Panoramica su architetture per Virtualizzazione di sistemi

Prof. Vittorio Ghini

Sistemi Operativi 2021/22

Corso di Laurea in Ingegneria e scienze informatiche
Università di Bologna

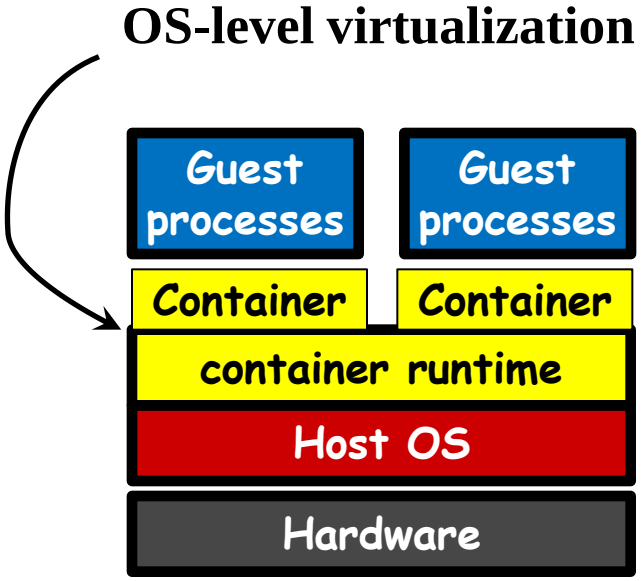
Virtualizzazione vs. Emulazione

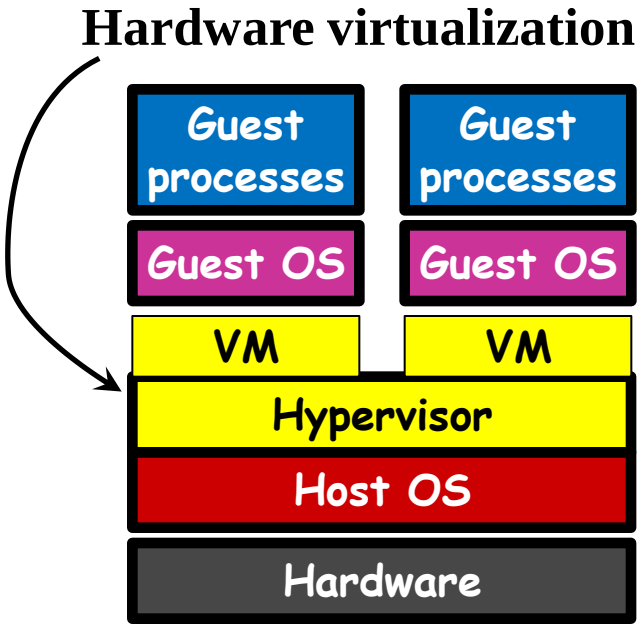
- Tramite **virtualizzazione** è possibile eseguire uno o più sistemi operativi (ed il relativo software applicativo) su un unico PC, in un ambiente protetto e monitorato che prende il nome di *macchina virtuale (VM)*.
- Il sistema operativo in cui viene eseguita la macchina virtuale, viene detto *ospitante* (host)
- La macchina virtuale è chiamata *ospite* (guest).
- Il codice della macchina virtuale viene eseguito direttamente dal sistema ospitante, ma il sistema ospite "pensa" di essere eseguito su una macchina reale.
  - Il codice della macchina virtuale, perciò deve essere codice macchina eseguibile dalla macchina hardware reale sottostante.
  - Non posso virtualizzare un sistema operativo, che dovrebbe girare su un x86 a 64 bit, su un processore ARM.

- **Emulazione di processore**. In questo caso l'hardware viene completamente emulato dal programma di controllo, cioè ogni istruzione che il sistema guest esegue viene tradotta in una sequenza di istruzioni della macchina host. Il processo di emulazione risulta più lento rispetto alle due forme di virtualizzazione precedenti, a causa della traduzione delle istruzioni dal formato del sistema ospite a quello del sistema ospitante.

Virtualizz. del Livello HW o Livello OS

- La prima grande distinzione riguarda il tipo di risorse virtuali che si vuole presentare all'utente: **macchine virtuali** oppure **partizioni (container) isolate**.
- Nel caso delle **macchine virtuali** (livello Hardware) all'utente del sistema di virtualizzazione viene presentata un'interfaccia su cui installare un sistema operativo, quindi una CPU virtuale (ma dello stesso tipo della CPU fisica) e risorse HW virtuali.
- Nel caso dei **container** (livello OS) all'utente viene presentata una partizione (container) del sistema operativo corrente, su cui installare ed eseguire applicazioni che rimangono isolate nella partizione, pur accedendo ai servizi di uno stesso o.s.





