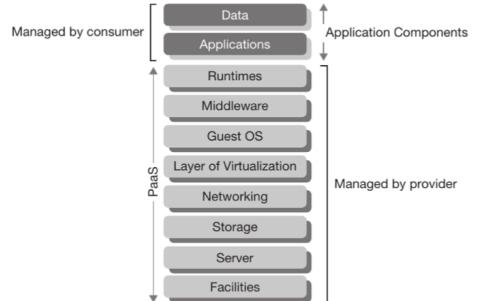
Le applicazioni devono essere programmate usando le APIs del sistema. Per questo motivo, le applicazioni programmate per una piattaforma cloud difficilmente possono essere trasferite su un altro different cloud.

## Software as a Service (SaaS)

Fornisce ai consumers le applicazioni come servizi.

Queste applicazioni sviluppate dai cloud provider sono offerte solitamente tramite web interface ai consumers.

Tutti gli aspetti hardware e software vengono gestiti dal provider.



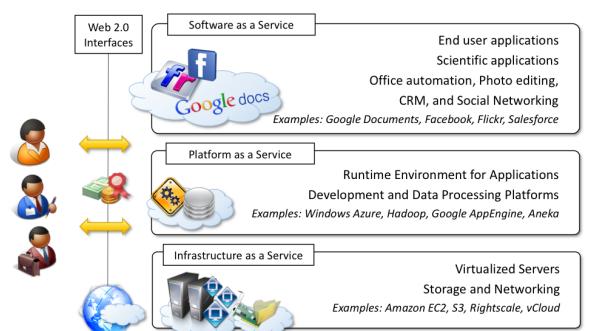
## PaaS / SaaS implementation

PaaS e SaaS possono essere implementati sopra una infrastruttura IaaS.

PaaS → IaaS: Software creato sulle VMs disponibili

SaaS → PaaS: PaaS platform è usata per creare il software offerto come SaaS

## Model Examples



## Deployment Methods

Il deployment di una infrastruttura cloud può essere fatto in diversi modi, dipendentemente dai requirements della consumer organization.

### Public Clouds

È il deployment più popolare. In questo modello la infrastruttura cloud è gestita da una organizzazione / azienda esterna che vende servizi / risorse. L'accesso è da remoto e le risorse sono condivise da più consumers.

### Private Clouds

L'infrastruttura è costituita con tutte le features di un sistema cloud ma è utilizzata internamente all'azienda (accesso limitato).

Di conseguenza, la gestione è responsabilità del consumer o di un'azienda esterna fidata.

### Community Clouds

L'infrastruttura cloud è condivisa da consumer appartenenti alla stessa community.

Sono una forma di private clouds generalizzata. Il modello supporta il concetto di multi-tenancy. I costi sono condivisi tra la community. Il pay-per-use model può essere utilizzato.

### Hybrid Clouds

Sono cloud formati combinando infrastrutture cloud private e pubbliche.

Il consumer ottiene sia low-cost computing e risorse scalabili che controllo e privacy

### Public vs Private

Private Cloud	Public Cloud
It can be both of types of on-premises and off-premises.	There cannot be any on-premises public cloud deployment.
On-premises private cloud can be delivered over the private network.	It can only be delivered over public network.
It does not support multi-tenancy feature for unrelated and external tenants.	It demonstrates multi-tenancy capability with its full ability.
The resources are for exclusive use of one consumer (generally an organization).	The resources are shared among multiple consumers.
A private cloud facility is accessible to a restricted number of people.	This facility is accessible to anyone.
This is for organizational use.	It can be used both by organization and the user.

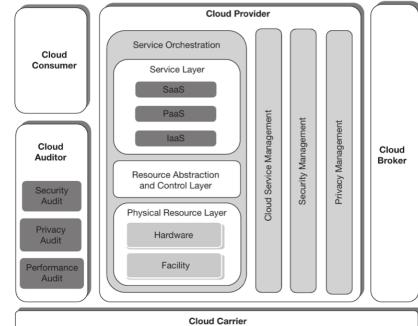
### Selezionare il modello

- The choice of the appropriate model depends on several factors
- Mainly it depends on the business needs (e.g. data security, low latency) and on the size of the consumer organization
- Consumers should carefully analyze their needs and the pros and cons for each deployment model and derive the requirements
- For general use any public cloud deployment is a good option
- Private/community deployments become an option when the company has concerns (or specific requirements) on data privacy or sensitive business related data
- Budget is another factor, the cost of migration and the cost of ownership of the infrastructure must be considered before selecting a deployment

## NIST Architecture

Definisce l'insieme di elementi che compongono l'ecosistema cloud. Si concentra su cosa compone una infrastruttura cloud e non come è implementata.

Specifica anche gli attori coinvolti.



## Actions

Cloud Consumer: Chi usa l'infrastruttura cloud

Cloud Providers: Chi fornisce l'infrastruttura cloud

**Cloud Broker:** Entità di terze parti che si mette nel mezzo, in alcuni casi, tra provider e consumer. Gestisce la fornitura del cloud service da differenti providers al consumer e negozia le relazioni. Tramite il broker, i consumers possono evitare la responsabilità di interazione direttamente con il provider e può beneficiare del riduzione i costi, richiedendo servizi da più provider.

## Cloud Auditor

# Cloud Comien

# Virtualization Technologies

Per virtualizzazione intendiamo la capacità di rappresentare le risorse fisiche in maniera simulata usando un software layer aggiuntivo.

La virtualizzazione ci permette di creare virtual machines sulla macchina fisica.  
(guest system) (Host system)

Ogni VM rimane indipendente dalle altre e indirettamente accede all' HW dell' host system.

## Hypervision

L' hypervisor è un software layer (Virtualization layer) che crea gli environment per le VM.

Permette l'accesso alle risorse del sistema da parte delle VM e controlla e monitora l'esecuzione di quest'ultime sull' HW.

### Hypervisor Types

Ci sono due differenti tecniche per la server virtualisation:

- 1) Hosted Approach: Un OS è installato sul physical machine. Successivamente un hypervisor viene installato su esso.

Pro: Più facilità di installazione e configurazione dell'hypervisor, dovuta al fatto che l'OS gestisce gli hardware drivers.

Contro: Tutte le richieste delle VM passano dall'OS e di conseguenza le performance potrebbero essere ridotte.

- 2) Bare Metal Approach: L'hypervisor è installato direttamente sulla macchina fisica.

Pro: Fornisce l'accesso diretto all'HW alle VM  $\Rightarrow$  Migliori performance

Contro: Limited hardware support  $\Rightarrow$  Non può ruotare su molte piattaforme HW

### Virtualisation Techniques

- 1) Full virtualisation: L'hypervisor simula completamente l'HW. Questo permette di creare VM con versioni non modificate degli S.O disponibili.

- 2) Para virtualisation: Una porzione della gestione della virtualizzazione viene assegnata al guest S.O. le normali versioni degli S.O non possono essere utilizzate in questo caso e richiedono modifiche speciali (Porting).  
Dovuto a ciò, il guest OS ha bisogno a priori di sapere se verrà eseguito su una piattaforma virtualizzata, nello specifico su quale hypervisor verrà eseguito.

Pro:

- Riduce il virtualization overhead dell'hypervisor rispetto alla full virtualisation
- Hypervisor non contiene nessun device driver, sono contenuti dal guest S.O

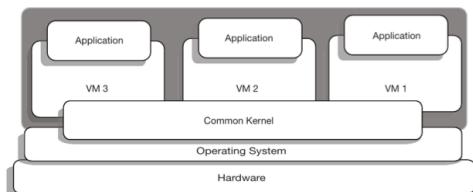
Contro:

- Versioni non modificate degli S.O non sono compatibili.
- la sicurezza è compromessa essendo che il quest OS ha più controllo sull'HW.

3) HW assisted Virtualization: Alcune AMD e INTEL CPU gestiscono direttamente loro le calls effettuate dalle VM.

Permette di eliminare la traduzione da parte dell' hypervisor.

4) OS level Virtualization: Non viene usato un hypervisor e le VM sono gestite dal kernel del S.O della host machine.



Il kernel dell'host system è condiviso tra tutte le VM e si occupa di creare più VM distinte sul singolo kernel.

PDO: più leggero essendo che tutte le VM sono su un singolo kernel.

Questo permette a un singolo system fisico di supportare più VM rispetto al numero di VM COMPLETE che potrebbe supportare.

Contro: Tutte le VM devono usare lo stesso S.O.

OSS: Differenti distib. dello stesso kernel possono essere usate.

5) Altri tipi di virtualizzazione

a) Network Virtualization: processo di combinare risorse e funzionalità network in una singola entità chiamata Virtual Network.

Simula l'infrastruttura network reale per la data transmission

b) Storage Virtualization

## Vantaggi virtualizzazione

- 1) Possibilità di eseguire più VM sullo stesso HW (server consolidation) e permette di aumentare l'utilizzo HW
- 2) Reduce HW e infrastructure cost
- 3) Semplificare l'amministrazione del sistema, diversificando tra guest e host
- 4) Semplifica la system installation, un nuovo sistema può essere creato clonando un altro.
- 5) Fault tolerance e zero downtime maintenance permettendo lo spostamento delle VM su hw differenti e backuppando e replicando le VM
- 6) Sicurezza essendo le VM isolate

## Svantaggi virtualizzazione

- 1) Ogni VM ha un singolo punto di fallimento, se fallisce una macchina fisica, diverse VM down
- 2) lower performance, VM possono raggiungere l'85/90% delle performance essendo che non accedono direttamente all'HW.

## Emulation

Si indica la capacità di un sistema di imitarne un altro. Utilizza un emulator per tradurre le CPU instructions.

Tipi di implementazione:

- Binary translation: Consiste in una total conversion dei binary data.  
La conversione ricompila tutte le instruction in un'altra binary form per l'altra piattaforma.
- Interpretation: Ogni istruzione è interpretata dall'emulator ogni volta che viene incontrata.  
→ Più lenta anche se più semplice da impl.

## Emulation vs Virtualization

Virtualization	Emulation
The virtual machine may execute the code directly that's available in various languages.	An emulator needs an interpreter to translate the source code.
In virtualization, hardware may be directly accessible.	In terms of emulation, we need a software connector to access hardware.
Virtual Machines solutions are costlier than the emulator.	It is comparatively cheaper than virtualization.
Virtual Machines are relatively quicker in their operation.	Emulators are comparatively slower than virtualization.
Virtualization offers better backup features.	Emulation falls short of virtualization as far as backup and recovery are considered.

## Virtualization technologies

### Virtualization requirements

Equivalence: Gli S.O che girano sulle VM dovrebbero comportarsi allo stesso modo di quando vengono eseguiti su macchine fisiche

Resource Control: L'hypervisor deve essere in controllo delle risorse fisiche e l'OS sulle VM deve avere il controllo sulle risorse virtualizzate

Efficiency: Una frazione dominante delle istruzioni macchina devono essere eseguite senza l'intervento dell'hypervisor.

Questi requirement sono simili a quelli del multiprogramming

---

### Multiprogramming

In Multiprogramming emuliamo una macchina con più processi rispetto all'hw reale.

Vengono creati dei virtual processor (VCPUs) sui quali i processi devono avere l'impressione di avere in completo controllo il processore ed è l'unico processo eseguito su esso.

### VCPU esecuzione

L'OS crea una virtual representation del processore, che contiene una copia dei suoi registri. Una macchina con un processore fisico emula solo un processore virtuale alla volta, questo è ottenuto caricando i registri del processore host con i valori salvati nel virtual processor e successivamente facendo continuare al processore host l'esecuzione. In qualche momento l'esecuzione viene stoppata e le date structure del virtual processor vengono aggiornate con il valore corrente dei registri dell'host, successivamente un nuovo virtual processor può essere selezionato per l'esecuzione e così via.

PS: Assumiamo che tutte le VCPUs condividono la stessa memoria.

### Context Switching

Per context switching intendiamo l'operazione di switch da un processore a un altro.

È determinata da un timer o un interrupt.

### Multi programming support

- Interruption : Timer periodico che permette un periodic context switching.
  - 1) Salva lo stato delle host CPU nella VCPU representation
  - 2) Seleziona una nuova VCPU da eseguire e loda il suo stato nei registri dell'host CPU
  - 3) Esegue una special instruction per saltare all'address salvato in net e ritornare in user mode
- Memory protection Mechanism: Isolano VCPUs per prevenire l'accesso di un processo ai registri delle altre VCPUs.
  - 1) usn/sys register : flag che specifica se la CPU è attualmente in system o user mode
  - 2) sys/mem register : contiene l'indirizzo di inizio delle privileged memory dove le strutture dati delle VCPUs sono salvate
  - 3) net register : salva un memory address da essere usato quando si ritorna da una interruption
- Copia automatica dello stato dei registri
- privilege levels : System (accesso a tutta la mem.) o User (accesso al subset che non include le VCPU data structure)

### Virtual Memory

È un meccanismo introdotto per permettere ai processi di usare un virtual address space (VAS).

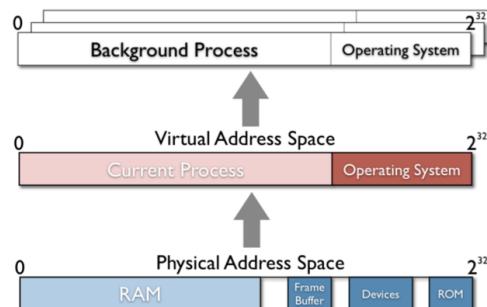
Un VAS è creato e assegnato per ogni applicazione e permette di dare l'impressione ai processi di avere accesso all'intera mem. fisica

È implementata attraverso due meccanismi:

- Address translation :

L'MMU si occupa di tradurre gli indirizzi virtuali in indirizzi fisici

- Virtual Address space management : Viene fatto dal So e si occupa di gestire lo spazio virtuale creato per ogni applicazione.



L' MMU gestisce la traduzione degli indirizzi usando la page table, la quale è gestita dall'os. La page table è divisa in page directories, ognuna contiene info per tradurre una porzione del VA. La page table è organizzata in una struttura multi livello, più page directories devono essere accedute per tradurne il VA in PA.

### Segmentation

Lo spazio della virtual memory può eccedere quello della memoria fisica, usando uno storage secondario, ad esempio un hard disk. In questo caso, alcune pagine vengono salvate al di fuori delle RAM, quando non usate.

Quando un processo cerca di accedere a una pagina non caricata in mem., avviene un page fault e l'os muove la pagina nella RAM. L'os aggiorna le page table.

### Interrupt / Exceptions

Interrupt / Exceptions sono usate per notificare il sistema di un evento che necessita attenzione immediata durante l'esecuzione del programma (es: page fault).

Quando un interrupt/eccezione occorre, avviene il cambio di contesto da user mode a kernel mode e una specifica funzione dell'os viene eseguita. Successivamente, ritorna fatto il cambio di contesto.

Eccezioni: Usate per gestire errori interni al programma.

Interrupt: Usate per notificare alla cpu eventi esterni

Per linkare gli interrupt / exceptions ai loro handles viene usata la Interrupt descriptor table.

### Full Virtualization

Nella full virtualization, l'hypervisor (VMM) deve dare alle VMs l'impressione di avere il completo controllo del physical HW.

### Hypervisor

Ha come obiettivo la creazione di una virtual representation del sistema risparmiando l'hw.

Se host e guest hanno la stessa architettura, la creazione può essere implementata minimizzando

l'adozione di emulation la quale introduce un overhead significativo.

### Virtualizing CPU

L'hypervisor adotta la stessa tecnica adottata per creare le VCPUs in un multiprogramming environment:

- L'hypervisor code è eseguito nel system/kernel space, il guest OS code è eseguito nello user space.
- L'hypervisor carica lo stato di una VCPU nell'host processor e in seguito permette all'host CPU di eseguire il target code com'è, fin quando la CPU trova un'istruzione che non può essere eseguita direttamente.
- Quando avviene il context switch, il VMM riottiene il controllo e emula le target instruction che non potevano essere eseguite dal guest OS e in seguito fa il re-load del virtual CPU nel host CPU per continuare l'esecuzione.

