

Introduction to big data processing infrastructures

SEZIONE 1-Computational Challenge

Termine	Definizione sintetica
Kernel	Nucleo del sistema operativo. Gestisce risorse hardware (CPU, RAM, I/O) e coordina i processi. I container condividono lo stesso kernel del sistema host.
Shell	Interfaccia testuale (es. bash) che permette di interagire con il sistema operativo tramite comandi.
SSH (Secure Shell)	Protocollo sicuro per accedere in remoto a un sistema Linux da terminale (es. per accedere a cluster HPC o cloud).
CLI (Command Line Interface)	Interfaccia a riga di comando, usata per eseguire comandi testuali in shell o terminale.
Mount	Operazione che collega un file system (locale o remoto) alla struttura del file system corrente, rendendolo accessibile (es. mount -t nfs ...).
Volume	Directory o file condiviso tra host e container o tra più container, utile per input/output.
Snapshot	Copia istantanea di uno stato di un sistema o immagine (es. snapshot VM o immagine Docker).
Hypervisor	Software che gestisce macchine virtuali (es. VirtualBox, KVM), permettendo a più VM di girare su un singolo host.
Daemon	Processo di sistema che gira in background (es. dockerd è il demone di Docker).
Job	Unità computazionale, cioè un processo o script lanciato su batch system (es. HTCondor, Slurm).
Scheduler	Software che gestisce l'ordine e la distribuzione dei job tra i nodi disponibili.
Node	Unità fisica (server) o virtuale (VM) che esegue computazioni in un cluster.
Instance (VM)	Macchina virtuale lanciata su cloud (es. EC2 su AWS).
Queue (coda)	Lista di job in attesa di essere eseguiti da un sistema batch (HTCondor, Slurm).
Fork	Operazione con cui un processo genera una copia di sé stesso (figlio), tipico dei sistemi UNIX.

Termine	Definizione sintetica
Commit (Docker)	Operazione che salva lo stato corrente di un container in una nuova immagine.
Tag (Docker)	Etichetta che identifica una versione di un'immagine (es. python:3.9).
Repository (Docker Hub / GitHub)	Luogo online dove si salvano immagini Docker o codice versionato.
Image (Docker)	File immutabile che contiene tutto l'ambiente necessario per eseguire un'applicazione (sistema base + tool).
Container	Istanza in esecuzione di un'immagine Docker, isolata dal sistema host.
Virtual Machine (VM)	Simulazione completa di un computer, inclusi OS, CPU virtuale, RAM, storage.
Burst (cloud)	Momento in cui una VM su cloud utilizza risorse CPU in eccesso in modo temporaneo.
Instance type (EC2)	Configurazione hardware predefinita di una VM in cloud (es. t2.micro, c5.large).
Metadata server	Componente di un DFS che gestisce nomi file, permessi e la posizione dei blocchi.
Block (HDFS)	Unità di memorizzazione in cui viene suddiviso un file su un file system distribuito.
Throughput	Quantità di dati o job processati in un certo tempo (HTC punta all'alto throughput).
Latency	Ritardo tra input e risposta. Edge e Fog computing cercano di ridurla al minimo.
I/O (Input/Output)	Lettura e scrittura di dati su disco o rete, spesso causa di colli di bottiglia.
Scalabilità	Capacità di un sistema di aumentare efficienza aggiungendo risorse (verticale/orizzontale).
Fault tolerance	Capacità di un sistema di continuare a funzionare anche in presenza di errori hardware/software.
Orchestrazione	Gestione automatica di container e job (es. Kubernetes per Docker).
API (Application Programming Interface)	Interfaccia che permette a software diversi di comunicare. Usato per gestire VM, container, job da codice.

Termine	Definizione sintetica
Cloud-native	Software progettato per funzionare su ambienti cloud, tipicamente in container.

1. Definizione del problema

L'analisi NGS (Next Generation Sequencing) produce milioni di **reads** (brevi frammenti di DNA). Una sfida chiave è **mappare** questi frammenti sull'intero **genoma di riferimento** (es. genoma umano), che può contenere miliardi di basi.

Il problema si riduce a un'operazione computazionale nota come:

String Matching Problem: trovare rapidamente una sottostringa (read) all'interno di una stringa molto lunga (genoma).