Livello HW vs. Livello OS(container)

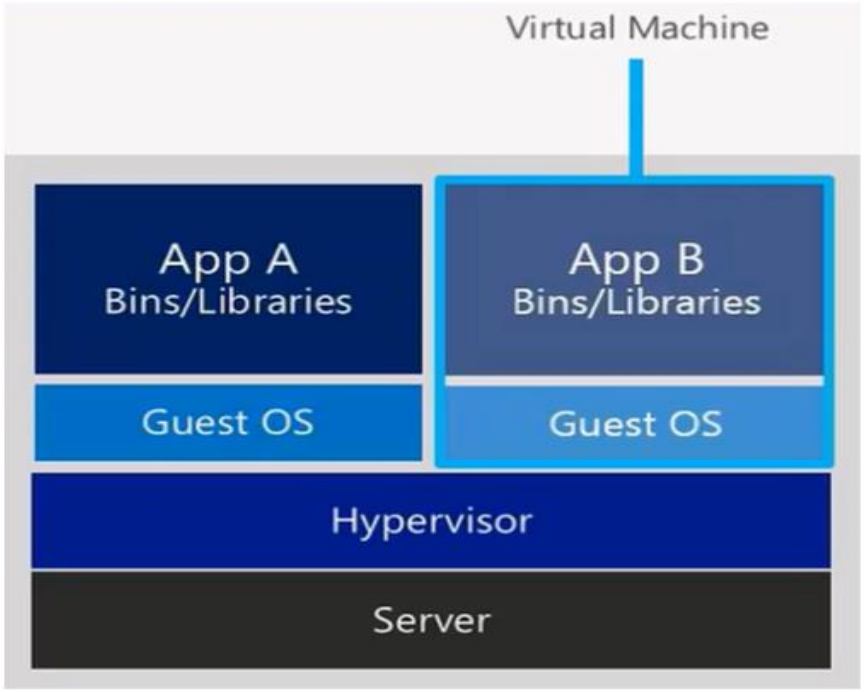
- La virtualizzazione OS-level è composta da un solo kernel (quello del sistema operativo host) e multiple istanze isolate di user-space (chiamate anche partizioni o contenitori), che possono essere avviate e spente in maniera indipendente tra loro. Ogni container contiene un proprio filesystem e proprie interfacce di rete. Viene garantito l'isolamento del filesystem, IPC e network. Inoltre fornisce un sistema di gestione delle risorse quali CPU, memoria, rete e operazioni I/O.
- Nella virtualizzazione hardware, i sistemi operativi eseguono in modo concorrente sullo stesso hardware e possono solitamente essere eterogenei. L'**hypervisor** (chiamato anche *Virtual Machine Monitor*) si occupa di multiplexare l'accesso alle risorse hardware e garantire protezione e isolamento tra le macchine.

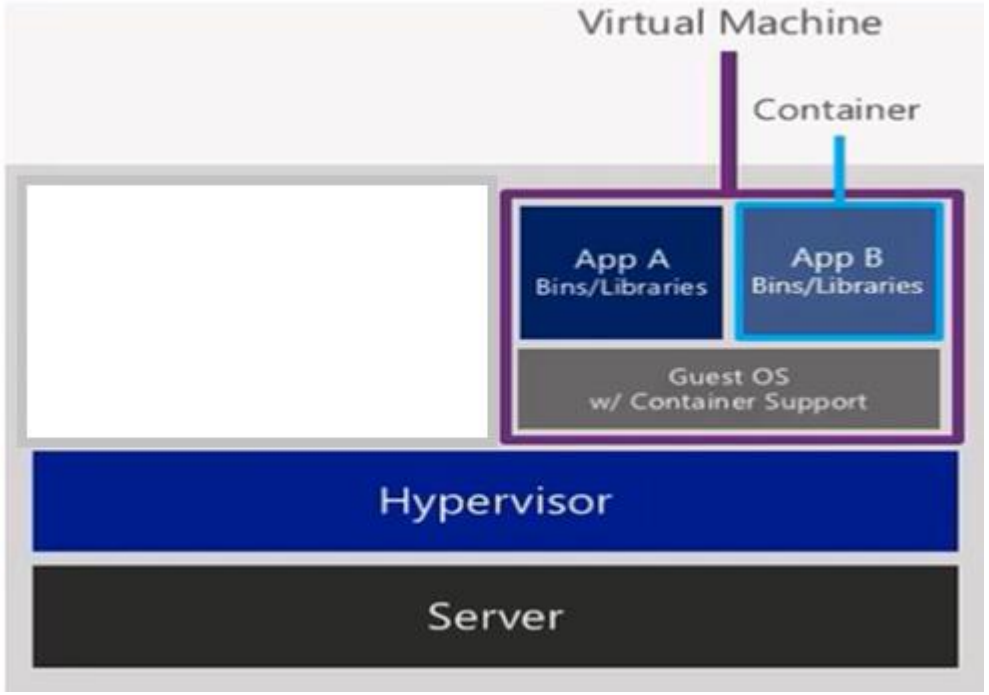
Quindi:

- **OS-level (o container-level)**: offre contenitori per applicazioni ed esegue un solo kernel, quello del o.s. Host.
  - Vantaggi: basso overhead per il context-switch, basso overhead di memoria
  - Svantaggi: non può ospitare sistemi operativi differenti, l'isolamento non può essere del tutto perfetto
- **hardware-level**: quando offre macchine virtuali.
  - Vantaggi/Svantaggi: speculari rispetto a OS-level

2 parole sui Container - perché?