Se più VM sono eseguite sullo stesso host, il VMM può schedulare un timer per triggerare il context switch.

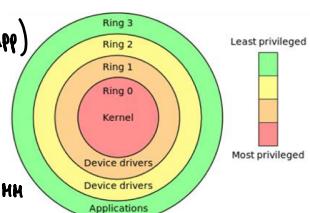
### Guest OS execution

In guest host (VM) vorrebbe eseguire il codice come se avesse il controllo completo del host hardware e, inoltre, eseguire non solo le ist. a user level ma anche a system level.  
Il VMM deve quindi emulare l'intero processore per poter supportare il guest OS.

Le eccezioni devono essere gestite, non vogliamo che la VMM killi una VM.

Il guest system è eseguito nello user-space : Ring 0 (OS), Ring 3 (user app)

1) Ogni volta che una privileged instruction viene eseguita, viene sollevata una eccezione che viene capturata dalla VMM ed avviene il cambio di contesto VM → VMM



2) la VMM esegue una binary translation per eseguire le privileged instruction del guest OS, emulando il suo comportamento. Se il numero delle privileged instructions è limitato, le performance non sono affatto significativamente

## Virtualizing Physical Memory

Per la memoria della VM viene usato un sottoinsieme di memoria dell'host. Le guest VM devono avere accesso solo alla porzione di memoria a loro assegnata e devono pensare che il physical address della loro memoria inizi da 0. Una parte della memoria fisica viene riservata per l'esecuzione delle VM. Per mappare una porzione della mem. fisica con la mem. fisica dell'host, viene usata la MMU dell'host CPU. Essa viene configurata con l'address range assegnato a ogni VM. Ogni qual volta un'istruzione della VM viene eseguita, l'address viene tradotto. I page fault vengono gestiti dalla VMM.

Il guest OS dovrebbe essere in grado di preparare e usare la sua traduzione da virtual address a Physical Address creando e gestendo la sua page table.

Il guest host dovrebbe essere capace di definire una funzione G, che mappa il virtual address del guest host con il physical address del guest host. La VMM è responsabile di creare e mantenere una funzione H che mappa dal guest physical address all'address fisico della memoria.

Il sistema necessita quindi di creare una virtual MMU che implementi le due funzioni G-H, cioè traduce da guest physical address a host physical address.

La page table è modificata per aggiungere un livello aggiuntivo (shadow page tables) che implementa la H function.

Queste shadow tables sono settate come write-protect, ogni possibile azione di modifica punta all'uscita dalla VM. Questo metodo ha un overhead significativo richiedendo diversi context switch e traduzioni continue degli indirizzi da parte della VMM. Per aggiungere lo shadow level e mantenerlo aggiornato, la VMM deve catturare tutte le azioni relative all'MMU e la page table:

- Cambio del PTBR (registro che punta al primo indirizzo della page table)
- Cambio dei valori della page Table in qualche directory.

## Virtualizing I/O devices

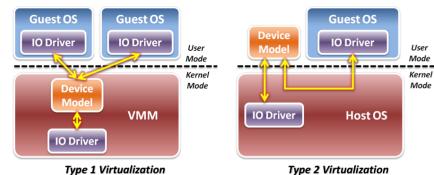
La VMM deve emulare l'HW reale creando una rapp. virtuale implementata come un insieme di strutture dati in memoria.

La VMM si occupa di tradurre i comandi dalla periferica virtuale a quella fisica o di emulare il device completamente.

La VMM crea un device model il quale è la virtual representation del device.

Il device model può essere implementato in due modi:

- 1) Come parte della VMM
- 2) Stand alone service running in user space



## Interrupt Management

La VMM si occupa di gestire gli interrupt che provengono dai device fisici.

VMM → Mantiene una "host interrupt description table" che l'host CPU usa per gestire gli interrupt

VM → Ogni VM ha la sua "interrupt description table" gestita dal guest OS

Quando la VMM vuole emulare la ricezione di un interrupt in una VM, deve guardare la guest interruption table e fare le operazioni richieste per eseguire l'interrupt code nel guest os:

- 1) Salvare lo stato corrente
- 2) Cambiare l'instruction pointer alla prima istruzione dell'interrupt code
- 3) Quando la VMM restituisce il controllo, la guest VM deve eseguire l'interrupt code.

OSS: la guest CPU può disabilitare gli interrupt e la VMM deve aspettare fin quando non sono nuovamente attivati prima di emulare la interrupt reception

## Performance Panality

Ogni volta che la VM switcha nel Kernel space, abbiamo un context switch nella VMM.

## Hardware Assisted Virtualization

Per risolvere il problema di performance, AMD e INTEL hanno esteso le loro CPU con nuove funzionalità disegnate per supportare implementazioni efficienti del VMM-hypervisor software.

Ci focuseremo sull'estensione intel chiamata VMX (Virtual Machine eXtension).

### VMX (Virtual Machine eXtension)

VMX introduce due nuove operating modes nella CPU intel: Root e non-Root.

per la VMM  
per la VM

Esse sono ortogonali alle modes esistenti: Kernel (system) e user.

Abbiamo quindi 4 possibili combinazioni (in ordine di privilegi):

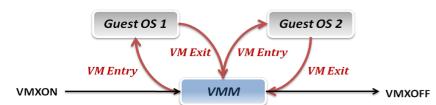
- 1) root / system } Abbiamo bisogno di questa distinzione perché permette l'implementazione dove
- 2) root / user    } la VMM è parte di uno standard OS running sull'host, nella quale le user space appl. runs in root/user e il kernel os, VMM op. sono eseguite in root/system
- 3) non-root / system
- 4) non-root / user

Lo scopo principale delle due modes è di introdurre limitazioni controllate sull'hw nelle azioni della VM.

Ogni qual volta che la VM esegue istruzioni non consentite, vengono catturate a livello hw e si switcha alla VMM execution. Però un insieme di primitive posso essere eseguite direttamente dalla VM.  
*solo in root/system mode*

Sono aggiunte ist. assembly: **VMLAUNCH**, **VHRESUME**, VM Exit

Esse vengono usate per il context switch tra VM e VMM.



L'esecuzione di INT e IRET può essere fatta senza l'utilizzo del VMM code, ogni qualvolta l'INT è invoked l'operating mode switcha da non-root/user a non-root/system e viceversa per IRET.

### Virtual Machine Control Structure (VMCS)

È una struttura di controllo introdotta da VMX che contiene tutte le info associate con lo stato di una VM. Vengono introdotte un insieme di istruzioni per manipolare la VMCS che vengono usate durante i cambi di contesto per eseguire la VMM o quando la VM viene ripristinata.

La VMCS contiene diversi campi che possono essere raggruppati come segue:

**Guest state:** Salva lo stato dei virtual processor associati alla VM. Lo stato è caricato dalla struttura quando la VM viene eseguita e viene salvato all'uscita.

Contiene l'instruction pointer relativo alla prima istr. da essere eseguita dalla VM quando caricata.

**Host state:** Salva lo stato della CPU fisica prima dell'esecuzione di una VM. Può contenere lo stato delle VMM prima del VM launch. Lo stato è ripristinato durante un VM exit.

Contiene l'instruction pointer relativo alla prima istr. da essere eseguita dalla VMM dopo una VM exit.

**VM execution control:** Questo field specifica quale tipo di operazioni sono/non sono consentite in non-root mode. Un'azione non consentita causa una VM exit.

Contiene diversi flag, in particolare determina la gestione degli interrupt e I/O.

Un flag determina cosa dovrebbe succedere quando la CPU riceve un interrupt esterno quando è in non-root mode, se può essere gestito dalla VM o se causa un VM exit.

Un altro flag specifica se le I/O op. sono permesse all'interno della VM o se è richiesto un VM exit.

**VM exit control:** Contiene diversi flag e fields che determinano alcuni optional behaviors delle transitioni da root → non-root

**VM enter control:** Contiene diversi flag e fields che determinano alcuni optional behaviors delle transitioni da non-root → root

Ci sono field che possono essere usati dalla VMM per simulare una falle external interrupt o per iniettare eccezioni o faults.

Con questo obiettivo, la VMM scrive un vettore di interrupt desiderati nel VM enter control. Durante il VM enter, il processore performa tutte le azioni in risposta agli interrupt, nello specifico: salva lo stato nel guest stack e guarda la guest interrupt descriptor table per determinare l'indirizzo dell'interrupt handler.

VM exit reason: contiene le info relative alla motivazione che ha causato l'ultima VM exit.

## RAM management

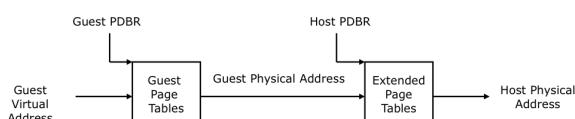
Sia AMD che intel hanno introdotto la "extended page table".

L'HW che implementa la host MMU è esteso per avere due puntatori PTBR, uno che fa riferimento alla page table della guest VM e l'altro che fa riferimento alle page table create dalla VMM.

L'HW esteso si occupa di applicare sia la G function che la H function.

Con questa estensione, non c'è più bisogno della VM exit ogni volta che la guest VM modifica la page table.

OSS: Questa estensione è più costosa in termini di address translation, richiedendo più accessi in memoria. Per ridurne questo overhead le nuove MMU hanno spesso una page table cache per ridurre gli accessi.



## Hardware passthrough

Per ridurre l'overhead causato dalle partecipazioni della VMM in ogni interazione I/O, una soluzione è dare alla VM l'accesso diretto e esclusivo alle periferiche.

Questo approccio è chiamato passthrough e mappa il device direttamente alla guest VM.

Per implementare questo approccio c'è bisogno di due main functionalities:

1) I/O Mapping : Vmx include nel vmcfg un I/O bitmap con un bit per ogni possibile I/O address per supportare specificatamente l'hardware passthrough

I/O bitmap viene modificata in modo da impostare tutti i bit corrispondenti all'indirizzo dell'I/O space del device, per garantire l'accesso diretto.

Nella modalità non-root, la CPU prima verifica la bitmap quando un'operazione I/O viene svolta, se il bit è settato allora viene completata l'istruzione, altrimenti avviene una VM exit e viene dato il controllo alla VMH.

2) Interrupts : Intel VMX permette due opzioni per la gestione degli interrupt

- L'interrupt causa una VM exit e viene gestito dalla VMH
- L'interrupt è gestito direttamente dalla VM senza VM exit

Quando abbiamo un HW passthrough, l'HW è capace di guardare direttamente la Interruption Vector Table (IVT) del guest OS e eseguire il suo handler. Il sistema deve essere capace di gestire gli interrupt da un passed-through device anche quando la VM non è eseguita.

La VM può essere in diversi stati :

1) Running : Quando la VM CPU sta usando la host CPU.

Quando è in questo stato l'interrupt dovrebbe essere gestito direttamente dal processore, senza l'intervento della VMH

2) VM ready: Quando la VM CPU è stoppata perché la host CPU è in uso dalla VMH o un'altra VM.

Essendo che la CPU sta eseguendo qualcosa' altro, quando avviene un interrupt vogliamo salvare la richiesta da qualche parte per fare in modo che la VM CPU possa gestirla più tardi quando la VM torna in esecuzione

3) VM halted: È uno stato che avviene quando la VH esegue la hlt instruction. Viene messa in questo stato da parte della VMM e può essere resumed quando riceve un interrupt. Gli interrupt devono essere salvati e lo stato della VH rapidamente cambiato in ready.

### Posted interrupt

È il meccanismo usato per salvare gli interrupt. Per ogni VH vengono utilizzate due strutture dati aggiuntive.

### Interrupt Remapping Table

Permette di indirizzare ogni HW interrupt direttamente dalla VMM in modo dinamico. Ha una entry per ogni possibile richiesta di interrupt. Ogni entry mappa la richiesta verso l'interrupt vector di una VH o PiD. Ogni volta che un interrupt avviene, l'IRT decide quale interrupt vector o PiD lo gestisce.

### Posted Interrupt Descriptor (PiD)

Il PiD contiene info sullo status della VM e salva le informazioni sulle interrupt notifications per notifiche asincrone.

Contiene i seguenti field:

- 1) Posted Interrupt Request (PiR) : Dove gli interrupt vengono postati
- 2) Suppess Notification (SN) : Determina se il controller dove aver postato l'interrupt nel PiR deve notificare la CPU ( $SN=0$ ) o no ( $SN=1$ )
- 3) Notification Vector (NV) : Punto all'attuale interruption vector per notificare interrupt alla VCPU della VM

Quando l'interrupt controller necessita di deliverare l'interrupt alla VM e il posted interrupt mechanism è attivo, la corrispondente entry nel IRT punta una PiD istante.

In questo caso l'IRT performe le seguenti op:

- It sets the proper bit in the PIR
- If SN is 1 (e.g. because the VM is in ready state), it does nothing else
- Otherwise:
  - If the VM is running it interrupts the processor by using the vector NV
  - If the VM is halted it triggers the awakening of the VM

Come lo stato della VM cambia, la VMH deve aggiornare il PiD status:

- VM va in running state  $\Rightarrow$  VMH imposta SN=0 e NV in un vettore chiamato

Active Notification Vector che punta IDT della VM

Quando avviene questo cambio di stato, la VMH deve controllare il PiR e se un qualsiasi bit è settato, deve cambiare il NV nel modo corretto quando "avvia" la VM.

- VM va in ready state  $\Rightarrow$  VMH imposta SN=1

- VM va in halted state  $\Rightarrow$  VMH imposta SN=0 e NV in un vettore chiamato

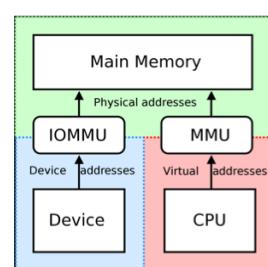
Wake up Notification Vector che trigerà il VM awakening

## I/O devices

Sono I/O device, istruiti dalla CPU, che leggono direttamente dalla RAM.

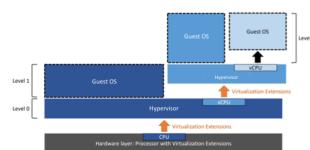
Usano una MMU specifica chiamata IOMMU.

La IOMMU può essere usata per la virtualizzazione ed è capace di triggerare page fault interrupt se le pagine non vengono trovate nella RAM.

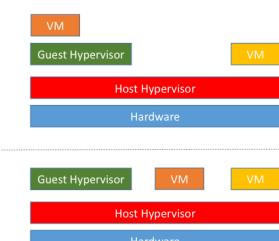


## Nested Virtualization

- Nested virtualization describes a system in which the hypervisor runs inside a VM, which in turn is running inside a host hypervisor
- Nested virtualization is useful mainly for testing, although today is used also in solutions for production (it reduces the effort for upgrading the infrastructure)
- It allows cloud consumers to setup their own IaaS infrastructure on top of a set of VMs running on a IaaS infrastructure



- Multi-level nested virtualization requires specific hardware support
- Intel and AMD only supports a single virtualization level
- Multiple virtualization levels can be multiplexed into a single virtualization level, i.e. the VMs running on top of the guest hypervisor are run as the other VMs running on top of the host hypervisor
- Guest hypervisor access to hardware extensions for virtualization are emulated by the Host hypervisor



## Pana Virtualization

Prima che l'hardware support venisse usato, un'alternativa alla full virtualization era la panavirtualization.

In questo approccio abbiamo sempre la VMM, ma l'hypervisor è implementato invitando dall'assunzione che il guest os può essere modificato e può essere avvisato del fatto di essere eseguito all'interno di una VM.

## Anchitettura

Modificare il guest OS in modo che alcune operazioni (ad esempio I/O) sono sostituite con una chiamata a specifiche APIs fornite dalla VMM tramite un virtualization layer. Queste funzioni esposte dalla VMM vengono chiamate Hypercalls e sono eseguite in system mode.

## Pno

- Migliora le performance della virtualizzazione evitando l'emulazione
- Non richiede HW support specifico.