2. Soluzioni algoritmiche

◆ BLASTn (Basic Local Alignment Search Tool)

- **Tipo:** allineatore basato su similarità locale.
- **Vantaggi:** Alta sensibilità → ottimo per trovare somiglianze anche in presenza di mutazioni.
- **Svantaggi:** Lento e poco scalabile per milioni di reads NGS.

◆ BWA (Burrows-Wheeler Aligner)

- **Tecnica usata:**
 - **Burrows-Wheeler Transform (BWT)** → riordina le sequenze per renderle facilmente comprimibili.
 - **FM-index** → struttura dati compressa che consente ricerche rapide.
 - **Vantaggi:**
 - Estremamente veloce (lineare rispetto alla lunghezza dei dati).
 - Adatto a processare **milioni di reads in parallelo**.
 - **Formato output:** file .SAM o .BAM.
-

3. Riferimento pratico nel GitHub BDP1_2025

Nel notebook (esercitazione Jupyter):

Code:

```
bwa index human_reference.fasta
```

```
bwa mem human_reference.fasta reads.fq > alignment.sam
```

Questo mostra come creare l'indice FM e allineare reads con BWA.

📁 I file .fasta e .fq simulano un piccolo esperimento NGS.

4. Checksum: controllo di integrità file

☑ Cos'è?

È una funzione che genera un codice univoco (hash) da un file, utile per:

- Verificare che il file non sia stato corrotto dopo il download o il trasferimento.
- Garantire l'integrità dei dati biologici (es. sequenze .fastq).

Esempio pratico:

```
md5sum sample.fastq
```

Se il checksum coincide tra mittente e ricevente, il file è integro.

5. Collegamenti con concetti successivi

- **Big Data:** Il mapping di milioni di reads rientra nei problemi di tipo **Big Data per volume e velocità**.
- **Cloud/HPC:** Queste operazioni di allineamento vengono eseguite su cluster HPC o piattaforme cloud per gestire l'elevata mole di dati.
- **File System condiviso** (prossima sezione): Necessario per accedere a reference genome e output condivisi da più nodi.

Aspetto	BLASTn	BWA	
Tipo	Allineamento locale	Allineamento globale rapido	
Sensibilità	Alta	Media	
Velocità	Bassa	Alta	
Scalabilità	Limitata	Ottima	
Usato per	Similarità e mutazioni	Mapping NGS su larga scala	

Big Data – Le 5V fondamentali

V	Significato	Esempio bioinformatico	
Volume	Quantità di dati immensa	Milioni di reads da sequenziamento NGS	
Velocità	Velocità con cui i dati sono generati/acquisiti	Real-time sequencing (Nanopore)	
Varietà	Diversità di formati e sorgenti dati	Dati genetici, epigenetici, clinici	
Veridicità	Qualità e affidabilità dei dati	Errori nei reads, dati rumorosi	
Valore	Utilità reale dell'informazione estratta	Nuovi biomarcatori, diagnosi precoce	

Le 5V sono fondamentali per comprendere perché servano tecnologie avanzate (cloud, HPC, container) per elaborare questi dati in modo efficiente.

SEZIONE 2 – Shared File Systems and NFS

1. Definizione del problema

Nei sistemi HPC/HTC o nel cloud, diversi **nodi** (computer) devono accedere **simultaneamente agli stessi file** (es. reference genome, output, database). Questo richiede un **file system condiviso** per garantire:

Accesso coerente, simultaneo e affidabile ai dati da più processi o nodi.

2. Tipi di File System

◆ File System Locale

- Ogni nodo ha il proprio disco.
- I dati **non sono visibili** agli altri nodi.
- ✗ Non adatto a workflow distribuiti.

◆ File System Condiviso

- I file sono **accessibili da tutti i nodi** attraverso una rete.
- I dati sono centralizzati ma visibili a tutto il cluster.
- ☒ Necessario per progetti bioinformatici distribuiti (es. analisi NGS parallele).

3. NFS – Network File System

☒ Cos'è?

È un **protocollo client-server** che consente a più computer di accedere allo **stesso file system remoto** come se fosse locale.

Come funziona?

- Il **server NFS** esporta una directory (es. /data/ngs/).
- I **client NFS** montano questa directory nel proprio file system (es. su /mnt/shared).
- I file sono visibili e modificabili da tutti i nodi.