- Creano uno spazio utente isolato, con proprie interfacce di rete, librerie e files, isolando una applicazione dal resto del sistema operativo.
  - **Pacchettizzo una applicazione, rendendola pronta per il deployment.**
- Più container possono appoggiarsi ad uno stesso spazio kernel, risparmiando spazio disco e condividendone i servizi di base.
- Costruito e customizzato un container per una specifica applicazione, posso replicare quel container più volte, anche su una stessa VM o su VM diverse.





Quali Container ?

Ricordiamo solo i principali:

- **Docker**
  - Si appoggia sul s.o. **Linux** che fornisce, a livello kernel) un supporto per container detto **LXC (Linux Container)**.
    - Non confondete LXC con LXD che è un sistema per dispiegare facilmente virtual machine Linux.
- **Hyper-V** di Microsoft (fornisce unità di isolamento chiamate Container, in realtà sono delle macchine virtuali).
  - Tra l'altro, si parla di container di Hyper-V, ma su sistemi diversi (ad esempio, windows server e windows 10) sono piuttosto differenti, come tipo di isolamento fornito
- **Container di Windows Server** (sono effettivamente dei container)
- NB: E' possibile installare il s.o. Windows server in una configurazione minimale, detto Nano Server, ottimizzato per stare dentro macchine virtuali di hyper-v o per sostenere container di windows server. https://docs.microsoft.com/it-it/windows-server/get-started/getting-started-with-nano-server

Docker - Contents

- 0. Immagini di container disponibili su repository docker hub.
- 1. Eseguire un container Docker interattivamente.
- 2. Installare applicazioni in un container docker.
- 3. Salvare i cambiamenti di un container creando una nuova immagine di container.
- 4. Eseguire una immagine di container salvata localmente esponendo all'esterno una porta di protocollo del container.
- 6. Kubernetes
- 7. Kubernetes-as-a-Service.
- 8. Container-as-a-Service
- 9. Functions-as-a-Service.

0. Lavorare con le immagini docker

- I container Docker sono eseguiti a partire da immagini di container che possono essere scaricate dal repository Docker Hub, un cosiddetto "registro" di immagini Dockergestito dalla compagnia che sostiene il progetto Docker.
- Ciascuno può scaricare un'immagine docker ed eseguirla localmente.
- Dopo che l'immagine del container è stata scaricata, la runtime machine di docker crea il container dall'immagine.
- La copia dell'immagine scaricata viene salvata su un registro (local registry).

- Per verificare se il sistema docker funziona puoi provare a scaricare l'immagine di test denominata hello-world

  docker **run** hello-world

1. Eseguire container docker interattivamente

- **Containers interattivi**. Come esempio eseguiamo un container usando l'ultima immagine disponibile di Ubuntu. La combinazione dei flags **-i -t** consente di interagire con una shell bash all'interno del container connessa mediante stdin/stdout/stderr alla shell in cui lanciamo il container in foreground:

  docker **run** -it --name myubuntu ubuntu

- l'opzione --name myubuntu assegna il nome myubuntu al container. Se non è specificato il nome, docker assegna un nome generato casualmente.

- Your command prompt should change to reflect the fact that you're now working inside the container and should take this form:

  root@d9b100f2f636:/#

- Note the container id in the command prompt. In this example, it is d9b100f2f636. You'll need that container ID later to identify the container when you want to remove it.

1.1. Eseguire interattivamente comandi bash all'interno di un container.

- **Ora puoi eseguire dei comandi all'interno del container myubuntu.**
- Per esempio esegui il comando ls

  ls
- Poi edita un file

  echo ciao > miofile.txt
- E verifica l'esistenza del file nel container

  cat miofile.txt
- **Ogni cambiamento apportato al filesystem del container rimane confinato al filesystem del container.**
- **Per terminare il container, digita exit al prompt dentro il container. La shell termina e con esso termina il container.**
- **Se fai partire nuovamente il container dall'immagine ubuntu, il file precedentemente creato non c'è più nel nuovo container.**
- Prova ancora "docker run -it --name myubuntu1 ubuntu" e verifica che il file miofile.txt non c è.

2. Installare applicazioni in container

docker **run** -it --name myubuntu ubuntu

- Installare il pacchetto node.js e alcune altre utility:

  apt update

  apt install vi wget curl

  apt install nodejs

- Configura il server http dinamico editando il file /root/index.js

```
var http = require('http');
        http.createServer(function (request, response) {
                response.writeHead(200, {'Content-Type': 'text/plain'});
                response.end('ciao a tutti\n');
}).listen(8888);

console.log('Server running at http://127.0.0.1:8888/');
```

- Fai partire il server nodejs

  node index.js &

- Testa il funzionamento del server nodejs dall'interno del container

  curl http://127.0.0.1:8888/

3. Salvare i cambiamenti creando una nuova immagine del container in locale

- Dopo avere eseguito un container interattivamente, battezzandolo myubuntu, e dopo averne modificato il filesystem, installando il server hhtp dinamico nodejs,
- termino l'esecuzione del container
- e poi ne salvo l'immagine in locale chiamando la nuova immagine di container ubuntuVic

```
docker commit -m "added node.js" -a "Vic" myubuntu vic/ubuntuvic
```