## Contro

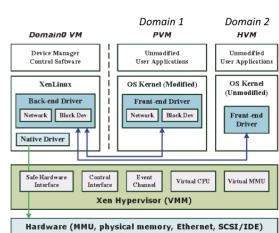
- Supporta solo versioni modificate degli os
- Alcuni closed os non sono supportati

## XEN

È il più popolare hypervisor basato sulla panavirtualization.

### Xen Architecture

- Xen è un very small kernel che isolato fin dall'inizio sul hardware
- Sopra a Xen, a livello di privilegio inferiore, abbiamo so-called domains
- Potrebbero esserci molti domini, e all'interno di ciascuno di essi possiamo eseguire un'intera OS con le proprie applicazioni
- Domains sono isolati fra di loro, e possono essere utilizzati per implementare le macchine virtuali
- Un dominio speciale, Dom 0, ha accesso all'API di Xen per creare e distruggere altri domini. All'interno di Dom 0 qualsiasi OS può essere installato, e strumenti specifici possono essere sviluppati per gestire gli altri domini
- I domini possono avere accesso diretto a certi dispositivi (I/O devices), oppure possono utilizzare dispositivi virtualizzati (fully virtualized devices), o possono utilizzare dispositivi paravirtuali (paravirtualized devices)



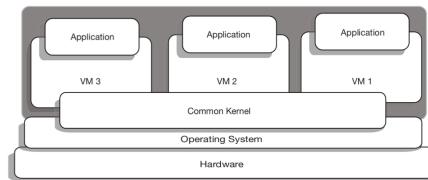
## Operating System Virtualization

L'obiettivo di questo approccio è di avere un lightweight approach che minimizza l'overhead dell'eseguire più VMs con lo stesso OS sullo stesso sistema.

L'hypervisor viene mosso, i virtual servers sono enabled dal kernel dell'OS della macchina fisica.

Il kernel dell'OS è condiviso tra i vari virtual server

Pro



- Virtualizzazione OS più leggera in overhead essendo che tutti i server condividono lo stesso kernel

• Un singolo sistema con la stessa quantità di risorse può supportare più VM

Contro

- Tutte le VM devono condividere lo stesso kernel
- Non tutti gli OS offrono OS virt. solutions

## Containers

Sono un modo per isolare un insieme di processi e farti pensare che sono gli unici in esecuzione sulla macchina.

La macchina che vedono può fornire un sottoinsieme delle risorse disponibili.

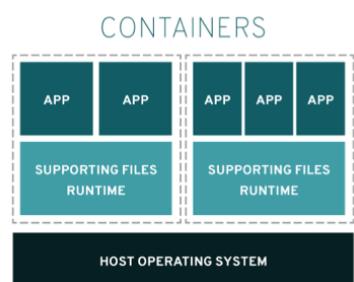
I container non sono VM, i processi all'interno del container sono normali processi dell'host kernel. Non puoi eseguire un S.O arbitrario all'interno del container, deve essere lo stesso dell'host.

Vantaggio: Non ci sono perdite di performance.

## Linux containers

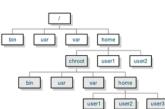
Sono i più usati e adottati nel caso dell'OS virtualization.

Sono implementati usando due kernel features:



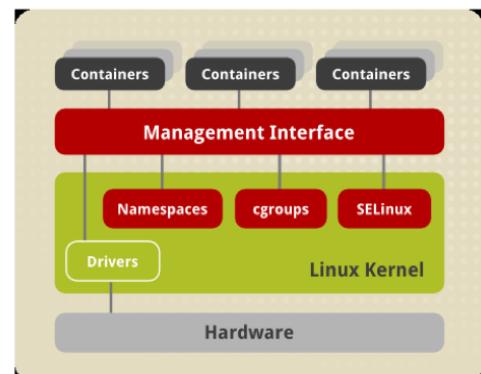
## 1) Namespaces

- Namespaces provide a means to segregate system resources so that they can be hidden from selected processes
- They are the extension of an old Unix function: `chroot()`, a system call that allows for a process to specify a portion of the file system, to which the application is confined. Using chroot a root user select a subdirectory (a subtree of the file system) that is make as the root file system of the process
- It was defined to confined untrusted process to a portion of the file system so they can access only the files they need
- This function worked for file system, namespaces have been introduced to hide or create other copies of other resources of the system
- Different namespaces are defined, e.g. network namespaces to hide network interfaces or to create virtual interfaces, pid namespaces to hide processes



## 2) Control Groups

- Namespaces are not sufficient to isolate a set of processes so they can not interfere with others
- Even if processes cannot interact with others, they can abuse of system resources, e.g. by allocating too much memory, by using too much CPU time or network bandwidth
- Control groups are groups of processes, for which the resource usage is controller and enforced
- They are created by root user, each process must belong to a group to which it cannot escape. When a process creates a child process the child inherits the control group of the parent
- Each group can be linked to one or more subsystems to limit access to system resources, for instance: `memory`, to limit the amount of RAM accessible to each group; `cput`, to limit the maximum fraction of CPU that each group can use



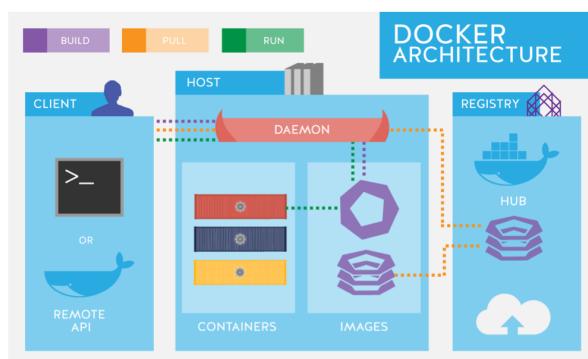
## Docker

3 container oltre ad essere usati nell' OS virtualization sono usati nel software distribution. Forniscono un pacchetto isolato nel quale possono essere inclusi software, librerie e file di configurazione. Dopo l'installazione del pacchetto docker, esso è pronto all'esecuzione.

### Docker architecture

Docker permette di istanziare container su un sistema sulla base di una immagine scaricata da una rete centralizzata, chiamata Docker Registry.

Il docker engine si occupa di scaricare e eseguire un container scaricato dal registry.



## Cloud Applications

Si fa riferimento con cloud applications alle applicazioni che sono sul cloud.

## Cloudification

Con il termine cloudification si fa riferimento all'operazione di muovere l'impl. di una applicazione sul cloud, trasferendo i dati e i servizi dall'HW fisico alle VMs sul cloud.  
È importante notare che questa operazione permette il trasferimento, ma per poter utilizzarne i vantaggi e capacita' del cloud computing abbiamo bisogno di ridisegnarla ed adattarla all'architettura delle cloud applications.

## Cloud Applications architecture

L'architettura delle cloud applications è basata su un multi-tier system, cioè le differenti funzionalità sono suddivise su differenti VM.



Il risultato è una catena di interazioni tra differenti server di differenti tiers.

I server sono connessi attraverso una LAN e usano un application protocol che può essere implementato attraverso un middleware software per semplificare lo sviluppo.

## Compute Cluster

L'insieme delle VMs che appartengono allo stesso tier è chiamato compute cluster.

Infatti, la stessa funzionalità di una applicazione può essere implementata su più VMs.

Le VMs nello stesso cluster non performano molte operazioni I/O, tranne per trasferire i dati tra esse.

Inoltre, solitamente usano un middleware software per gestire le comunicazioni.

Esistono tre tipologie di cluster:

- 1) High-availability Clusters
- 2) Load-Balancing Clusters
- 3) Compute Intensive clusters

### High-availability (HA) clusters

Sono cluster disegnati per essere fault tolerant e sempre disponibili.

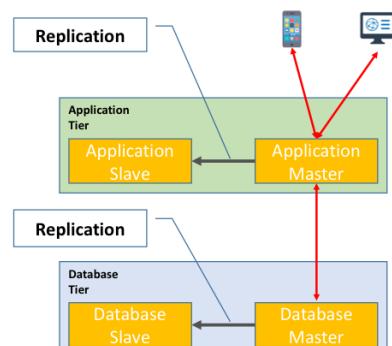
Viene usato il concetto di ridondanza, cioè ogni tier è implementato usando VMs multiple che possono performare le stesse operazioni e servizi.

#### Architettura

La ridondanza è gestita selezionando una VM come master.

Gli slaves sono configurati in modo da prendere il posto del master ogni qual volta esso fallisce.

Viene usata una election procedure per la selezione dello slave e una replication mechanism per replicare i dati.

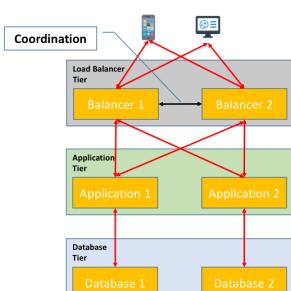


### Load-Balancing Clusters

Sono cluster disegnati per un alto utilizzo di risorse distribuendo il carico in modo bilanciato su tutti i nodi del cluster.

#### Architettura

Il load balancing è implementato introducendo un layer aggiuntivo chiamato "load balancer tier". Esso riceve le richieste e le invia all'applicativo.



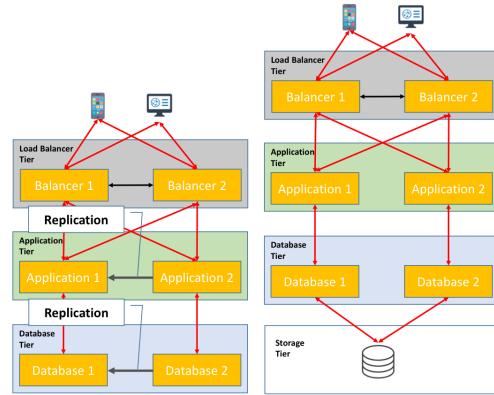
La VM che riceve la richiesta viene scelta dal load balancer tier seguendo una specifica politica.

### Data Synchronization

I dati devono essere sincronizzati fra le istanze dello stesso tier. Possono essere usate due possibilità:

- 1) Data Replication (introduce overhead)
- 2) Shared Storage tier

La scelta dipende dall'uso specifico e la quantità di dati da salvare.



Se viene coinvolto un grande dataset, viene usata solitamente la 2<sup>a</sup> opzione.

### Compute Intensive (ci) clusters

Sono cluster disegnati per l'analisi di grandi quantità di dati.

Essi adottano un appuccio "divide et impera":

- 1) I task di analisi viene diviso in sub-tasks (jobs)
- 2) I dati sono divisi in piccoli "chunks", ognuno assegnato a un job.
- 3) Un job e un chunk viene assegnato a un server.
- 4) Il server performa il job sul chunk e ritorna il risultato a un collector.
- 5) Il collector unisce i risultati.

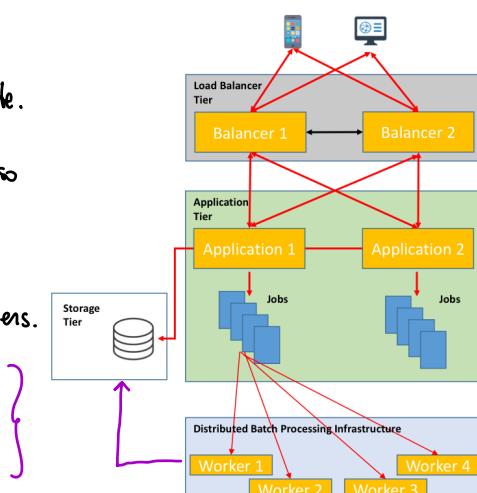
### Anchitettura

Viene usato un load balancer tier per gestire le richieste.

I dati iniziali vengono salvati in un cloud storage condiviso. Le istanze dell'app. tier dividono i dati in chunks e creano una lista di jobs. I jobs vengono assegnati ai workers.

Per ridurre il communication overhead, i workers possono avere accesso

diretto ai dati che devono essere analizzati.



## Cloud application design

Un software, facente parte di una cloud application, dovrebbe essere disegnato seguendo i seguenti requisiti:

- 1) Interoperabilità: Dovrebbe offrire un'interfaccia per interazione con gli altri componenti.
- 2) Scalabilità: Dovrebbe essere pronta per essere integrata con un numero crescente di componenti, dinamicamente e senza sforzo.
- 3) Componibilità: Dovrebbe permettere il suo utilizzo con differenti componenti, senza essere cambiata.

## Service-Oriented Architecture (SOA)

SOA è un approccio modulare per il design dei software objects.

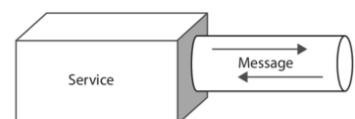
I componenti di un software, in SOA, sono riferiti come "servizi", i quali sono componenti autonomi che implementano specifiche funzionalità.

I servizi possono essere implementati usando differenti linguaggi di programmazione.

### Architettura

I servizi comunicano attraverso il passaggio di messaggi, i quali hanno uno specifico schema che definisce il formato e lo scambio di informazioni.

Il message schema definisce il servizio (cosa fa e quali info fornisce) e descrive i message exchange patterns per interagire con il servizio.



### Ruoli

Nelle interazioni all'interno di SOA possiamo contraddistinguere due ruoli:

- 1) Consumer service: Invia una service request al provider del servizio, attraverso un messaggio.
- 2) Provider service: Ritorna una risposta al consumer con il risultato.

Un servizio può ricoprire uno dei due ruoli, o entrambi.

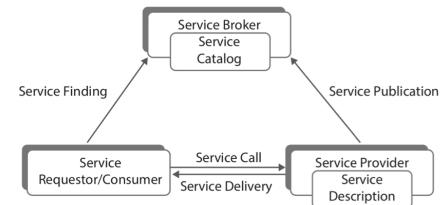
## Broker

Per poter effettuare la comunicazione tra service provider e consumer, il consumer dovrebbe conoscere l'indirizzo e il descrittore del provider. Questo è lo scopo del broker.

Il broker crea un service catalog (registry) dove tutti i servizi vengono listati.

I provider service possono essere utilizzati solo se pubblicati sul broker.

Dopo che il consumer ha richiesto al broker uno specifico servizio e ottenuto la sua identità, può interagire direttamente con il provider.



## Scalabilità

SOA services sono stateless e loosely coupled, di conseguenza sono perfettamente adattabili alla scalabilità dell'infrastruttura cloud.

È possibile infatti creare e deallocare istanze di un servizio in base al numero delle richieste.

## Vantaggi

SOA offre diversi vantaggi, oltre la scalabilità:

- 1) l'implementazione dei servizi rimane nascosta
- 2) la caratteristica del loose coupling permette di collegare e scollegare specifici servizi
- 3) Un sistema può essere assemblato usando una combinazione di servizi esistenti.
- 4) la comunicazione basata su messaggi permette a un modulo di trattare ogni transazione separatamente.

## Web Services

Sono una tra le implementazioni SOA più comuni.

Un webservice sono un software system disegnato per fornire un'interazione interoperabile da macchina a macchina attraverso il web.

Hanno bisogno di due elementi: URI e HTTP.

## Uniform Resource Identifier (URI)

Una URI è una stringa di caratteri usata per identificare una risorsa.

Una URI può essere:

- 1) Un nome: Uniform Resource Name (URN)
- 2) Un indirizzo: Uniform Resource locator (URL)

## Hypertext Transfer Protocol (HTTP)

È un application-level protocol per sistemi distribuiti, collaborativi e hypermedia info.

## Web service interaction

Un webservice usa due interface:

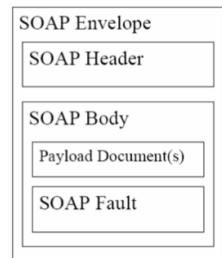
- 1) Web Services Description language (WSDL): Usata per l'interazione con il broker.
- 2) Simple Object Access Protocol (SOAP): Usata per l'interazione tra provider e consumer.

## SOAP

Fornisce una standard packaging structure per la trasmissione di documenti XML.

Un SOAP message consiste in un root element chiamato envelope che contiene un:

- 1) Header: Contiene info aggiuntive e security info.
- 2) Body: Contiene un payload, il quale include request/response info, e una fault section che include, se ci sono, errori e eccezioni.