□ Comandi tipici (esempio nel cluster o Docker):

lato client

```
sudo mount -t nfs server:/data/ngs /mnt/shared
```

lato server (in /etc/exports)

```
/data/ngs *(rw, sync)
```

4. Esempio pratico nel contesto bioinformatico

- **Input condiviso:** tutti i job BWA leggono i reads e il reference da /mnt/shared/reads/ e /mnt/shared/genome.fa.
- **Output condiviso:** tutti i risultati .bam vengono scritti su /mnt/shared/output/.

💡 Questo modello consente il **parallelismo** nei job e facilita il **monitoraggio dei risultati**.

🚫 5. Problematiche comuni

Problema	Descrizione	Soluzione	📄
Single point of failure	Se il server NFS si guasta, tutti i nodi perdono l'accesso.	Redundancy o replicazione	
Bottleneck I/O	Accesso simultaneo può rallentare il server.	Bilanciamento o cache locale	
Permission mismatch	UID/GID diversi tra server e client.	Sincronizzazione utenti	

6. Collegamenti con altri concetti

- **HTCondor / Slurm:** i job inviati da diversi nodi lavorano su file condivisi (NFS).
- **Docker e Volume Sharing:** anche i container montano volumi condivisi per persistente accesso ai dati.
- **Cloud Storage:** concetto simile, ma con interfacce REST/HTTP.

Tipo File System	Visibilità	Uso tipico	Esempio	📄
Locale	Solo nodo locale	Testing, piccole analisi	/home/user/data	
NFS condiviso	Multi-nodo	HPC, analisi NGS parallela	/mnt/shared/genome.fa	


SEZIONE 3-Cloud Computing: Infrastructure as a Service (IaaS)

1. Definizione del concetto

Il **Cloud Computing** consente di utilizzare **risorse IT (server, storage, rete)** su richiesta, via Internet, senza doverle possedere fisicamente.

Un modello chiave è:

IaaS (Infrastructure as a Service): fornisce infrastruttura virtuale (VM, rete, storage) come servizio, scalabile e configurabile dall'utente.

 2. I livelli del Cloud		
Modello	Cosa offre	Esempio
IaaS	Infrastruttura (VM, storage)	AWS EC2, OpenStack
PaaS	Ambiente runtime per app	Google App Engine
SaaS	Software già pronto	Google Drive, Dropbox

Il focus qui è sull'IaaS: crei macchine virtuali (VM), scegli CPU, RAM, disco, e gestisci tutto come se fosse un tuo datacenter.

3. Come funziona IaaS

Componenti principali:

- **VM (Virtual Machines)**: istanze Linux/Windows accessibili via SSH.
- **Storage virtuale**: dischi permanenti e volatili (es. /dev/vda).
- **Network**: indirizzi IP pubblici, firewall, NAT.
- **API**: accesso programmabile (es. OpenStack CLI o AWS SDK).

4. Esempio pratico nel contesto bioinformatico

Supponiamo tu voglia allineare reads su BWA usando il cloud:

1. Avvii una VM su OpenStack con 4 core, 16 GB RAM, 100 GB disco.
2. Usi scp o rsync per trasferire dati .fastq e genome.fa.

3. Lanci lo script BWA come se fossi su un HPC.
4. Al termine, spegni la VM → **paghi solo il tempo usato**.

💰 **Vantaggio**: nessun costo fisso.

☑ **Scalabilità**: puoi creare 10 VM identiche per elaborare dataset diversi in parallelo.

5. Sicurezza e isolamento

- Ogni utente ha **tenant separato** (spazio isolato).
- I dati non sono visibili ad altri.
- Accesso solo tramite **chiavi SSH**, senza password.
- Firewall e regole di sicurezza configurabili da interfaccia web o riga di comando.

6. Collegamenti con altri concetti

- **Container (es. Docker)**: possono essere usati **dentro** le VM IaaS per ambienti riproducibili.
- **File System condiviso**: è possibile montare un NFS o usare storage oggetti (S3, Swift).
- **HTCondor su cloud**: si può installare un job scheduler sulle VM per simulare un cluster.