- **Ora localmente ho la nuova immagine del mio container e posso eseguire dei container partendo da questa immagine. Nei container creati mi ritrovo installato il mio server nodejs e le utility vi e wget**

```
docker run -it --name myubuntu vic/ubuntuvic
```

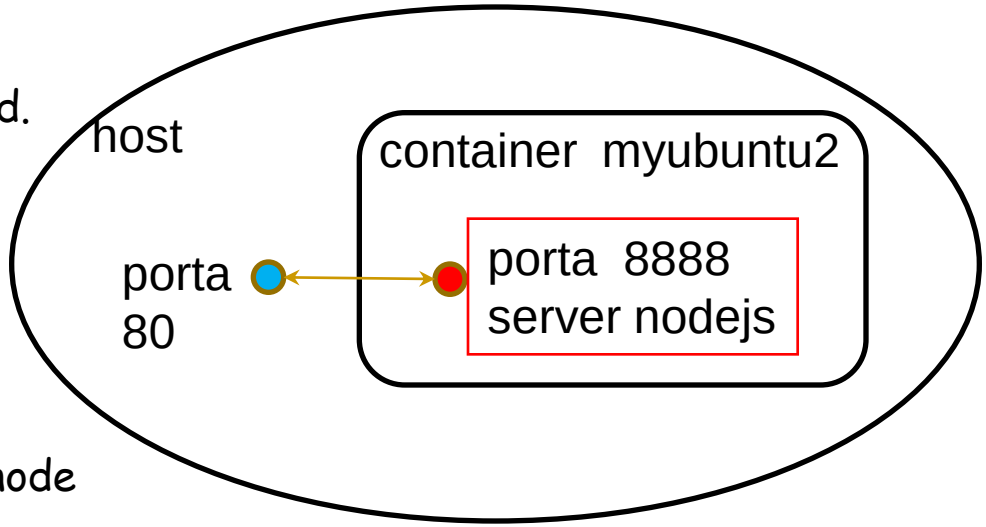
- Prova dentro il container a far partire il server web e ad usarlo con curl

```
node index.js &
curl http://127.0.0.1:8888/
```

4. Eseguire un container in background esponendo una porta all'esterno del container

docker **run** -it -d --rm **-p 80:8888** --name myubuntu2 vic/ubuntuvic node index.js

i parametri hanno i seguenti scopi:
vic/ubuntuvic è l'immagine del container.
-d significa esegui il container in background.
--rm elimina il container quando termina.
-p 80:8888 collega la porta 80 dell'host
alla porta 8888 interna del container.
node è il comando da lanciare non appena
il container viene creato e parte.
index.js è l'argomento passato al comando node
da eseguire quando il container parte.



Verificare: stando nella macchina in cui si esegue il container, e quindi FUORI dal container, eseguire

curl http://127.0.0.1:80/

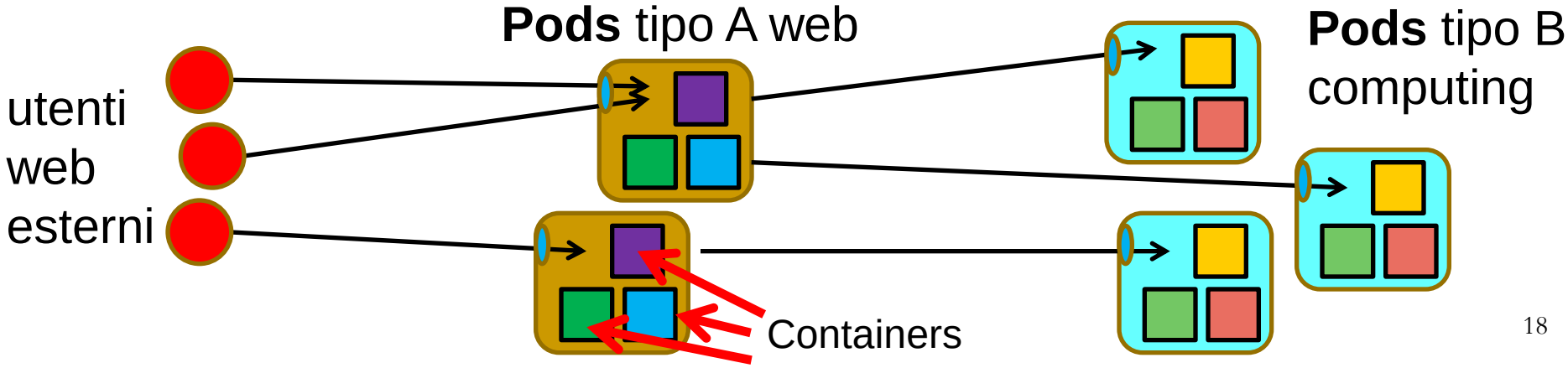
per accedere al servizio fornito dal container stando FUORI dal container.