Le SOAP messages sono trasmessi tramite HTTP e può usare sia GET che POST.

Con il GET la risposta è un SOAP message ma la richiesta non obbligatoriamente.

Con il POST sia la risposta che la richiesta sono messaggi SOAP.

## WSDL

WSDL descrive l'interfaccia e l'insieme di operazioni supportate da un Web service in un formato standard.

Inoltre, standardizza anche il modo nel quale i messaggi saranno trasferiti al/dal servizio.

Tramite la WSDL i client possono automaticamente capire:

- 1) Cosa può fare un servizio
- 2) Dove il servizio è locato
- 3) Come il servizio può essere invocato

Ogni volta che un client vuole invocare un WS sconosciuto, esso richiede e analizza il WSDL document, solo dopo questa op. il client invoca il servizio tramite SOAP.

Un WSDL message è un documento XML e comprende due sezioni:

- 1) Service Interface Definition : Definisce, in modo astratto, il servizio.

Esso comprende :

- a) Tipo dei dati (types)
- b) I messaggi scambiati tra requestor e provider (messages)
- c) Operazioni che possono essere fatte (PortType)

- 2) Service Implementation Definition: Definisce, in modo specifico, come il servizio è invocato

Esso comprende :

- 1) Come trasferire i messaggi (binding)
- 2) Come invocare il servizio (service)

## Universal Description, Discovery and Integration (UDDI)

Definisce un modo per pubblicare e scoprire info sui Web Service.

Usando l'uddi le aziende possono registrare un WS a un broker.

## REST

Il protocollo SOAP è complicato, essendo che richiede la creazione di una struttura XML. Dovuto a ciò è stato introdotto il REST (Representational State Transfer) come un'valida alternativa lightweight per la WS construction.

In un RESTful system, il client invia una richiesta tramite HTTP usando i metodi HTTP standard, la risposta del server include la rappresentazione della risposta.

Il contenuto dei dati è trasferito sempre tramite XML come parte dell'HTTP content, ma è rimossa la parte markup aggiuntiva richiesta dal SOAP.

### SOAP



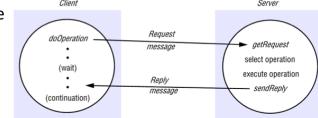
- Both SOAP and REST adopt a Request-Reply communication protocol

- It implements a synchronous communication:* the client blocks until the reply arrives from the server (time-coupled)

- It is reliable:* the reply is an acknowledgement of the reception of the request

- It is a direct communication:* the client interacts directly with the server without intermediaries (space coupling), which introduces additional overhead and complexity

### REST



## Space / Time Coupling

Request-response paradigm soddisfa i requisiti per la comunicazione tra i clients e il cloud frontend, tollerando sia lo space che il time coupling.

La comunicazione tra frontend e backend però è differente essendo che il backend sia aspetta di avere un numero vari di VM per supportare la scalabilità attraverso il load-balancing o supportando la disponibilità attraverso la data replication.

Lo space e time coupling non sono suited per assicurare la scalabilità nelle comm. all'interno del backend o tra frontend e backend.

Abbiamo quindi bisogno di una comunicazione indiretta per assicurare:

Space uncoupling: il sender non sa o non necessita di sapere l'identità del receiver, e viceversa.

→ Ottenuto tramite un intermediario senza coupling diretta tra sender e receiver

Time uncoupling: senders e receivers hanno lifetimes indipendenti.

→ Ottengo tramite l'intermediario che adatta un paradigma di comm. asincrona.

Il sender invia i mez all'intermediario che si occupa di consegnare il mez.

## Message Oriented Communication

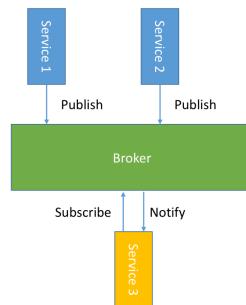
Il sender invia i mez in un message bus, implementato da un intermediario, il quale si occupa di inoltrare il mez al receiver e viceversa inviare, se c'è, una risposta al sender.

Tipi di intermediari:

1) Message Broker: implementa un publish-subscribe system

Tipi di subscription:

- **Channel-based**: publishers publish data to named channels and subscribers then subscribe to one of these named channels to receive all data sent to that channel
- **Topic-based**: each data has a set of fields, one of them denotes the topic of the data. Subscriptions are then defined in terms of the topic of interest. This approach is equivalent to channel-based approaches, with the difference that topics are implicitly defined in the case of channels but explicitly declared as one of the fields in topic-based approaches
- **Type-based**: subscriptions are defined in terms of types of events and matching is defined in terms of types or subtypes of the given filter



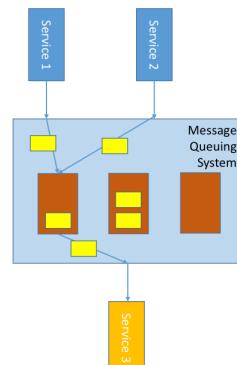
2) Message Queue: Fornisce uno scambio di messaggi point-to-point tra i producer e i consumer.

Il sender inserisce il mez nella coda e viene rimosso dal receiver.

All'interno del sistema, code differenti possono essere istanziate, un producer può inviare il mez in una coda specifica e il consumer può ricevere un mez da essa.

L'insieme di operazioni offerte dal message queuing system:

- Send, issued by the producer to place the message in the queue
- Blocking Receive, issued by the consumer to block until an appropriate message is available
- Non-blocking receive (polling operation), issued by the consumer to check the status of the queue, it will return a message if available or a not available indication otherwise
- Notify, issued by the message queuing system to notify a consumer when a message is available in its associated queue (the consumer has to subscribe first updates on a certain queue)



L'Advanced Message Queuing Protocol (AMQP) è una delle impl. più adottate per i Message Queuing Systems

Ogni messaggio consiste in:

1) Una destinazione: ID univoco che rapp. la queue di destinazione

2) Metadati associati al messaggio. Includono fields come la priorità del messaggio e la delivery mode.

Solitamente le code sono Fifo ma possono supportare la priority.

I consumer possono anche selezionare i mes in base alle proprietà dello stesso.

Persistenza: I messagg salvati nella coda saranno disponibili fin quando non consumati e verranno salvati anche sul disco per garantire la fornitura di essi.

### Message Transformation

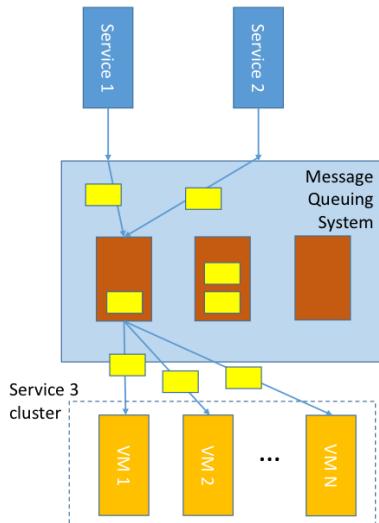
Sui messagg in arrivo possono essere applicate delle trasformazioni, ad esempio per formattare i mes in modo estogeno.

### Scalabilità

le message queues possono gestire il variazione in numero dei VMs per gestire la scalabilità del sistema.

le VMs che compongono un cluster ed impl. le stesse funzionalità possono ricevere i dati dalla stessa coda in accordo con qualche policy (es: load balancing o disponibilità).

le VMs possono essere create in qualsiasi momento e consumare i messaggi della queue.



### Implementation

Sia il message brokers che il Message Queuing System possono essere implementati sia in un modo centralizzato che distribuito.

1) Implementatione centralizzata: I broker/message queuing system è implementato su un singolo sistema (ad esempio una singola VM)

→ Problema: Pecca in scalabilità e resilienza perché c'è un singolo punto di fallimento.

2) Implementatione distribuita: Una rete di brokers / message queuing system

→ Più complesso essendo richiesta coordinatione e data replication, però garantisce resilienza e scalabilità

## Data Replication

Quando si effettua la data replication su differenti VMs, essa dovrebbe tenere in considerazione i seguenti requirements:

- 1) Replication transparency: Users dovrebbero accedere differenti copie dei dati senza accorgersene.
- 2) Consistency: le repliche dovrebbero essere consistenti con tutte le altre, una op. eseguita su una qualsiasi replica dovrebbe produrre gli stessi risultati.
- 3) Resiliency: Il sistema dovrebbe essere resiliente dalle VMs failures, anche se, è assunto che le network partitions cannot occur.

## System Model (tipo di cloud appl.)

I dati nel sistema sono una collezione di oggetti: (file, classi, insieme di records).

Ogni logical object è implementato come una collezione di copie fisiche, chiamate repliche.

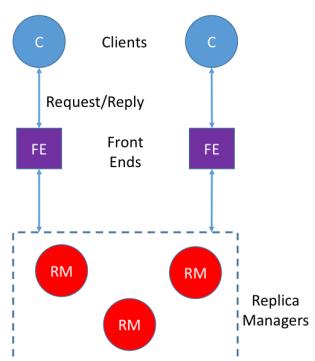
Ogni replica è salvata in un differente replica manager (una VM ad esempio) e accetta le update op. in un recoverable manner, così che le inconsistenti dovute ai fallimenti possano essere recuperate.

Per assicurare la consistenza, le repliche sono state machines, cioè oltre ai dati replicano anche dello stato aggiuntivo.

## Request Handling

Una richiesta può essere di due tipi:

- 1) Una query: Non modifica lo stato dell'oggetto.
- 2) Un update: Modifica lo stato dell'oggetto



In generale, i seguenti step sono implementati dal sistema per gestire una singola richiesta da parte del client:

- 1) Request: la richiesta è ricevuta dal frontend che assegna la richiesta a uno o più replica managers.
- 2) Coordination: gli replica managers si coordinano per preparare l'esecuzione della richiesta.  
Si possono coordinare per decidere se la richiesta può essere eseguita o deve essere diffusa in modo da mantenere la consistenza.
- 3) Execution: gli replica managers eseguono la richiesta. Essa viene solitamente eseguita per tentativi, cioè in un modo che possa esserne undone lateri se necessario.
- 4) Agreement: gli replica managers raggiungono il consenso sugli effetti della richiesta (se necessario).  
Se il consenso è raggiunto, l'esecuzione della richiesta viene committuta e i cambiamenti, se ce ne sono, applicati.
- 5) Response: Uno o più replica manager rispondono al frontend

Coordination e Agreement necessitano che gli replica managers si scambino messaggi per sincronizzarsi. Gli messaggi devono essere inviati a tutte le VM dello stesso ten con una comunicazione multicast.

Ci sono differenti multicast methods a seconda delle proprietà richieste:

- 1) Reliable / Atomic multicast: il messaggio o viene inviato a tutti, o nessuno. Non viene garantito l'ordine.
- 2) Total / Casual order multicast: ogni membro deve ricevere tutti gli updates nello stesso ordine.  
Non è però garantito un ordine tra i membri.
- 3) View-synchronous communication: Per ogni messaggio, tutti i membri hanno la stessa view ogni volta

Abbiamo visto i due replication models:

### 1) Primary Backup Replication

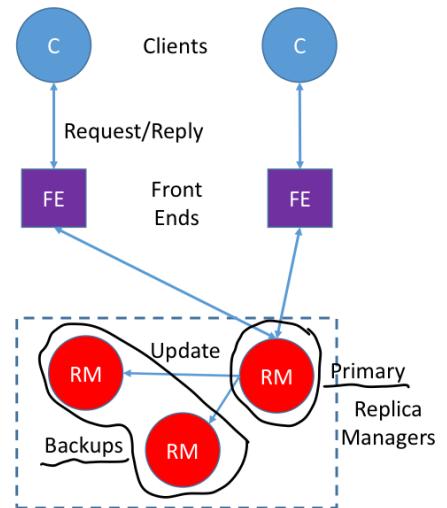
Questo modello viene usato per garantire la fault tolerance.

Solo un replica manager è selezionato come primary, mentre gli altri come backup.

I frontends comunicano solo con il primary, se esso fallisce ne viene selezionato uno tra i backup.

#### Request Handling

1. Request: the frontend issues a request, the request contains a unique identifier, which identifies univocally the request
2. Coordination: The primary handles requests atomically, in the order in which they have been received. It checks the unique identifier, if the request has been already executed the same response is sent back. No coordination with other replica managers is needed
3. Execution: The primary executes the request and stores the response
4. Agreement: If the request is an update, the primary sends the updated state, the response and the unique identifier to all the backups. The backups send an acknowledgment. For this kind of communication atomic multicast is sufficient
5. Response: the primary responds to the frontend, which hands back the response to the client



### 2) Active Replication

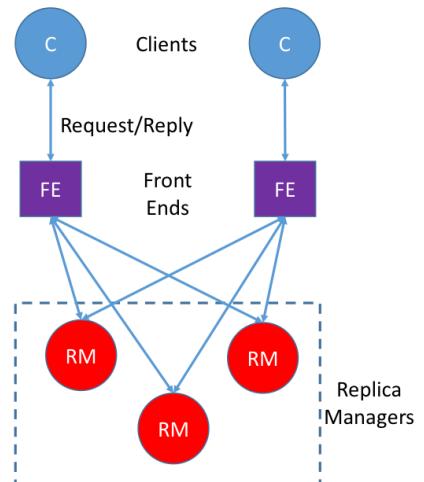
Replica managers sono organizzati in gruppi.

I frontends inviano le richieste a tutti i RM del gruppo.

Questo permette di non preoccuparsi di un possibile crash di un RM.

#### Request Handling

1. Request: The frontend attaches a unique identifier to the request and multicasts it to all the replica managers.
2. Coordination: No coordination is needed in this phase, the multicast method takes care of sending the request to all the replica managers. The method adopted has to be a total order multicast to ensure that requests are received in the same order by all the replica managers
3. Execution: Every replica manager executes the request. Since the requests are delivered in total order, replica managers process the request identically
4. Agreement: Thanks to the total order multicast, there is no need for agreement
5. Response: Each replica manager sends its response to the frontend. Considering that the multicast algorithm is a total ordered algorithm the frontend can forward the first response received.



Entrambi gli appross., 1 e 2, garantiscono high availability ma non scalability.

### 3) Gossip Architecture

Questo modello permette di assicurare high availability e scalability.

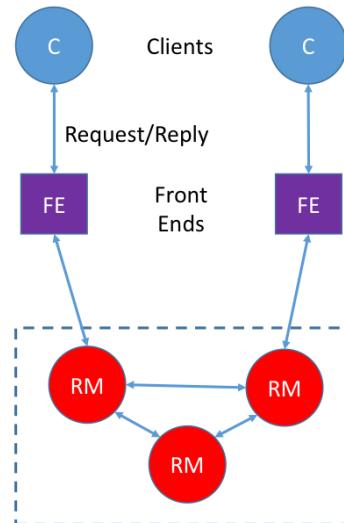
Per ottenere questa caratteristica, viene garantita la Sequential Consistency, cioè il client riceve info consistenti anche se non c'è detto siano le più recenti.

I replica managers si scambiano dei gossip messages periodicamente in modo da condividere gli update che hanno ricevuto dai clients.

#### Request Handling

1. Request: The front end normally sends requests to only a single replica manager at a time
2. Coordination: The replica manager that receives the request does not process it until it can apply the request according to the required ordering constraints. This may involve receiving updates from other replica managers, in gossip messages. No other coordination between replica managers is involved
3. Execution: The replica manager executes the request
4. Agreement: The replica managers update the data by exchanging gossip messages, which contain the most recent updates they have received. The updates are exchanged in a lazy fashion, i.e. gossip messages are exchanged only occasionally, after several updates have been collected, or after a certain amount of time
5. Response: If the request is a query, the replica manager replies after the execution, if it is an update it replies right after receiving it

l'invio dell'  
update non è dello  
stesso immediato

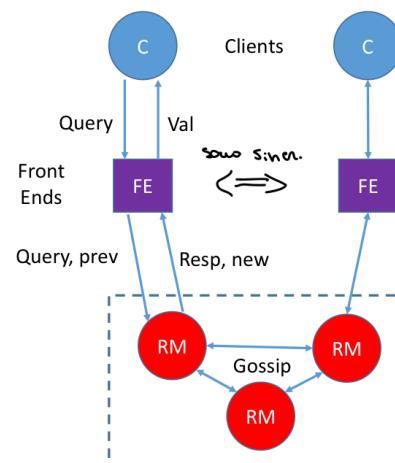


#### Timestamps

Per mantenere traccia dell'ordine delle richieste, vengono usati i timestamps. Per fare ciò, i FE devono essere sincronizzati.