Caratteristica	IaaS
Controllo	Alto: scegli hardware e OS
Flessibilità	Alta: crei/spegni VM a piacere
Costo	Pay-per-use
Esempi	AWS EC2, GCP Compute, OpenStack

Amazon Web Services (AWS) – Esempio pratico di IaaS

- ☐ AWS è il provider di cloud più usato in bioinformatica.
- ☐ Offre servizi IaaS come:
 - **EC2 (Elastic Compute Cloud)** = VM personalizzabili

- **S3 (Simple Storage Service)** = storage oggetti
- **Batch** = schedulatore per HTC sul cloud
- **CloudFormation** = per automatizzare infrastrutture

Avvio istanza EC2 da AWS CLI

```
aws ec2 run-instances --image-id ami-xyz --instance-type t2.large --key-name mykey --security-groups mysg
```

AWS permette di lanciare pipeline bioinformatiche on-demand, con Docker + volumi S3.

SEZIONE 4-Batch Systems e HTCondor

1. Definizione del concetto

In ambito HPC o cloud, quando molteplici job devono essere eseguiti in modo automatico e coordinato su più nodi, si usano i **sistemi batch**.

Un **Batch System** è un software che gestisce l'**invio**, la **schedulazione**, l'**esecuzione** e il **monitoraggio** di job (processi) su un insieme di risorse computazionali.

2. Caratteristiche chiave di un Batch System

- **Code di esecuzione** (queue): gestiscono i job in attesa.
- **Schedulazione intelligente**: assegna job ai nodi disponibili.
- **Politiche di priorità**: basate su utente, tempo richiesto, risorse.
- **Parallelismo**: consente di eseguire job indipendenti in parallelo.
- **Controllo e monitoraggio**: stato, output, errori dei job.

3. HTCondor – High Throughput Computing

✓ Cos'è?

È un **sistema batch open-source** progettato per eseguire grandi quantità di job distribuiti, ottimizzando l'uso delle risorse anche su macchine non dedicate.

📁 Funzionalità principali

- **Job seriali o paralleli**.
- **File di sottomissione .sub** per descrivere ogni job.

- **Schedulazione automatica** su CPU disponibili (inclusi laptop o cloud).
- **Supporto per checkpoint e resume.**

4. Esempio pratico – File .sub (dalla pratica nel GitHub BDP1_2025)

executable = run_bwa.sh

output = log/job.out

error = log/job.err

log = log/job.log

queue

condor_submit job.sub # invia il job

condor_q # stato dei job

condor_rm <jobID> # rimuove un job

condor_status # mostra i nodi attivi

5. Esempio reale nel workflow bioinformatico

1. 100 file .fastq → 100 job HTCondor → 100 esecuzioni parallele di BWA.
2. Output .sam → salvati in /mnt/shared/output/.
3. Risparmio di tempo ed efficienza computazionale elevata.

Componente	Descrizione
<code>.sub</code> file	Specifica comando, I/O, log
<code>condor_submit</code>	Invia job al sistema
<code>condor_q</code>	Visualizza job in attesa/esecuzione
<code>condor_status</code>	Mostra i nodi disponibili e loro stato
<code>condor_rm</code>	Cancella un job in coda

7. Collegamenti con altri concetti

- **File System Condiviso (NFS):** essenziale per condividere input/output tra i job.
- **Cloud (IaaS):** Condor può essere installato su VM per creare cluster temporanei.

- **Container:** ogni job può lanciare un container Docker per garantire riproducibilità.

SEZIONE 5-Distributed File Systems (DFS)

1. Definizione del concetto

Un **Distributed File System (DFS)** consente di **salvare, accedere e gestire file distribuiti** su più nodi, in modo **trasparente per l'utente**.

È progettato per sistemi su larga scala (es. cluster, cloud), dove i file devono essere disponibili ovunque, indipendentemente dal nodo fisico.

2. Caratteristiche principali

- **Distribuzione trasparente:** l'utente lavora come se i file fossero locali, anche se fisicamente distribuiti.
- **Replica dei dati:** per aumentare **tolleranza ai guasti**.
- **Scalabilità:** supporta migliaia di nodi.
- **Accesso concorrente:** più utenti/processi possono accedere ai dati.

Sistema DFS	Descrizione	Uso tipico
HDFS	Hadoop Distributed File System	Big data, analisi MapReduce
CephFS	File system distribuito open-source	Cloud, HPC, container
GlusterFS	DFS scalabile open-source	Storage condiviso su cluster
Lustre	DFS ad alte performance	Supercomputer, bioinformatica

4. Come funziona (concetti chiave)

- **Client:** accede ai file tramite una mount locale.
- **Metadata server:** gestisce la struttura del file system (nomi, permessi, blocchi).
- **Storage nodes:** conservano i dati veri e propri, spesso in blocchi (es. HDFS → blocchi da 128MB).