6. Orchestrazione di container con KUBERNETES

fornire automaticamente
scalabilità e resilienza
ad applicazioni
basate su container docker
in esecuzione su cluster
di host fisici o virtuali

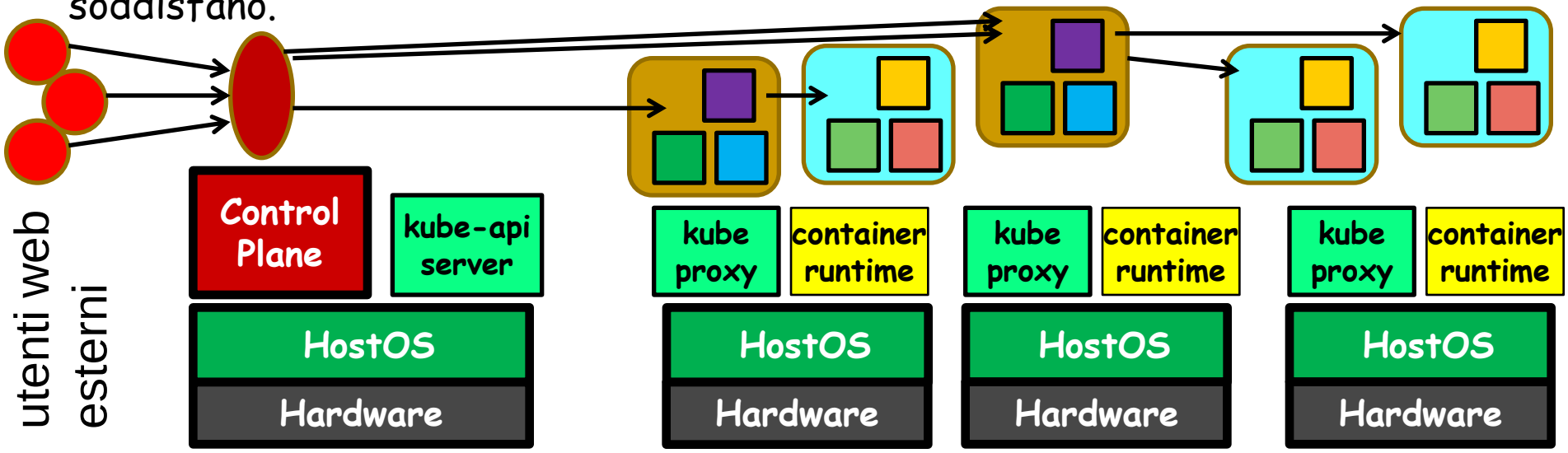
6.1 Applicazioni per kubernetes

- Le applicazioni progettate per kubernetes sono costituite da più **pods**.
- Ciascun pods è un insieme di container che cooperano e comunicano tra loro come se si trovassero in un unico host fisico e svolgono nel complesso un certo servizio, ad esempio un **pod può realizzare un servizio web** mentre un altro **pod può eseguire una serie di calcoli,** oppure può salvare dati in un database.
- **Un pod può essere replicato in più istanze** se occorre eseguire in parallelo le stesse operazioni e distribuire il carico. **Le diverse istanze dei** pods possono eseguire su uno stesso host oppure **su host diversi per distribuire meglio il carico**.
- Ciascun pods può esporre delle porte di protocollo per ricevere comunicazioni da container di altri pods.
- In tal modo, uno dei pod di tipo A, che espone il servizio web, riceve le richieste dagli utenti web esterni, poi chiede ad uno dei pod di tipo B di effettuare un calcolo, ricevuto il risultato del calcolo lo restituisce all'utente web esterno.



6.2 Cluster kubernetes - replicare pods su host diversi per distribuire carico

- Kubernetes organizza degli host (**nodi del cluster**) fisici o virtuali per poter eseguire e replicare i pods sui nodi, distribuendo così il carico dell'applicazione.
- L'applicazione decide quanti pods utilizzare e kubernetes decide automaticamente su quali nodi allocare i pods.
- Uno degli host svolge il ruolo di controllore del cluster (control plane), monitora gli altri host del cluster, decide automaticamente in quali nodi eseguire i pods e svolge il ruolo di punto di accesso al cluster dall'esterno. Gli altri host (nodi worker) eseguono i pods della o delle applicazioni.
- Gli utenti web esterni vedono solo il nodo controllore gli inviano le richieste.
- Il nodo controllore instrada le richieste ricevute ai pods dei nodi worker, che le soddisfano.



6.3 Cluster kubernetes - gestione resilienza

- Se un pod va in crash kubernetes ne crea immediatamente una nuova istanza.