Ogni FE mantiene un vettore di timestamp che riflette la versione dell'ultimo data value acceduto.

- 1) Il timestamp viene inviato al RM insieme alla query (p<sub>new</sub> field)
- 2) Il RM aggiunge un nuovo t.mestamp con la risposta. (new field)
- 3) Ogni timestamp ricevuto viene unito con il precedente timestamp value.



## Processing Query

- The timestamp associated with the query request reflects the latest version of the value that the frontend has read or submitted as update
- For this reason, every time a query is received, the replica manager has to return a value that is at least as recent as the value provided with the query
- For this reason if the timestamp of the latest local replica's update is oldest than the timestamp provided ( $valueTS < prev$ ) the request is deferred and kept on a list of pending operation, in order to wait for a missing update
- Once the query can be executed, the replica manager returns the  $valueTS$  of the last value update to the frontend

## Processing Updates

- When a replica manager receives an update request from a FE it first analyzes the request ID to check if the request has been received and processed already (the frontend uses the same ID each time)
- If the request is not duplicated, the replica manager creates a unique timestamp by incrementing the prev value received from the FE and then it sends back the response with the updated timestamp
- It also adds a new entry to the local log record of pending update operations
- If the same timestamp condition of the query is satisfied ( $prev \leq valueTS$ ) the update operation is committed, otherwise it is deferred until it is satisfied (every time a next gossip update is received the condition is verified)

## Global - Scale Applications

Anche se la grandezza dei cluster può aumentare, il massimo numero di richieste è limitato dai vincoli fisici della network infrastructure che trasmettono i dati dal client alle VMs.

Per mitigare questa bottleneck, le VMs del cluster vengono sparse in differenti datacenters. Il risultato è che il carico viene diviso sui differenti datacenters.

### Load balancing

Il load balancing può essere fatto a due livelli:

1) Global level: Su più datacenter

- Policies usate:
  - a) Nearest datacenter per minimizzare la latenza
  - b) Datacenter con il path meno usato

2) local level : Decidere quale VM server deve gestire la richiesta all'interno del datacenter.

### Global load balancing

Per gestire il global load balancing viene usato il DNS e i virtual IPs.

### Virtual IPs

Un virtual IP address (VIP) è un IP address che non è assegnato a un host o una rete ed è condiviso tra diversi dispositivi.

Un insieme di VIPs può essere assegnato a un'applicazione e ogni VIP (o sottointerme) può essere assegnato a una regione, in ogni regione più datacenter possono essere disponibili.

Il Regional DNS service è configurato con i Vips assegnati a quella regione per un device specifico.

Per ogni regione uno o più global load balancer è sviluppato. Ogni global load balancer ha assegnato un Vip dall'insieme di Vips assegnati alla regione.

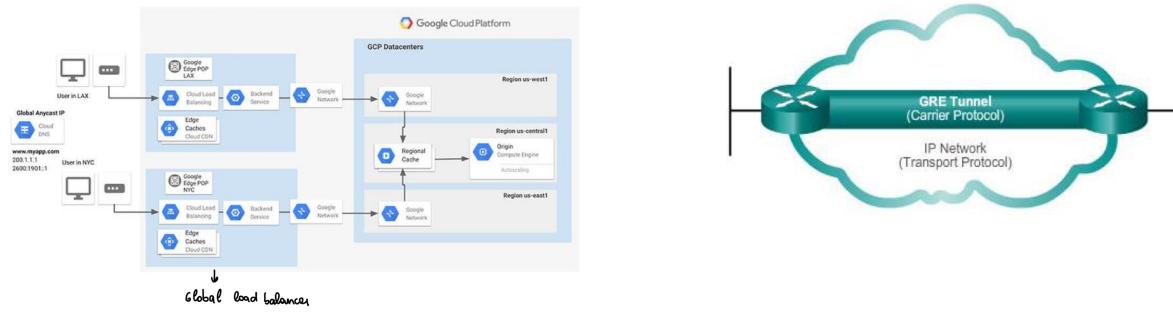
### Packet encapsulation

Il global load balancer è responsabile di ricevere le richieste e distribuirle al datacenter più conveniente in quel momento.

Questo viene fatto tramite la packet encapsulation, cioè un packet è incapsulato in un altro packet e inviato al datacenter selezionato.

A destinazione il pacchetto viene decapsulato e processato come se fosse ricevuto dal datacenter.

Generic Routing Encapsulation (GRE) è solitamente usato per stabilire un tunnel tra il global load balancer e il datacenter.



### Globally distributed system

In un sistema decentralizzato come i datacenter a livello globale c'è bisogno di un meccanismo di datareplication e synchronization, inoltre deve tollerare il network partition, cioè una certa regione <sup>o datacenter</sup> si può disconnettere per un certo periodo di tempo, ma deve continuare ad offrire il servizio.

Per questo motivo, tranne se la consistenza shetta è richiesta dall'applicazione, un

sistema di eventual consistency deve essere adottato.

Si tende quindi a preferire un BASE system rispetto a un Acid system.

Il BASE system ha alla base il concetto di eventual consistency, cioè lo stato del sistema e i suoi dati possono essere inconsistenti per un certo periodo di tempo, però, è garantito che la consistenza sarà raggiunta eventualmente in ritardo.

Di conseguenza, gli update saranno propagati asincronicamente usando uno specifico protocollo chiamato anti-entropy protocol e il controllo sarà invece restituito immediatamente al client.

### Inconsistent data

- 1) le query possono ritornare dati inconsistenti
- 2) c'è bisogno di un meccanismo per risolvere i conflitti, ad esempio: l'update con l'ultimo timestamp è accettato mentre gli altri scartati

### Vantaggi e Svantaggi

+ Quando una partition occurs nel global network, una regione o datacenter può continuare a operare.

Quando la partition è risolta, l'anti-entropy protocol distribuirà i dati per raggiungere la consistenza.

+ Response delay è minimo

- Questo sistema non è tollerabile per sistemi che non tollerano conflitti.

### Bayou architecture

È un eventual consistent architecture che fornisce data replication per high availability.

### Bayou operations

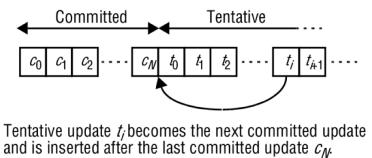
- 1) Quando un update viene ricevuto, viene initialmente applicato ma markato come "tentative"
- 2) Quando un update è tentativo, il sistema può undo e riapplicarlo se necessario per

raggiungere un consistent state.

Quando un update è committato, rimane applicato.

- 3) Lo stato del sistema in un certo momento è formato dai committed updates seguiti dai tentative updates.

- I committed updates sono ordinati nell'ordine dei commitments
- le tentative transactions sono ordinate basandosi sul timestamp



### Primary replica manager

Bayou adotta un primary commit scheme, cioè un RM è selezionato come primary e ha la responsabilità di committare gli updates.

La lista degli update viene scambiata attraverso l'anti-entropy protocol.

Se il primary diventa non disponibile:

- Gli update sono accettati nell'ordine che vengono ricevuti
- le query vengono risposte con lo stato corrente del sistema

Quando torna disponibile, committerà gli updates.

### Conflict detection e Resolution

Bayou adotta due meccanismi per rilevare e risolvere conflitti:

- A dependency check: a query and the expected result provided by the developer, to check if a set of pre-conditions are met for the execution of the update. Before applying the update, the replica manager executes the query. If the results differ with the ones provided by the developer a conflict is arisen
- A merge procedure, which is a procedure that is run when a conflict is detected. The merge procedure are provided by the developers

Entrambe le procedure sono deterministiche in modo che ogni RM risolve lo stesso conflitto nello stesso modo degli altri.

### Performance measurement

- Eventually consistent systems are widely adopted today, even though they don't provide safety, it is the proof that such systems can work in practice and has multiple advantages in terms of latency and availability
- A common metric to measure eventual consistency is time:
  - ↳ The window of inconsistency
    - ↳ The time required to have an information visible on the system
- Both of them are dependent on the anti-entropy protocol and in particular the anti-entropy rate, the frequency to which the anti-entropy protocol sends the update
- Given a certain anti-entropy rate the expected time to consistency can be calculated in order to obtain a certain Probabilistically Bounded Staleness (PBS)
- Many existing system allows to set some system parameters to obtain a certain PBS

## Cloud Platforms

Una cloud platform è un sistema distribuito che controlla gli hypervisors di tutti i server dell'infrastruttura.

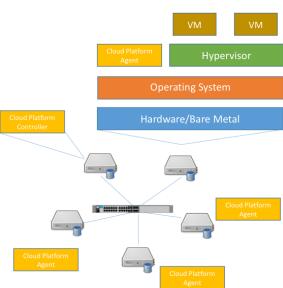
Essa gestisce la initializzazione / distruzione delle VMs nell'infrastruttura, cioè in ogni momento controlla quali VMs devono essere eseguite su quale server e con quali proprietà.

Per la gestione delle VMs viene offerta un'interfaccia che può essere usata dagli umani.

### Architettura generale

#### General Architecture

- The overall architecture is simple: a set of servers are connected to a high speed local area network (Ethernet LAN > 10Gbps)
- Each server has an operating system installed on top of the bare metal (the physical hardware)
- A hypervisor is installed on the OS to virtualize the physical resources offered by the servers
- In addition to the hypervisor a cloud platform software is installed to control the local resources virtualized by the hypervisor. This software is often referred as the agent of the cloud platform
- In order to implement system wide management and control functionalities of the cloud platform on one (or more to ensure resiliency) server it is installed a particular instance of the cloud platform software referred as cloud platform controller. This software can be installed on a server that does not have a hypervisor or it could be even installed on a virtual machine. The software is responsible for controlling the overall platform communicating with the agents



OpenStack è una software platform opensource per IaaS cloud computing.

È la principale soluzione adottata per il private cloud computing.

### OpenStack platform

La piattaforma software è installata su ogni server ed è eseguito "in testa" all'host operating system.

Essa può essere eseguita solo sul linux os mentre supporta hypervisor technologies multiple.

Attraverso ogni hypervisor, OpenStack crea/distrugge le VMs.

Ogni server che ha installato OpenStack è chiamato **openstack node**.

La piattaforma inoltre gestisce la connessione delle VMs alle virtual networks e lo storage.

### OpenStack instance

L'insieme degli openstack nodes formano una openstack instance la quale combina le risorse computazionali, di memoria e storage per creare le VMs.

Ogni istanza è composta da:

- 1) Almeno un **controller** il quale si occupa di coordinare le openstack functions e gestire le risorse disponibili dell'istanza.
- 2) Altri compute nodes che offrono le risorse computationali e di storage per eseguire le VMs.  
I nodi sono connessi attraverso una local network, la quale è usata come:
  - Una infrastruttura sulla quale le virtual networks sono create
  - Communication networks usata dalle Openstack components per comunicare per coordinatione e gestione.

### Openstack architecture

Le funzionalità della piattaforma sono raggruppate in servizi, ognuno dei quali è implementato come un modulo differente e separato. Ogni modulo può lavorare separatamente dalla piattaforma ed è sviluppato e mantenuto come un progetto separato.

Ogni service espone un insieme di REST API, attraverso il quale può ricevere le richieste via web/command line e dagli altri service modules.

Le operazioni di coordinatione e le info di stato tra i servizi vengono invece scambiate usando un message queue system.

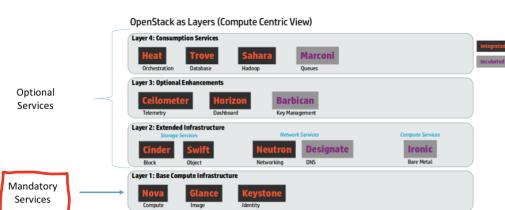
La piattaforma espone:

- 1) Una web dashboard che permette a utenti e amministratori di gestire le risorse virtuali.
- 2) Una interfaccia REST APIs che può essere usata dai programmi esterni.
- 3) Una command line interface

### Servizi openStack

I servizi sono divisi in due categorie:

- 1) Core services: Obbligatori in ogni installazione
- 2) Optional services: Installati solo se le funzionalità fornite da quelle servizio sono necessarie



g servizi possono essere installati sul controller node o sui compute nodes.

In alcuni casi, i servizi sono installati su entrambi con differenti configurazioni.

**OSS:** Tutti i servizi nel control node sfruttano alcuni supporting services, come ad esempio un DB service (es: Mysql) o un Message broker service (es: Rabbit MQ).

## Information Exchange / communication

g servizi possono comunicare tra loro in 3 modi differenti:

- 1) Message broker : Scambiano i messaggi tramite il message broker
- 2) REST API : Un servizio invoca la REST API di un altro servizio
- 3) Database : Scambiano info tramite il DB usando una conn. TCP/IP

## Key Stone

Keystone è il componente che si occupa dell' identity management.

Si occupa di gestire le autenticazioni e le autorizzazioni agli oggetti ai differenti utenti.

Gli oggetti sono solitamente raggruppati in progetti e le autorizzazioni per un certo utente può essere data all'intero progetto.

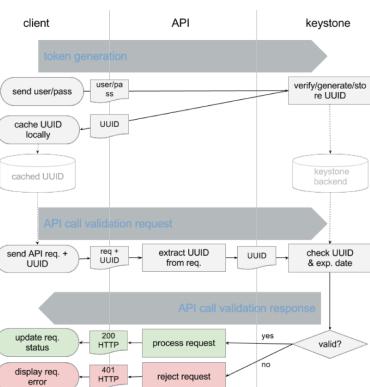
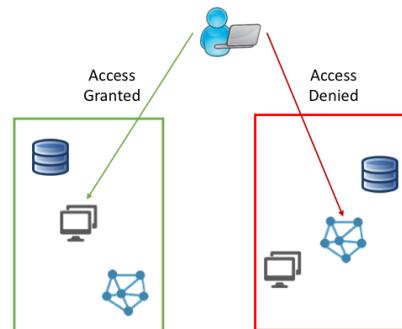
**OSS:** Keystone è solitamente installato sul Controller Node.

## Implementation

Keystone implementa una token-based authentication.

- 1) Un utente interagisce con il Keystone attraverso una autenticazione basata su username e password.
- 2) Se l'autenticazione avviene con successo  $\Rightarrow$  Viene ricevuto un token
- 3) Il token è usato per accedere ai servizi di OpenStack

**OSS:** Oggi servizio si occupa di validare il token!



## Nova

Nova è il componente che si occupa della creazione e gestione delle VMs.

Nova si interfaccia con gli hypervisori esistenti per la gestione delle VMs.

Nova include due differenti moduli:

- 1) Nova controller: Installato sul controller e responsabile della gestione delle richieste dagli utenti e delle risorse disponibili sui compute nodes.

Nello specifico colleziona le richieste di creazione e distruzione delle VMs, valuta le azioni corrispondenti che devono essere eseguite e invia i comandi al compute node.

È composta dai seguenti sub-components:

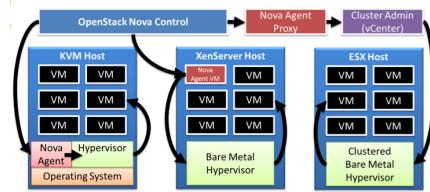
- Nova - API: Interfaccia dove arrivano le richieste
- Nova - Conductor: Gestisce tutte le operazioni
- Nova - Scheduler: Valuta il piazzamento appropriato di una nuova VM rispetto allo stato dei compute nodes e le risorse disponibili
- Nova - network: Implementa i networking services di base per le VM o interface
- Nova - Consoleauth e Nova - Novncproxy: Implementa un console service attraverso il quale gli utenti possono connettersi alle VM.

comunicano attraverso la messaging queue

- 2) Nova agent: Installato sui compute nodes e responsabile di ricevere le istruzioni dal controller module e tradurle nei comandi corrispondenti per l'hypervisor.