☞ Quando apri un file, il client contatta il metadata server per sapere dove sono i blocchi → li scarica dagli storage node.

5. Esempio nel contesto bioinformatico

- Dataset NGS da 1 TB → diviso in blocchi replicati su 10 nodi.
- Tutti i nodi Condor leggono il dataset **simultaneamente**, senza congestionare un solo server (come accade in NFS).
- 💡 **Performance migliorata + fault tolerance** → se un nodo fallisce, i dati sono replicati altrove.

Caratteristica	NFS	DFS (es. HDFS)
Architettura	Centralizzata (client-server)	Distribuita (peer-to-peer o master/slave)
Fault Tolerance	Bassa	Alta (con replica)
Scalabilità	Limitata	Alta (aggiungi nodi)
Prestazioni	Rischio di bottleneck	Load balancing e parallelismo

7. Collegamenti con altri concetti

- **HTCondor**: job distribuiti possono usare DFS per scrivere dati in parallelo.
- **Docker/Cloud**: Ceph e Gluster sono spesso montati su container o VM.
- **Data locality**: in sistemi DFS (es. Hadoop), il job viene spostato vicino al dato, non viceversa.

SEZIONE 6-Container e Docker

1. Definizione del concetto

Un **container** è un'unità software che **impacchetta codice, librerie e dipendenze** necessarie per far girare un'applicazione, in modo **isolato, portabile e riproducibile**.

Docker è la piattaforma standard per creare, distribuire ed eseguire container.

Caratteristica	Docker Container	Virtual Machine (VM)
Peso	Leggero (~MB)	Pesante (~GB)
Avvio	In pochi secondi	Più lento (boot completo)
Isolamento	Processo isolato	Sistema operativo completo
Uso	Microservizi, bioinformatica	Cloud, ambienti multipli

I container condividono il kernel del sistema operativo, a differenza delle VM che virtualizzano anche l'OS.

Elemento	Descrizione
Dockerfile	Script per costruire un'immagine
Image	Snapshot portabile dell'ambiente
Container	Istanza in esecuzione di un'immagine
Volume	Directory condivisa host↔container

4. Esempio pratico in bioinformatica

Supponiamo di voler eseguire BWA in un container Docker:

FROM ubuntu:20.04

RUN apt update && apt install -y bwa

ENTRYPOINT ["bwa"]

docker build -t bwa_container .

docker run -v \$(pwd)/data:/data bwa_container mem /data/genome.fa /data/reads.fq

I file sono nella directory `./data` sul computer host, montata nel container.

5. Volumi e I/O

- I **volumi Docker** permettono ai container di leggere/scrivere file nel file system del computer host.
- Essenziali per far sì che input e output dei job bioinformatici (es. .fastq, .sam) vengano salvati.

6. Docker su ambienti condivisi (HPC)

- **Problema:** Docker richiede **permessi di root**, pericoloso su cluster condivisi.
- **Soluzioni:**
 - **udocker:** versione user-space di Docker, eseguibile senza root.
 - **Singularity/Apptainer:** container system specifico per HPC (non nel programma ma correlato).

7. Collegamenti con altri concetti

- **HTCondor:** può lanciare container Docker per garantire ambienti controllati per ogni job.
- **Cloud (IaaS):** puoi lanciare container su VM cloud.
- **NFS/DFS:** container possono montare volumi da file system condivisi per elaborare dati in rete.

Concetto	Docker
Portabilità	Massima, eseguibile ovunque
Isolamento	Isola librerie, ambienti, tool
Volumi	Per accesso a file di input/output
Comando base	<code>docker run</code> , <code>docker build</code>

SEZIONE 7-High Performance vs High Throughput Computing (HPC vs HTC)

Argomento chiesto molto

1. Definizione dei modelli

- **HPC (High Performance Computing)**
→ Esegue **un singolo job complesso** usando tanti core/processori contemporaneamente (job parallelo, spesso MPI).
✓ Ideale per simulazioni fisiche, modellistica molecolare, clustering massivo.
- **HTC (High Throughput Computing)**
→ Esegue **molti job indipendenti** in parallelo (job seriali).
✓ Ideale per analisi bioinformatiche massive (es. allineamento reads, BLAST).