6.4 Cluster kubernetes - gestione scalabilità

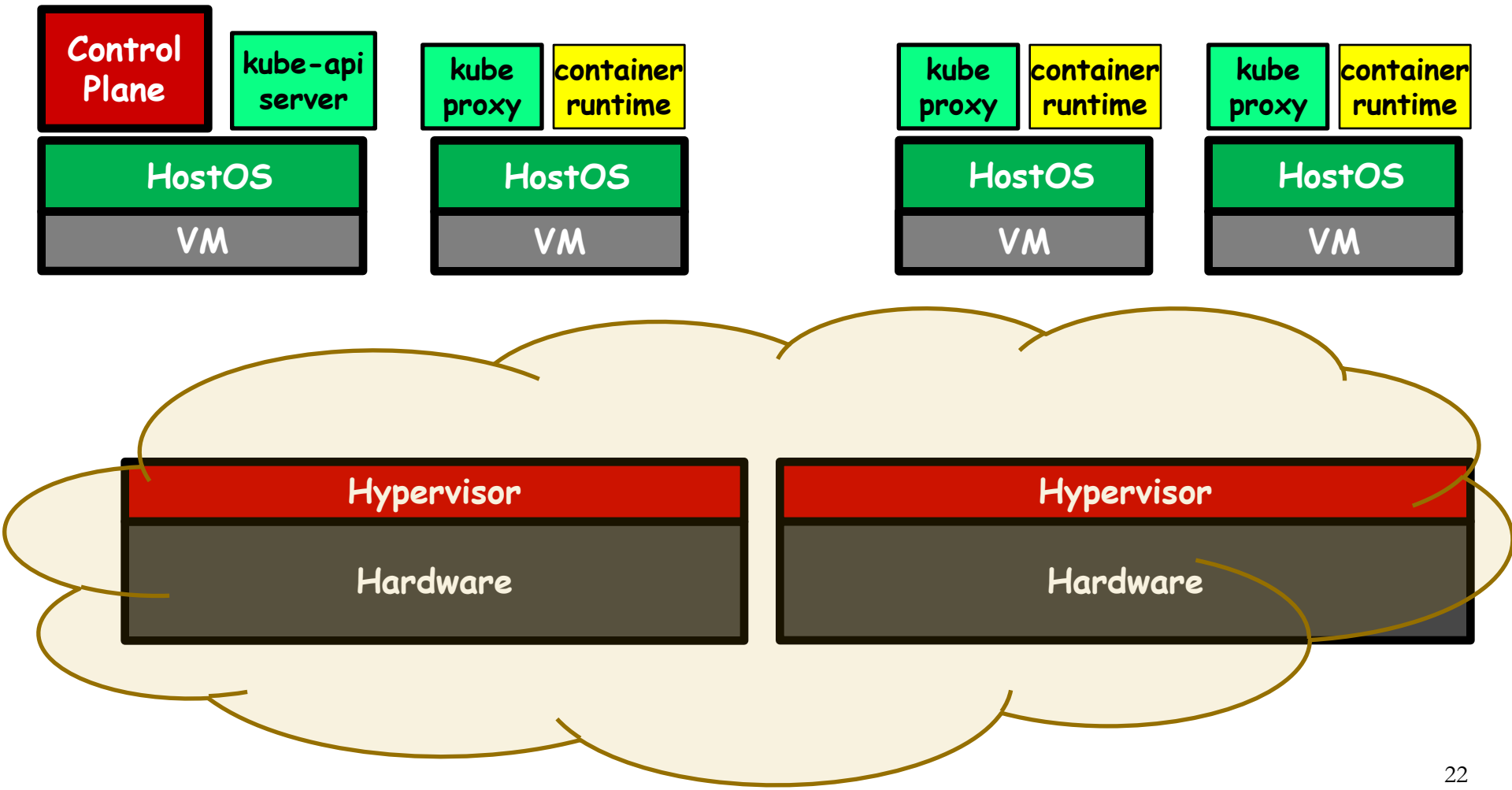
- L'applicazione decide quanti pods utilizzare e kubernetes decide automaticamente su quali nodi allocare i pods.
- L'applicazione può configurare Kubernetes affiche questo, all'aumentare del carico dell'applicazione, effettui 3 tipi di operazioni:
  - **autoscaling verticale**: aumenti le risorse a disposizione di un pod (uso della CPU, quantità di memoria usata).
  - **autoscaling orizzontale**: aumenti il numero delle repliche dei pods applicativi.
  - **cluster autoscaling**: aumenti il numero dei nodi che compongono il cluster redistribuendo i pods.

- Ciò consente di scalare aumentando o diminuendo il numero dei pods e/o dei nodi al crescere o al calare delle esigenze applicative.
- Ciò consente anche di mantenere l'applicazione in funzione anche se qualche host si guasta poiché ne rimangono altri a disposizione.

- La possibilità di realizzare il **cluster autoscaling** dipende da quali host usiamo:
- Per host fisici occorre un sistema di accensione degli host fisici guidabile via software (wake on lan o ...) mentre per gli host virtuali occorre un supporto, da parte del cloud pubblico o privato che ci fornisce le macchine virtuali, che ci dia la possibilità di creare e configurare nuove VM in automatico via software.