È composto solo dal compute service il quale riceve i comandi dal controller e inizializza / termina le VMs instances interagendo con l'hypervisor.

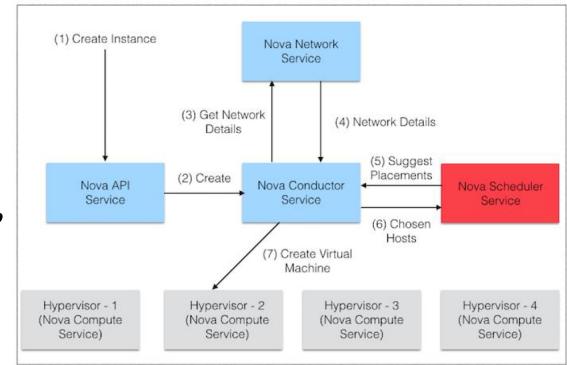
Ogni differente hypervisor ha un driver il quale espone un'interfaccia comune usata dal nova - compute agent per interagire con l'hypervisor.



## Nova - VM creation Workflow

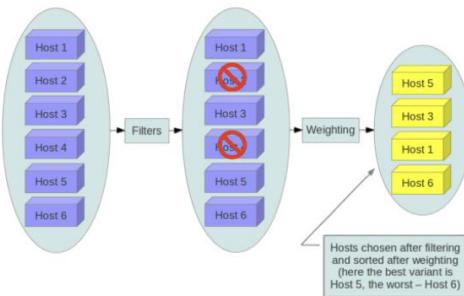
Quando la richiesta per la creazione di una nuova VM è fatta da un utente, i nova components fanno i seguenti steps:

- 1) Nova-API riceve le richieste che vengono inviate al Nova-conductor.
- 2) Nova-conductor invia la richiesta al Nova-scheduler, il quale suggerisce un piazzamento appropriato per una nuova VM.
- 3) Se il Nova-scheduler permette la creazione della VM, il Nova-conductor invia la richiesta al Nova-Network service per instantiare la giusta network configuration.
- 4) Quando il Network è propriamente configurato, la richiesta è inviata al Nova-Compute service del nodo selezionato dal Nova-scheduler.
- 5) Il local Nova-compute interagisce con l'hypervisor per creare la VM.



## Nova scheduling Strategy and overcommitment

- The scheduling policy can be customized to accommodate different requirements
- The scheduler performs an initial filtering phase in which not suitable hosts are removed (they don't have a specific characteristic like a passthrough-device or they are out of resources), then an ordering of the hosts
- The policy can be configured to include certain degree of over-commitment for RAM and CPU, i.e. more VCPU than the CPU available in a host are allocated (the CPU will go slower), more RAM than the one available in the system is allocated to VMs (the host will swap)



## Glance

Glance è il componente che si occupa della gestione delle immagini.

Ogni VM è istanziata da una immagine la quale include uno specifico OS pre installato e pre-configure.

Glance è composto da due sub-components:

1) Glance : Per la gestione delle immagini

Glance è responsabile di esporre una REST API per gestire le immagini e salvare la lista delle immagini disponibili e le sue features sul DB.

2) Glance Storage : Per la gestione dello storage

Glance Storage è responsabile di salvare le immagini. Supporta differenti storage options usando drivers differenti, tra loro:

- Il local file system del nodo sul quale il servizio è installato
- Un cloud file system, cioè un file system distribuito o un cloud storage

## Neutron

Neutron è il componente che si occupa della gestione del network.

Le VMs richiedono un virtual network per comunicare fra loro e vengono creati differenti virtual networks per la comunicazione tra differenti VMs per assicurare l'isolamento.

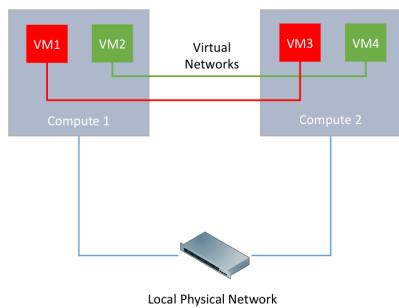
Neutron è il servizio responsabile di gestire la virtual network infrastructure che viene creata usando la local Physical Network.

Neutron è diviso in due sub-components:

1) Neutron Server: è installato sul controller ed è responsabile di gestire i virtual networks, coordinando i neutron-agents.

È anche responsabile di esporre le REST APIs per la gestione delle Virtual Networks agli utenti.

2) Neutron Agent: è responsabile di gestire il traffico da/alle VMs che sono sui compute nodes. Specificatamente, spedisce il traffico diretto/generato



permette diverse config.,  
 ad esempio l'uso  
 di OpenVSwitch nelle  
 ultime versioni.

alla local VMs per emulare differenti virtual networks che attraversano differenti compute nodes.

I dati vengono inviati usando il local physical network.

L'agent deve imporre l'isolamento tra Virtual Networks differenti.

### Come connettere le virtual network a internet

le virtual network sono solitamente reti private.

Per permettere la connessione delle Virtual Networks a internet, Neutron deve initializzare un Network node.

Un Network node è un nodo (solitamente il controller node) che è connesso a una rete esterna (attraverso internet) oltre a quella locale che connette gli openstack nodes.

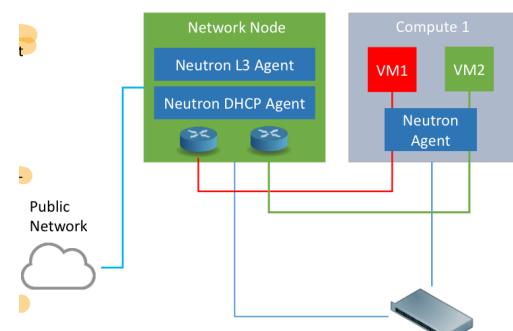
Questo nodo esegue un agente particolare, chiamato Neutron L3-agent, che è responsabile per reindirizzare il traffico da/alle private Virtual Networks da/alle public networks.

Questa funzionalità è implementata instantiando virtual routers, i quali sono responsabili per implementare traffic routing e NAT functionalities.

Oltre a questo, il Network node hosta un modulo chiamato Neutron-DHCP-agent, il quale è responsabile della gestione dell'assegnazione delle configurazioni di rete alle VMs.

### Physical Network Infrastructure

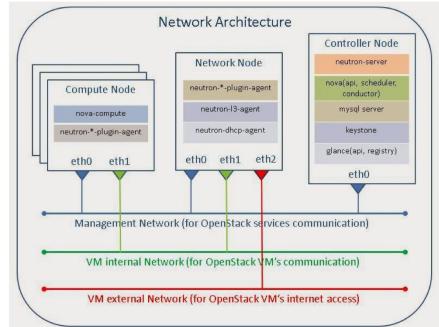
Una tipica Openstack network infrastructure connessa a una rete esterna, è configurata con tre differenti physical networks:



- Un management network = Usato per la openStack service communication
- Un VM internal network = Usato per distribuirne le comunicazioni interne tra le differenti VMs
- Un external network = Usato per l'accesso a internet

### Osservazioni :

- 1) Le management network e il VM internal network possono essere implementati sulla stessa physical LAN
- 2) Solo il Network node può essere connesso alla rete esterna
- 3) Il controller node può essere connesso solo al management network



### OpenVSwitch

OpenVSwitch (OVS) è una impl. open-source di un distributed network virtual switch.

È specificatamente disegnato per fornire uno switching stack per gli hardware virtualization envs. Esso può essere eseguito direttamente all'interno (o in combinazione) dell'hypervisor per gestire il traffico delle/alle VMs.

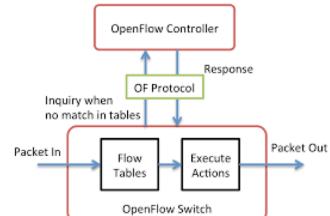
Supporta l'impl. di : Packet filtering, routing, switching e manipulation functionalities.

### Software Defined Networking (SDN)

OVS adotta il SDN paradigm, il quale è un approccio centralizzato nel quale ogni nodo ha la sua OVS instance la quale è controllata da un controller. Seguendo le direttive del controller, tutte le OVS instances implementano un distributed virtual switch.

### Caratteristiche:

- 1) le network rules sono configurate dinamicamente su ogni istanza basandosi sulle direttive provenienti dal controller.
- 2) Ogni OVS node ha una Tabella (flow table) che descrive le features di ogni incoming packet e la corrispondente azione da fare.
- 3) Ogni volta che un pacchetto viene ricevuto, la tabella viene acceduta  $\Rightarrow$  se il pacchetto ha un match con una entry della tabella allora l'azione viene eseguita. Altrimenti, OVS manda un query message al controller, il quale risponde con una nuova regola da essere aggiunta nella local table o una specifica azione da essere eseguita una volta.

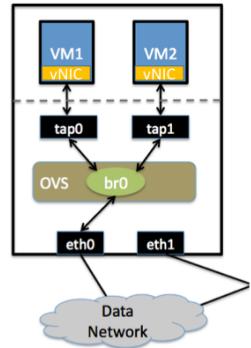


## VM - OVS communication

Ogni VM ha una Virtual Nic ed è linkata a un'altra virtual Nic creata nell' host OS , ad esempio Tap0, Tap1 .

Un Tap è una Virtual network interface che può essere linkata con un software, ad esempio l' hypervisor in questo caso, per ricevere e inviare network traffic (layer 2 data frames) to/from the software.

Il traffico dalle Tap interfaces è gestito da OVS che implementa le routing and forwarding functionalities.



## OVS - Network node

Per implementare le L3-routing functionalities e data dispatching , una OVS instance è installata anche sul network node.

Le funzionalità sono per instanciare il traffico a /da l' external network alle VMs.

## Network Virtualization

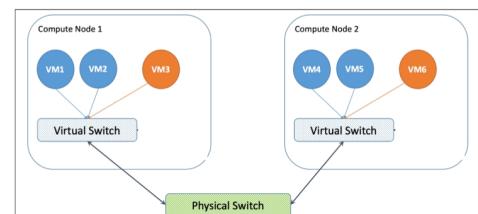
Per network virtualization si intende il processo attraverso il quale le Virtual Networks sono create sulla stessa physical infrastructure.

In Openstack, Network virtualization richiede che il traffico sia inoltrato attraverso Compute nodes differenti e to/from il network node according to Virtual network existence.

Con questo obiettivo, ogni pacchetto dovrrebbe essere marcato con il connetto ID della VN alla quale appartiene , in modo da essere correttamente distribuito.

Possono essere usate due differenti tecniche :

- Packet Tagging
- Tunneling



## Neutron VLAN operations

Neutron può essere configurato per usare la VLAN per creare le VN.

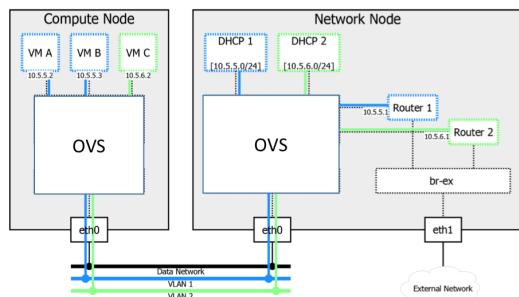
Quando una VN viene creata, gli viene assegnato un VID da parte del Nova-Server.

Quest'ultimo configura la OVS dei compute nodes coinvolti conseguentemente, in modo da taggare il traffico proveniente da una VM con il VID assegnato alle sua VN e inoltrare i pacchetti a tutti i compute nodes coinvolti.

Nel caso la VN fosse connessa a internet via Virtual Router, il Neutron Server configura anche il OVS eseguito sul Network node per taggare i pacchetti provenienti from/directed to il Router assegnato alla VN.

VLAN è un tipo di Network virtualization.

Aggiunge un nuovo field all'Ethernet header chiamato VLAN ID (VID) che specifica a qual virtual LAN appartiene.



## VXLAN

VLAN ha un limite dovuto agli ID di max 12 bit che impone un n. max di VLAN a 4096.

VXLAN permette di avere fino a 16 milioni di VN differenti.

È una packet encapsulation methodology tra due endpoints, chiamato Virtual tunnel endpoints (VTE). I frames vengono incapsulati in UDP packets e viene incluso un VXLAN header per riportare l'ID delle VLAN essendo che due VTEs possono supportare più VLANs allo stesso momento.

## Operations

- 1) Per ogni VN un nuovo VXLAN ID è istanziato dal Nova-server.
- 2) Il Nova-server istanzia le OVS instances di incapsulazione i pacchetti nei VXLANs tunnels in accordo con le VN instantiation.

## GRE Tunneling

Un altro virtualization mechanism adottato è il packet tunneling via GRE protocol.

GRE è un tunneling protocol che permette di incapsulare network IP packet all'interno di altri IP packets.

Il protocollo è usato per creare un tunnel tra due internet hosts. Inoltre aggiunge un GRE header aggiuntivo per specificare il tipo dell'encapsulated protocol.

Per creare una VN tramite GRE, viene creato un GRE tunnel per ogni paio di compute nodes (e tra il compute node e il network node) che eseguono VMs appartenenti a una specifica VN.

Vengono usati differenti GRE tunnel IAs per ogni VN il quale viene creato dal Nova-server.

Quest'ultimo, istruisce le varie instances di creare un VN adapter (GRE-1) sopra la physical interface.

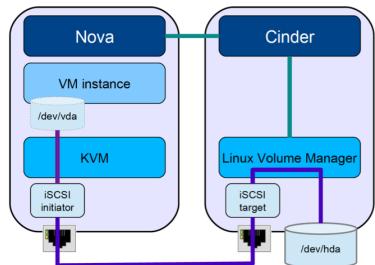
La GRE interface è responsabile di creare i tunnel tra i compute nodes coinvolti e incapsulare/decapsulare il traffico.

## Cinder

Cinder è il componente che si occupa della gestione dei volumi.

Ogni VM ha un default virtual harddrive il quale mantiene l'OS.

Se la VM richiede più spazio, volume aggiuntivo può essere dinamicamente creato e associato a una istanza.

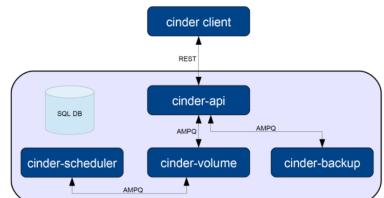


Un virtual Harddrive è esposto dall'hypervisor, il quale accede l'attuale virtual harddrive tramite il iSCSI protocol.

iSCSI è un protocollo che fornisce block-level access allo storage.

Cinder è composto dai seguenti componenti:

- 1) Cinder - API : Espone un'interfaccia REST ai client per controllare le cinder operations.
- 2) Cinder - Volume: Responsabile di gestire direttamente le richieste dalla REST interface
- 3) Cinder - scheduler: Responsabile di selezionare il giusto storage per i nuovi drivers
- 4) Cinder - backup: Responsabile di creare backup di volumi esistenti quando richiesto dagli utenti.



## Ceilometer

È un componente di monitoraggio.

Essa monitora tutti i componenti dell'istanza, misurando le risorse usate da ogni utente e le statistiche.

## Anchitecture

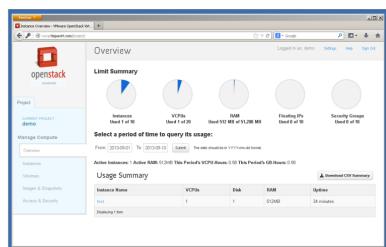
Ceilometer ha un Ceilometer - collector centralizzato che riceve tutti i dati dagli Openstack components e li salva in un DB.

Per collezionare i dati dai compute node, essi hanno installato un Ceilometer - Agent. Esso è responsabile di collezionare tutte le notifiche dai componenti locali e inoltrarle al collector.