Caratteristica	HPC	HTC
Tipo di job	Parallelo (1 job su molti core)	Seriali (molti job piccoli)
Esempio tipico	Simulazione molecolare (GROMACS)	1000 job BWA o BLAST indipendenti
Obiettivo	Velocità per singolo job complesso	Volume di job processati
Infrastruttura	Cluster con nodi collegati (MPI)	Cluster/cloud con nodi indipendenti
Tool comune	MPI, Slurm	HTCondor, Grid Engine

3. Esempi pratici

HPC:

- Simulare l'interazione tra proteine in GROMACS con 64 core.
- Lavoro strettamente parallelo: tutti i core lavorano sullo stesso problema.

HTC:

- Allineare 10.000 reads con BWA, uno per job.
- Ogni job è **indipendente**, può essere eseguito su qualsiasi nodo libero.



4. Applicazioni in bioinformatica

Scenario	Preferibile
RNA-seq, WGS, epigenetica	HTC
Simulazioni dinamiche (proteine)	HPC
Modellazione strutturale (AlphaFold)	HPC
Analisi di metilazione su 500 pazienti	HTC

5. Collegamenti con altri concetti

- **HTCondor** = framework per HTC
- **Docker** + HTC = esecuzione di job isolati e riproducibili
- **File System (NFS/DFS)** = necessari per input/output paralleli su molti job HTC
- **Cloud (IaaS)** = adatto a entrambi, a seconda del carico richiesto

Modello	Focus	Job tipici	Esempi strumenti
HPC	Performance per job	Simulazioni, MPI	GROMACS, LAMMPS
HTC	Volume di job	BLAST, BWA, RNA-seq	HTCondor, Docker

SEZIONE 8-Supercomputing (Supercalcolo)

1. Definizione

Il **supercalcolo** fa riferimento all'uso di sistemi di calcolo altamente performanti (**supercomputer**) per risolvere problemi **complessi, su larga scala e altamente paralleli**.

È la forma più avanzata di HPC (High Performance Computing), con **migliaia di core CPU/GPU**, interconnessioni ad alta velocità e sistemi di archiviazione dedicati.

2. Architettura tipica di un supercomputer

- **Nodi di calcolo:** unità che eseguono i job (CPU, RAM, GPU).
- **Nodi di login:** accesso remoto, preparazione job.
- **Interconnessione veloce:** rete ad alte prestazioni (es. InfiniBand).
- **Storage parallelo:** file system come Lustre, GPFS o BeeGFS.
- **Batch system:** software come Slurm o PBS per gestire i job.

Nome	Paese	Caratteristiche principali	
Leonardo	Italia	Basato su GPU, usato per AI e scienze	
Marconi100	Italia	CPU + GPU, in uso presso CINECA	
Frontier	USA	Exascale, 8 milioni di core	
LUMI	Finlandia	Uno dei più potenti in Europa	

4. Applicazioni in bioinformatica

- **Simulazioni molecolari** (GROMACS, NAMD)
- **Predizione strutturale** con AlphaFold-Multimer
- **Assemblaggio de novo di genomi complessi**
- **Training di modelli di deep learning** su dati omici

Esempio: l'uso di Leonardo/Cineca per predire strutture proteiche in malattie neurodegenerative con AlphaFold 2.0.

5. Modalità di accesso

- Accesso via **SSH** a nodi di login
- Invio job tramite **Slurm** (es. sbatch job.slurm)
- File system condiviso ad alte prestazioni (es. /lustre)

Esempio Slurm:

```
#!/bin/bash
```

```
#SBATCH --job-name=gromacs_sim
```

```
#SBATCH --nodes=2
```

```
#SBATCH --ntasks-per-node=32
```

```
#SBATCH --time=02:00:00
```

```
module load gromacs
```

```
srun gmx mdrun -s topol.tpr
```

6. Collegamenti con altri concetti

- **HPC** = categoria generale; **supercomputing** = forma estrema.
- **Batch system** come Slurm = fondamentale per gestire job su migliaia di core.
- **Cloud (IaaS)** può essere usato per simulare ambienti HPC, ma meno efficiente per job MPI complessi.
- **DFS** come Lustre = necessario per I/O parallelo.