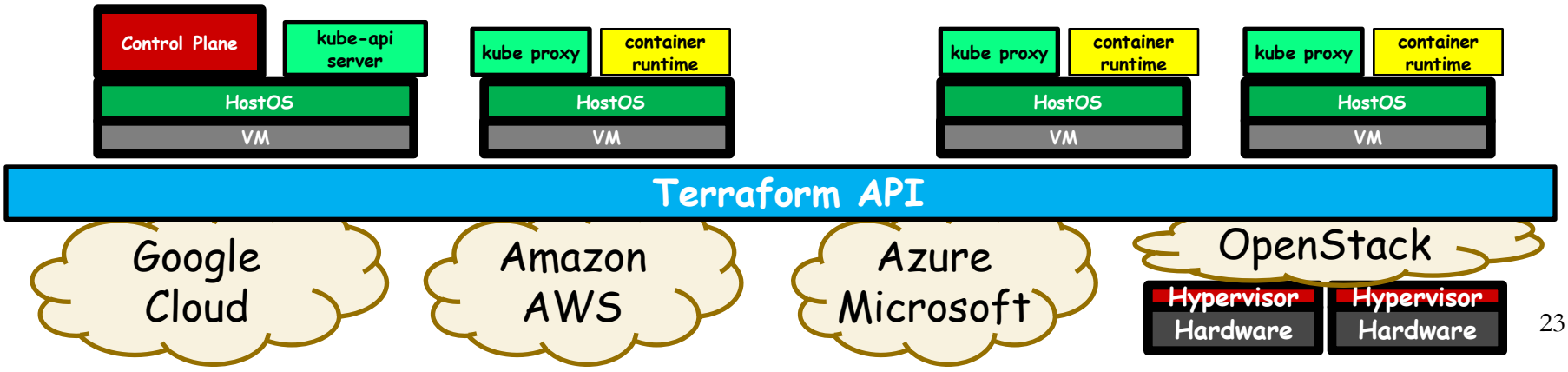
6.5 Cluster kubernetes con nodi virtuali

- E' possibile creare un cluster kubernetes formato da host che siano macchine virtuali. Questi macchine virtuali possono essere create manualmente su host fisici oppure possono essere create via software su cloud pubblici o privati.



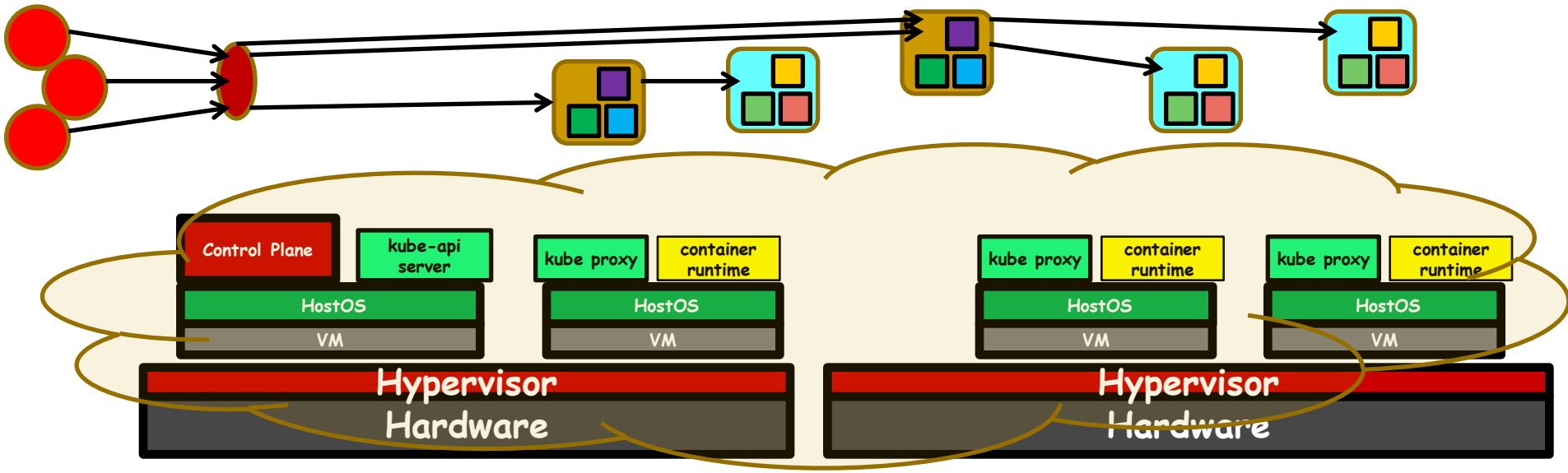
6.5.1 Cluster kubernetes con nodi virtuali

- Esistono cloud di diversi provider, i principali sono Google Cloud, Azure di Microsoft, AWS di Amazon, I.
- E' anche possibile costruire un cloud privato, partendo da macchine fisiche in una propria sala macchine, gestendo le proprie macchine con un insieme di applicativi open source basati su Ubuntu, noto col nome di **Openstack**.
- I sistemi cloud forniscono delle API mediante le quali possiamo ordinare, ,per via software, la creazione e la configurazione di VM e di reti virtuali tra le VM.
  - **Purtroppo le API differiscono da cloud a cloud.**
- Fortunatamente, alcuni software (ad esempio **Terraform**) offrono uno strato software che astrae dai sistemi cloud sottostanti ed offre un insieme di API utilizzabile su tutti i sistemi cloud per creare e configurare le VM.
- In tal modo è possibile creare un cluster kubernetes formato da host virtuali ed applicare il cluster autoscaling configurando kubernetes affinché usi Terraform per creare nuove macchine virtuali quando necessario per scalare.