Eventualmente il collector espone un insieme di REST APIs per fornire i dati dal DB.

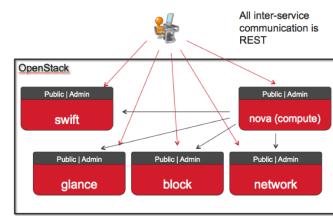
## Horizon

- OpenStack functionalities are exposed to Users through a web interface
- The dashboard is usually exposed by the controller
- It allows management of all the instances aspects
- A set of command line tools are also included for backend management



## Service APIs

- Every OpenStack service exposes a set of APIs
- All APIs communication is REST
- APIs are exposed by each service for inter-service interaction and to expose a set of functionalities to Users
- APIs can be exploited by Users to embed automation process in external applications



## Swift

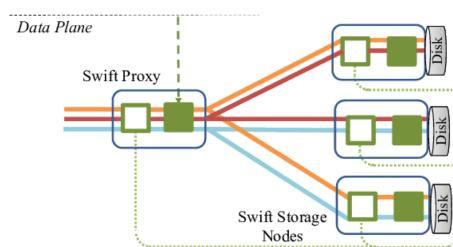
Swift è l'object storage service disponibile in Openstack.

Esso permette agli utenti e gli altri moduli di openstack di salvare i dati nella piattaforma cloud.

Gli altri servizi, come Cinder e Glance, possono usare swift per salvare i Volumes backup e le immagini, rispettivamente. Swift è responsabile di gestire tutti gli aspetti del data storage, dal ricevere e salvare i dati, al fornirli.

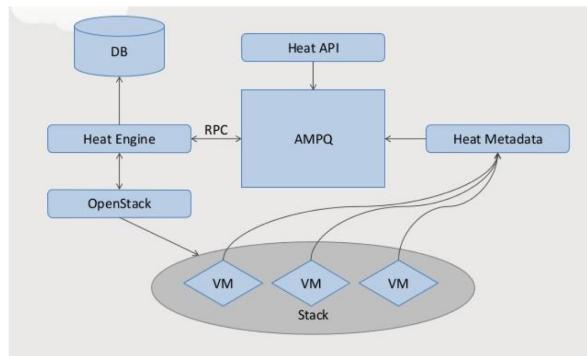
Si basa su due componenti:

- 1) Swift proxy: responsabile di esporre la Rest interface e gestire le richieste
- 2) Swift-Storage-Node: installato su ogni storage node, per salvare i dati sul physical drive.



## Heat

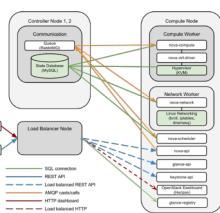
- Heat is the OpenStack orchestrator
- It can manage VMs creation/destruction or their settings automatically
- This automation is based on a set of rules that specify when certain conditions to trigger actions on the VM
- Through heat, for instance, an autoscaling service can be created that exploit the status data from ceilometer to create/destroy VMs according to the status of the system
- The component is composed of the following modules:
  - Heat-API that exposes the interface for the users to configure the orchestrator
  - Heat-Engine that is responsible for handling user request and implementing them



## Availability

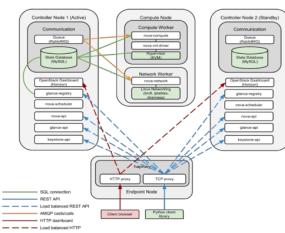
### High Availability (HA)

- The current architecture includes only one instance of the managing components
- OpenStack can be installed in the so-called high availability configuration, i.e. multiple instances of each service can be deployed in order to ensure high availability and resiliency
- This can be performed in a similar manner cloud applications are deployed in a redundant manner to ensure high availability
- In the high availability configuration, multiple instances of controller nodes are deployed with all the corresponding services to control the same set of computing nodes



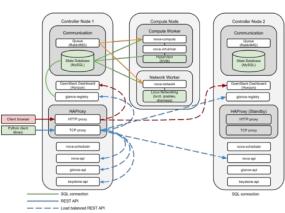
### HA with dedicated node

- Overall OpenStack APIs are exposed not directly by the controller node but through a HA proxy
- The HA proxy is responsible for receiving the requests and dispatch them to one of the controller nodes
- The proxy must implement functionalities to detect when Controller node failures occur to remove failing controller node to the list of active nodes
- The State storage must be replicated across the controller nodes



### HA with simple redundancy

- Overall OpenStack APIs are exposed directly by one controller, selected as master controller
- The functionalities of the HA proxy are implemented on all the controller, one of them is marked as active and serves requests, while the others are marked as inactive
- The HA proxy instances must coordinate to detect failures and trigger reconfiguration when needed
- Again, the State storage must be replicated across the controller nodes



## Cloud Platforms software development

Negli anni precedenti, le applicazioni erano grandi monoliti. Essi scalavano verticalmente ed erano/sono chiamati legacy systems.

Il cloud computing cambio radicalmente il modo in cui erano sviluppate le applicazioni, andando a dividere i monolitici in componenti indipendenti chiamati servizi/microservizi.

I servizi comunicano tra loro attraverso protocolli sincroni (SOAP, REST) o asincroni (Message queue).

## Application Development process

In passato, il ciclo di vita di un'applicazione era gestito da due team differenti e separati:

- 1) Development team (dev team): Design e program l'application software
- 2) Operations team (ops team): Deploy e maintain l'application software

Ora le organizzazioni adottano un **DevOps approach**, cioè lo stesso team sviluppa e mantiene l'app.

**Vantaggio:** Developers sono più coinvolti durante la produzione dell'applicazione.

**Problema:** Implementation difficile perché i membri del dev team è difficile abbiano il background e esperienza per gestire l'infrastruttura

Una soluzione ideale sarebbe mettere in pratica un **NoOps approach**, nel quale i developers fanno il deploy dell'applicazione senza doversi preoccupare dell'infrastruttura, la quale viene gestita da un team al di fuori dell'azienda.

## Service Deployment and management

Gestire una applicazione con un gran numero di componenti può essere complicato, dovuto al fatto che la complessità delle operazioni di deploy aumenta.

Una soluzione è automatizzare il deploy e la config dei componenti.

## Service Dependencies

I servizi sono sviluppati in modo separato da team differenti.

Dovuto a ciò, team differenti possono scegliere di usare librerie differenti, o uguali ma con differente versione.

In più ogni team vuole avere la libertà di riempire una libreria ogni volta necessario.

Queste caratteristiche possono essere complicate da gestire per un ops team.

## Consistent application Environment

Uno dei problemi più grossi che il dev e ops team devono gestire è la differenza di env nella quale eseguono le applicazioni.

Un modo per semplificare questa op. è di effettuare il deploy delle applicazioni in un env dove tutte

le librerie sono pre-installate e l' HW è astatto.

1) Full virtualization: Questo env può essere creato tramite la full-virtualization

Problema: Overhead in termini di risorse e config delle VM con le gruste librerie.

2) Lightweight Virtualization: Uso dei container, i quali possono essere usati per preparare un env virtuale nel quale librerie e dipendenze sono pre-installate e configurate per una certa application.

In più i container possono essere facilmente spediti e installati su machine diverse.

## Kubernetes

Kubernetes (sviluppato da google) permette di fare il deploy in modo semplice e gestire containerized applications su di se. Il suo scopo è semplificare il deploy e gestione dei container dell'app. prendendosi cura degli aspetti riguardanti scalabilità, failure tolerance, resource management ...

Una containerized application è una applicazione i quali componenti (servizi) sono sviluppati in containers.

L'ops team è comunque necessario con la responsabilità di mantenere e eseguire la Kubernetes infrastructure.

**Obiettivo:** Offrire una lightweight IaaS infrastructure per fare il deploy delle applicazioni in un modo semplice e effettivo, così i dev possono occuparsi del deploy senza dover gestire troppi aspetti

## Container operations

Kubernetes è una piattaforma distribuita composta da un master node e un vario numero di worker nodes.

Il master node espone un'interfaccia ai dev con la quale possono richiedere di eseguire appl. specifiche.

Il dev deve submittare un app descriptor, il quale include la lista dei componenti dell'app e la loro configuration.

in modo automatico

Kubernetes master si occupa di fare il deploy dei componenti sui worker nodes, basandosi sulla definizione fornita dal dev.

Tramite l'interfaccia il dev può specificare:

- 1) Quali servizi devono essere eseguiti insieme (sullo stesso working node) → i restanti vengono distribuiti tra i vari working node in base al loro status
- 2) Numero desiderato di istante (per la scalabilità) → Kubernetes si occupa di replicare i componenti e fare load-balance del traffico fra loro. Quelche si occupa di monitorare il loro stato e gestire i fallimenti (creando nuove istante se necessario).

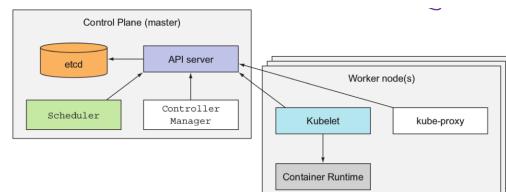
Vantaggi:

- 1) Kubernetes si occupa di implementare le funzionalità come scaling, load-balancing e self-healing
- 2) Dev si possono concentrare su sviluppare le applicazioni senza perdere tempo cercando di capire come integrare l'applicazione con l'infrastruttura
- 3) I compiti dell'ops team sono semplificati: Deploy components, configuarli, sistemare i failure sono gestiti dalla piattaforma stessa
- 4) Kubernetes può ottenere un miglior utilizzo delle risorse disponibili nell'infrastruttura

### Kubernetes Cluster Architecture

Il master node implementa il control plane, il quale è responsabile di controllare il cluster.

Ogni componente del control plane può essere installato su:



- Un singolo master node

- Splitto su più nodi

- Replicato in modo da assicurare alta disponibilità

Componenti control plane:

- 1) API server: Espone l'interfaccia per i dev
- 2) Scheduler: Schedula le app
- 3) Controller manager: Fa cluster-level functions (es: replicare componenti e gestire failure)
- 4) Etcd: Database distribuito affidabile

## Componenti: worker nodes

- 1) Container runtime: Un container runtime env (tipo docker) che si occupa di eseguire i container
- 2) Kubelet: Il quale "parla" con l'API server e gestisce i container sul nodo
- 3) Kubernetes Service Proxy: Il quale si occupa di bilanciare il traffico di rete tra i componenti dell'applicazione e inoltrare le richieste, mentre i container sono migrati tra i vari working nodes.

## Kubernetes Application Deployment

L'application description (data dai dev) lista i container che compongono l'applicazione.

I container sono raggruppati in pods (i quali possono essere dei container).

Un pod è un gruppo di containers che devono essere eseguiti sugli stessi worker nodes, cioè deployed insieme e non dovrebbero essere isolati.

Per ogni pod o singolo container il dev specifica la configurazione e il numero di repliche.

Dopo aver submittato il descriptor al control plane,

lo scheduler schedula il numero specificato di repliche dei pods agli worker nodes disponibili.

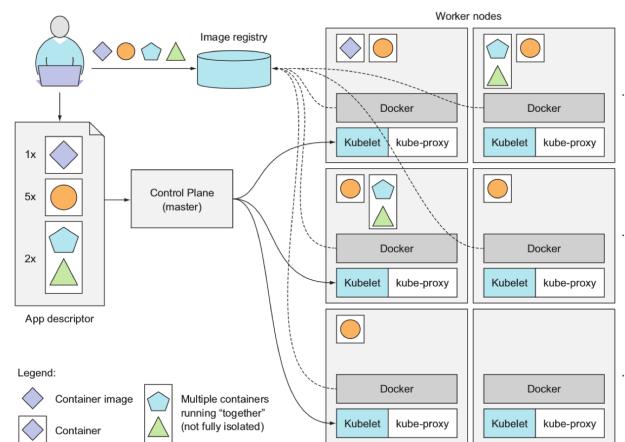
Il Kubelets eseguito sul worker nodes si occupa di controllare Docker to pull i container images dalla repo ed eseguirle.

## Management e Maintenance

Kubernetes controlla continuamente che lo stato dell'applicazione metta sempre la descrizione data dai dev.

Se uno o più istanze smettono di funzionare, Kubernetes si occupa di riavviare.

Se un'istanza, o un intero working node, diventa inaccessibile, Kubernetes seleziona un nuovo nodo per tutti i container che erano in esecuzione sul nodo.



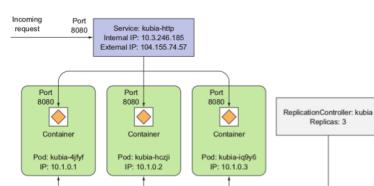
- While the application is running the developer can decide to increase/decrease the number of instances, in this case the platform can take care of adding/removing instances at runtime
- The developer can even let the platform decide the optimal number of instances of a certain container, in this case the platform will adjust the number of running instances based on real-time metrics like CPU load, memory consumption, query per second, etc.

## Container Migration

- The platform automatically takes care of instantiating containers and migrating them when required (e.g. when a working node fails)
- When a container provides a service to external client, the platform has to take care of forwarding requests as the container is moved across the platform
- What it is usually performed is that the platform exposes a specific service with a single public IP address, regardless of the number of containers and their location
- The platform with the kube-proxy component takes care of connecting to the containers and forwarding the traffic in a proper manner
- In addition, the component takes care of implementing load-balancing policies to forward the traffic in a balanced manner in order to ensure scalability

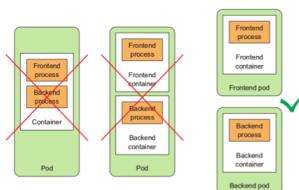
## Load Balancing

- A service can be connected to a pod that requires multiple redundant instances for load balancing
- In this case the service spreads the requests across the different instances
- To each pod the platform associates a replication controller for taking care that the proper number of instances is always running



## Pod design

- The general rule to group containers into pods is that developers should always run containers in separate pods unless a specific reason requires them to be part of the same pod, so they are deployed on the same working node
- For instance an application might have latency requirement in the data exchange between two container, in this case it is convenient to group them in the same pod so they do not communicate over the network



## Storage

Il più semplice tipo di storage che può essere impl. con un hard drive è il Block storage.

Un blocco è una seq. di byte o bit che possono essere stoccati direttamente sul settore fisico dell'hard drive.

In testa a un insieme di blocchi, un file system viene solitamente creato.

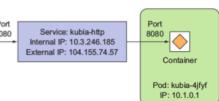
Recentemente un nuovo tipo di storage è stato creato: Object Storage.

Gli oggetti sono file con un metadata corrispondente.

Ogni oggetto ha associato un unique ID che è calcolato basato sul contenuto del file e meta-data.

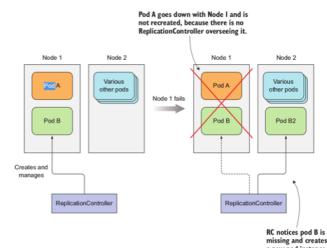
## Exposing Applications

- Pods are connected to a local private networks, in order to allow communication among them
- Each pod gets its own IP address, however, this address is internal to this network and it is not accessible from outside
- Such local private network allows the communication among containers running on the same or different workers
- If a service needs to be exposed to external networks (e.g. because it exposes the web or the REST API interface of the application) the developer needs to define a service object
- A service object instructs the platform to associate with a pod an external IP address and port so that the service exposed by the pod can be reached from external hosts
- The service takes care of forwarding the request coming to that external IP address to the local IP address of one of the associated pod instances



## Failure handling

- The replication controller is also responsible for monitoring the instances of a pod to detect failures
- If the failure of a working node is detected, the replication controller detects it and react by creating a new pod instance



IC notices pod B is missing and creates a new pod instance.

Le applicazioni possono accedere all'oggetto tramite ID, l'insieme dei metadata non è definito a priori e può essere esteso.

Ogni oggetto è immutabile, un cambiamento produce un'altra versione che è salvata come nuovo oggetto.

## RAID (Redundant Array of Independent Disk)

Soltanente gli Hard drives sono su un singolo server e un harddrive può fallire (lifespan ~ 5 years).

Inoltre un singolo server ha un numero di HD slots limitato.