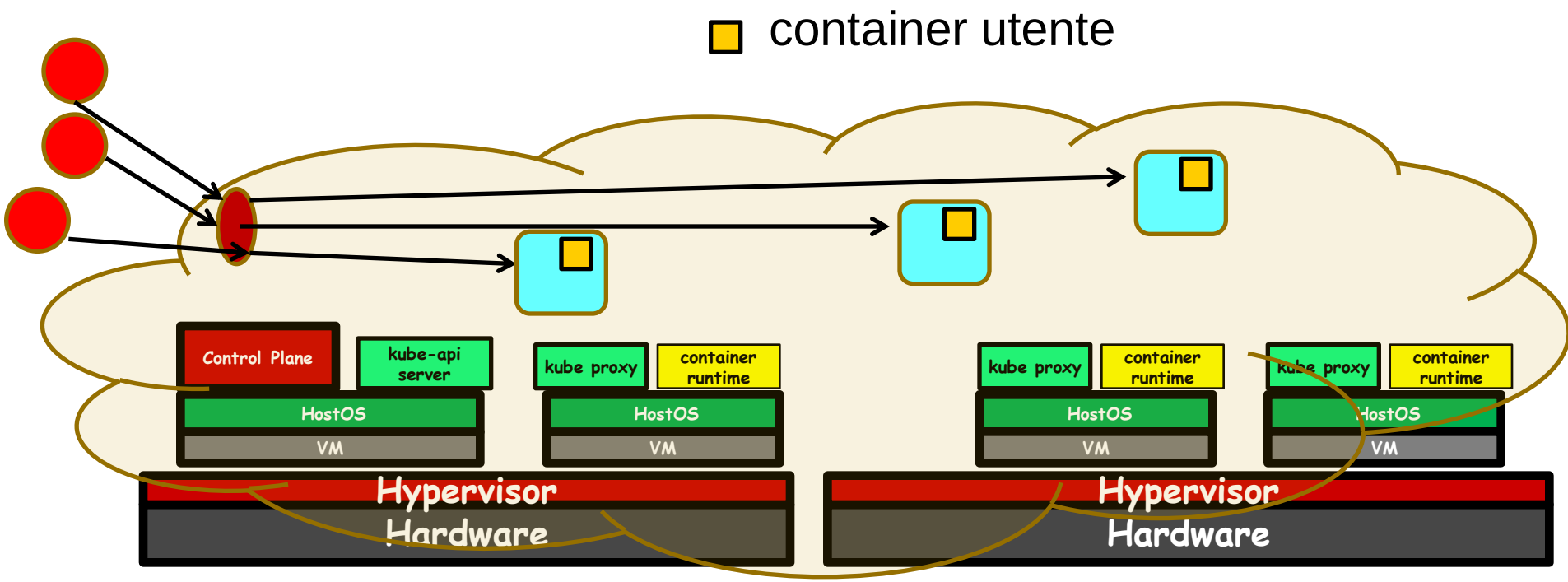
7. Kubernetes-as-a-Service

- Alcuni sistemi cloud offrono un servizio kubernetes senza che sia necessario creare esplicitamente il cluster kubernetes.
- Il cluster viene creato ed installato dal provider cloud su richiesta via software e a chi produce l'applicazione vengono fornite delle API mediante le quali chiedere di inizializzare il cluster kubernetes e di dispiegarvi sopra l'applicazione organizzata per kubernetes.
- Si installa una applicazione per kubernetes facendo costruire il cluster kubernetes al provider cloud.
- In tal modo è possibile costruire applicazioni scalabili e resilienti in maniera più semplice pagando al provider cloud il costo (non piccolo) del cluster.



8. Container-as-a-Service

- Alcuni sistemi cloud, in particolare offrono la possibilità di:
- far **eseguire un container, la cui immagine è fornita dallo sviluppatore dell'applicazione**, in molteplici istanze su un cluster kubernetes-as-a-service costruito nascostamente dal provider stesso.
- Il container viene incapsulato in un pod predisposto dal provider cloud ed eseguito nel cluster kubernetes. In tal modo lo sviluppatore non si deve occupare di gestire il cluster ma solo di costruire il container da far eseguire.
- Questo approccio è fornito da: Google Container Engine (GKE), Amazon EC2 Container Service (ECS), Azure Container Service (ACS), IONOS Container Cluster.



9. Serverless o Function-as-a-Service (FaaS)

- Alcuni ,provider cloud astraggono ulteriormente e offrono un servizio che consente di implementare delle funzioni che vengono eseguite, anche in più istanze contemporaneamente apparentemente senza definire né container né macchine virtuali e nemmeno cluster kubernetes su cui tali funzioni vengono eseguite.
- Quello che accade è che il provider cloud "probabilmente" incorpora la funzione definita dall'utente in un container e lo esegue su un cluster kubernetes replicandolo man mano che aumenta il carico di richieste da servire.
- Per l'utente è facile creare queste funzioni ed il problema della creazione del contesto di esecuzione (container e pod) è risolto, ovviamente dietro pagamento, dal provider cloud.
- Il provider offre alcune funzioni di libreria, lo sviluppatore implementa diverse altre funzioni, ciascuna delle quali può utilizzare, chiamandole, altre funzioni dello sviluppatore o di libreria, in modo da poter implementare così anche funzionalitò complesse.

- Troviamo questo approccio nei principali provider cloud: **AWS Lambda** in Amazon AWS, **Azure Functions** in Microsoft Azure e **Cloud Functions** in Google Cloud.