RAID è una tecnologia che combina HD multipli in uno o più HD virtuali.

Le funzionalità RAID possono essere impl.

1) Via hardware, tramite un RAID controller al quale gli HD sono connessi.

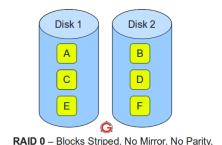
Il RAID controller gestisce gli HD e le funzionalità sono nascoste all'uso il quale può accedere solo all'HD virtuale.

2) Via software, dove le funz. RAID sono impl. dall'uso il quale gestisce l'accesso al HW fisico e la creazione dell'HD virtuale.

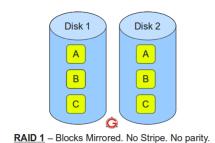
## RAID schema

A seconda dello schema, i blocchi del logical HD vengono salvati nell'HD fisici in modo diverso:

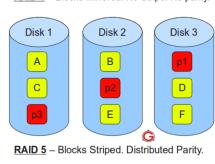
- RAID 0 : Adottato per la performance, dove i blocchi sono striped across i HD fisici.



- RAID 1 : Adottato per ridondanza massima, dove i blocchi sono copiati sui vari HD fisici.



- RAID 5 : È un tradeoff tra performance e redundancy, dove i blocchi sono striped sugli HD e dei parity block sono aggiunti a un insieme di blocchi per garantire la fault tolerance.



## Distributed File system

Il file system viene distribuito sui differenti servers, il trasferimento dei dati e la sincronizzazione vengono effettuate tramite LAN.

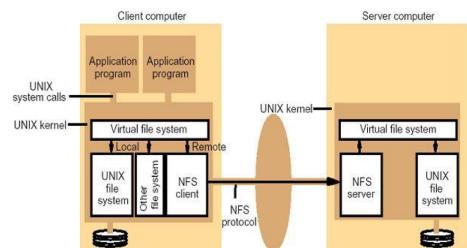
### NFS (Network file system)

È un tipo di DFS che adotta un approccio client server nel quale: un server centrale offre l'accesso ai client al suo file system.

NFS definisce il comm. protocol e i messaggi sono avbinati per assicurare la consistenza e semplificare l'impl.

Tutte le funzionalità sono impl. all'interno del kernel Linux e le funz. NFS sono nascoste all'App.

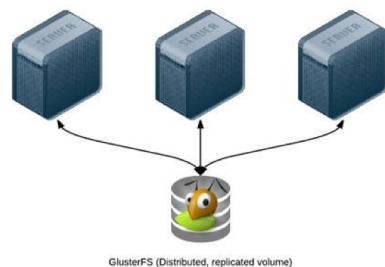
**Problema:** Architettura centralizzata



### Gluster FS

È un tipo di DFS che crea un distributed FS che unisce le capacità di storage disponibili su tutti i nodi.

Non c'è differenza tra client e server, tutti i nodi partecipano allo stesso modo.



**Basic Modes:** Replicated volumes, distributed volumes, striped volumes

**Advanced Modes:** Combinazione delle modalità di base: striped replicated e distributed replicated.

## Cloud Storage

Il cloud storage è diverso dal computing model tradizionale.

Il requirement principale è che esso deve scalare:

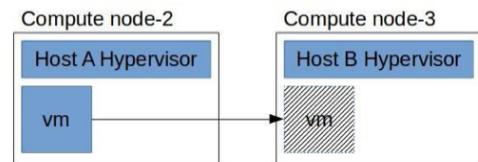
- Deve scalare con il numero di VMs che la piattaforma deve gestire
- Deve scalare con il numero e la grandezza dei volumi creati
- Deve scalare con la quantità di dati generati dai differenti servizi

È possibile andare a creare virtual harddrives sui physical hard drives, ma questa soluzione non sarebbe comunque scalabile e non permette la VM live migration.

### Live Migration

Permette alle VM di essere mosse dinamicamente da un compute node a un altro, mentre la VM è accessa e in esecuzione.

Questo permette di minimizzare il down-time e evitare di spegnere la VM.



La live migration richiede uno shared storage, altrimenti, la VM dovrebbe essere spenta e il virtual hard drive mosso dal local storage di un compute node a un altro.

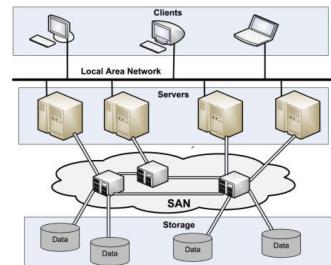
### Storage Area Network (SAN)

Una SAN è una rete ad alta velocità composta da storage devices (Not regular servers).

Gli storage devices sono connessi ai server che compongono la piattaforma cloud, per fornire lo storage.

Soltanente sono dispositivi disk-based che usano tecnologie veloci per comunicare.

Vengono introdotti protocolli ad-hoc per la comunicazione tra server



e storage, uno di essi è l': SCSI protocol che lavora su una rete TCP/IP.

**PROBLEMA:** La richiesta di HW ad-hoc rende questa soluzione complessa e costosa

### Distributed Storage Systems

Sono sistemi che hanno come obiettivo il fornire una unified storage solution per offrire scalabilità, alte performance, fault tolerance usando dell' hardware general-purpose comune.

Una delle soluzioni più popolare in questo momento è CEPH.

### CEPH

CEPH fornisce uno storage system affidabile, robusto e di livello aziendale su un hardware comune.

### Caratteristiche:

- 1) Ogni componente deve essere scalabile
- 2) Non ci può essere un singolo punto di fallimento
- 3) Il SW deve ruotare su HW comune
- 4) Ogni cosa deve essere gestibile da se stessa, se possibile

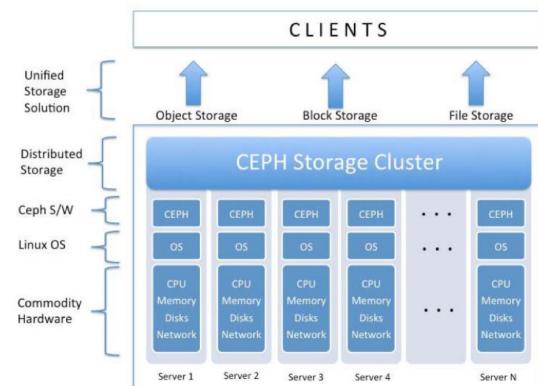
Basandosi su queste caratteristiche, ceph gestisce da solo la data replication e failure recovery.

### Ceph Services

Ceph può essere installato su un normale server per creare un Distributed storage system (ceph cluster) nel quale la capacità di tutti gli hard drives di tutti i server è resa disponibile.

Ceph offre diversi storage types:

- 1) Object Storage
- 2) Block Storage
- 3) FS storage

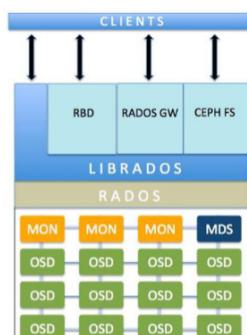


### Ceph Architecture

Per offrire client storage service oltre che l'obj storage, servizi

addizionali sono creati in testa a LIBRADOS:

- 1) Ceph Block service: Fornisce il block storage, il quale può essere mappato, (RADOS block service) formattato e montato come ogni altro disco al server.
- 2) Ceph file system: Offre un POSIX-compliant, DFS di una qualsiasi dimensione.



CephFS si affida a Ceph Metadata Server, il quale tiene traccia della file hierarchy e salva metadati solo per cephFS.

Un ceph block e RADOS non richiedono metadati essendo che non necessitano di un ceph MDS deamon.

Il RADOS layer è il core di CEPH, il quale fornisce tutte le features.

Le funzionalità come disponibilità, affidabilità... sono fornite implementando nel RADOS layer il CRASH algorithm.

Le funzionalità di RADOS sono impl. in modo distribuito dai servizi OSD e MON.

OSD (ceph obj storage Device): si occupa di salvare i dati sul HD fisico di ogni nodo del cluster.

Venne solitamente creata una OSD instance per HD fisico

MON (ceph Monitor): è responsabile di monitorare la salute dell'intero cluster.

Un insieme di MON instances è deployed per fornire fault tolerance.

### Ceph object

Un ceph object comprende dati e metadati che sono raggruppati e forniti tramite un globally unique ID.

Essi possono essere molto grandi e sono salvati in object-based Storage Device (OSDs) in modo replicato.

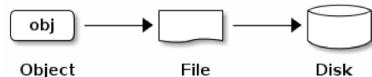
Quando il ceph storage cluster riceve una richiesta

successivamente,

di scrittura, salva i dati come oggetti e l'OSD deamon scrive i dati in un file del

OSD FS

ID	Binary Data	Metadata
1234	0101010101010100110101010010 0101100001010100110101010010 0101100001010100110101010010	name1 value1 name2 value2 nameN valueN

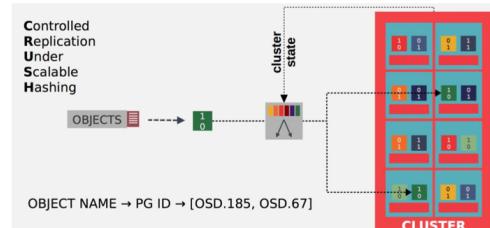


### Crush Algorithm

Ceph clients e OSD usano il CRUSH algo per calcolare efficientemente le info riguardo la posizione di un oggetto.

CRUSH fornisce un migliore data management system rispetto ai vecchi appross e distribuisce il lavoro a tutti i client e OSD deamons nel cluster.

CRUSH usa la data replication per assicurare la resilienza.



## Cluster Map

Lo stato di un cluster è rappresentato attraverso una Cluster Map.

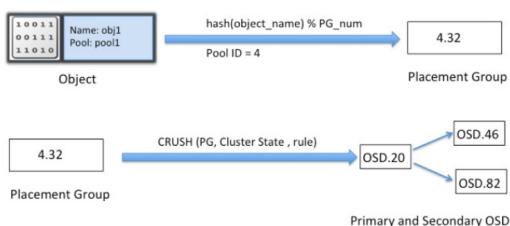
La Map può essere ottenuta da qualsiasi Mon instance ed include le seguenti info:

- 1) Monitor Map = Indica l'epoca corrente, quando la map è stata creata e l'ultima modifica
- 2) OSD Map = Contiene la lista dei pools, replica sets, list of OSDs e il loro stato
- 3) PG map = Contiene : dettagli su ogni placement group come : il PG id, Up set, Acting set , stato del PG
- 4) Cluster Map = Contiene la lista degli storage device , la failure domain hierarchy e le regole per attraversare la gerarchia durante il salvataggio dei dati

Un PG determina l'attuale OSD sul quale l'oggetto è salvato

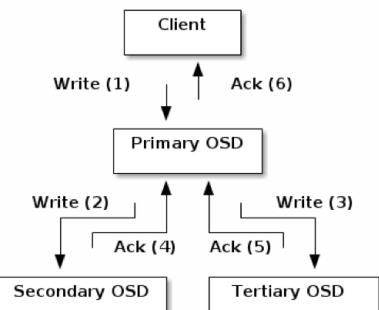
## Data Placement

- 1) Prima che i ceph Clients possano leggere/scrivere i dati, devono contattare un Ceph Monitor per ottenere la copia più recente della cluster map.
- 2) Successivamente, il client prima deriva il PG applicando una hash function sul obj ID.
- 3) Per calcolare l'OSD dove salvare l'oggetto, il crush algo viene eseguito per trovare il PG sulla lista di OSDs basata sul cluster state.
- 4) Il risultato è una lista di OSDs ordinati, un primary OSD è un insieme di OSDs secondari che salvano l'oggetto (primary OSDs) e la replica (secondary OSDs)



## Data Replication

- 1) Un client scrive l'oggetto, per identificare il PG, nel primary OSD.
- 2) Il primary OSD, con la sua copia della CRUSH MAP, identifica il secondo e terzo OSDs per la replica.
- 3) Dopo esegue le repliche sul PG, secondary OSDs e tertiary OSDs.
- 4) Eventualmente risponde al client che l'obj è stato salvato con successo.

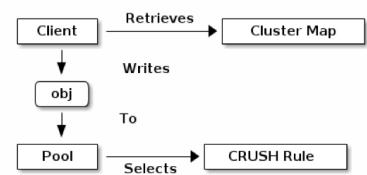


## Pools

Ceph storage usa le pool, le quali sono partitioni logiche per salvare gli obj.

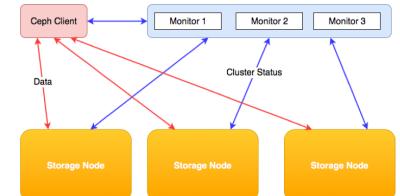
Un ceph client ottengono una Cluster Map da un Ceph monitor e scrivono l'obj nella pool.

Pool differenti possono essere istanziate per accomodare dati di diverse appl.



## MON Redundancy

Un ceph Storage Cluster può operare con un monitor singolo, ma per assicurare affidabilità e fault tolerance, Ceph supporta un cluster di monitor.



Alcuni monitor, causa latenza / altre fault, possono rimanere inidicio.

Ceph usa una maggioranza di monitor e il Paxos algo per stabilire un "consenso" tra i monitor rispetto allo stato attuale del cluster.

## Placement Group Computation

(+ o più)

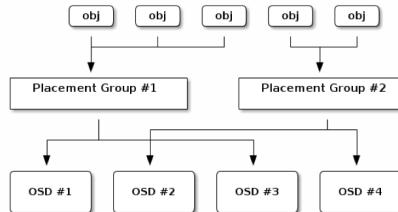
Ogni pool ha un numero PG che vengono mappati dinamicamente agli OSDs dal CRUSH algo.

Quando un ceph client salva un oggetto, l'hash mappera ogni obj al PG.

Il mapping agli OSDs è calcolato sulla base delle current failure zones del cluster, in modo che

i dati possono essere considerati safe e disponibili anche se qualche componente faila.

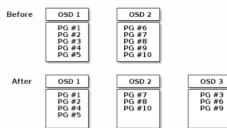
In aggiunta, i gruppi sono mappati per assicurare un data balancing tra gli OSDs, proporzionale al physical size di ognuno.



The definition of failure zones does not only take into account different hard drives and different servers but it can be modified in order to include other factors like the network infrastructure or the architecture of the power supply system

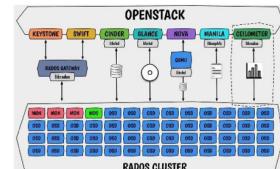
## Recovering and Rebalancing

- Mapping objects to placement groups creates a layer of indirection between the Ceph OSD Daemon and the Ceph Client
- This layer of indirection between the Ceph OSDs and the Ceph Client allows the Ceph Storage Cluster to grow (or shrink) and rebalance where it stores objects dynamically, for instance due to failures
- For instance, when you add a Ceph OSD Daemon to a Ceph Storage Cluster, the cluster map gets updated with the new OSD and consequently all the mapping of the placement groups
- This changes the object placement, thus resulting in rebalancing, i.e. movement of data across different OSDs to ensure proper balance across OSDs



## Ceph and OpenStack

- Many OpenStack services can exploit Ceph as cloud storage technology
- For instance
  - Cinder can exploit Ceph to store VM volumes, in this case Ceph block storage is exploited
  - Swift can exploit Ceph to store the its objects by using the default object storage service
  - Ceilometer can use Ceph to store telemetry data
  - Glance can use Ceph to store OS templates for instance creation
  - Nova can use Ceph to store VM virtual hard drives



## Storage as a Service

- Among the services offered by cloud provider one of them is the Storage as a Service (STaaS)
- This service offers access to cloud storage capabilities over the Internet
- In other words they offer access to their cloud storage infrastructure
- Often STaaS services adopts the Object storage model, they exposes a REST interface to store/read objects
- This can be used for backup or long term storage of large data
- One example is the Swift OpenStack service that can be used to expose a Ceph cloud storage for instance
- AWS exposes a large set of cloud storage services, from S3 storage (similar to Swift) or AWS Glacier (for cheap long term storage)