Concetto	Descrizione
Supercomputer	Cluster HPC ultra-performante
Esempi software	GROMACS, NAMD, AlphaFold
Gestione job	Slurm, PBS, LSF
Accesso	SSH + script batch (<code>sbatch</code>)
Storage	File system parallelo (Lustre)

SEZIONE 9-Virtualizzazione (VM vs Container)

1. Definizione

La **virtualizzazione** è una tecnologia che consente di **creare ambienti isolati** che simulano sistemi informatici completi o parziali, utili per testare, eseguire software, e garantire portabilità tra sistemi diversi.

Due approcci principali:

- **VM (Virtual Machine)** → simula un intero computer
- **Container (es. Docker)** → simula un'applicazione isolata

2. Virtual Machine (VM)

Caratteristiche:

- Esegue un intero **sistema operativo guest**.
- Richiede un **hypervisor** (es. VirtualBox, KVM).
- Isolamento completo → maggiore sicurezza.
- Avvio più lento, più consumo di risorse.

Esempi:

- VM Ubuntu con 8 core e 16 GB RAM lanciata su OpenStack o AWS EC2.
- VM con installati tools bioinformatici per esperimenti riproducibili.

3. Container

Caratteristiche:

- Esegue un'applicazione con tutte le sue dipendenze, **senza OS separato**.
- Avvio rapidissimo, uso efficiente delle risorse.
- Meno isolamento (condivide il kernel del sistema host).

Esempi:

- Container Docker con BWA, Python, R, Jupyter ecc.
- Pipeline di RNA-seq eseguita in ambienti containerizzati identici su HPC o cloud.



4. Confronto diretto

Aspetto	Virtual Machine (VM)	Container (Docker)
Isolamento	Completo (OS separato)	Parziale (condivide kernel)
Risorse	Alto consumo	Leggero
Tempo di avvio	Lento (~minuti)	Veloce (~secondi)
Portabilità	Media (più pesante)	Alta (file <code>.tar</code> o immagine)
Sicurezza	Maggiore	Minore in ambienti multiutente
Ideale per	Ambienti complessi	Microservizi, pipeline bioinfo



5. Applicazioni in bioinformatica

Caso d'uso	Scelta ideale
Simulare HPC locale su cloud	VM
Eseguire job riproducibili	Docker container
Offrire ambienti di sviluppo	VM + Docker
Automatizzare pipeline	Docker

6. Collegamenti con altri concetti

- **Cloud (IaaS)**: lancia VM che poi ospitano container Docker.
- **HTCondor**: può schedulare job eseguiti in container per garantire coerenza ambientale.
- **DFS/NFS**: sia VM che container accedono a dati condivisi tramite volumi montati.

Tecnologia	Isolamento	Avvio	Risorse	Uso tipico
VM	Completo	Lento	Alto	Ambienti complessi su cloud
Container	Parziale	Veloce	Basso	Bioinformatica, DevOps

SEZIONE 10-Edge & Fog Computing

1. Definizione

Edge Computing e **Fog Computing** sono modelli architetturali che **portano l'elaborazione dei dati più vicino alla fonte** (es. sensori, dispositivi IoT, macchine di laboratorio).


L'obiettivo è **ridurre la latenza, risparmiare banda** e abilitare decisioni in tempo reale, senza passare sempre dal cloud centrale.

2. Edge Computing

- L'elaborazione avviene **direttamente sul dispositivo locale** o molto vicino ad esso (es. router intelligente, Raspberry Pi, sequenziatore portatile).
- **Esempi:**
 - Analisi preliminare dei dati su un MinION (Oxford Nanopore).
 - Riconoscimento immagini su droni o microscopia con AI integrata.

3. Fog Computing

- Strato **intermedio** tra il cloud e l'edge.
- I dati passano da dispositivi locali → **fog node** (es. micro-server locale) → cloud.
- **Esempi:**
 - Laboratori distribuiti: preprocessing su server locale, analisi finale su cloud.
 - Ospedali: dati da pazienti filtrati localmente prima di invio al cloud centrale.

Modello	Dove avviene il calcolo	Vantaggio principale	Esempio	
Cloud	Centro dati remoto	Grande potenza	Analisi batch NGS	
Fog	Nodo intermedio (locale)	Ridotta latenza	Preprocessing in laboratorio	
Edge	Direttamente sul dispositivo	Tempo reale, privacy	Sequenziatore portatile	

5. Applicazioni in bioinformatica e sanità

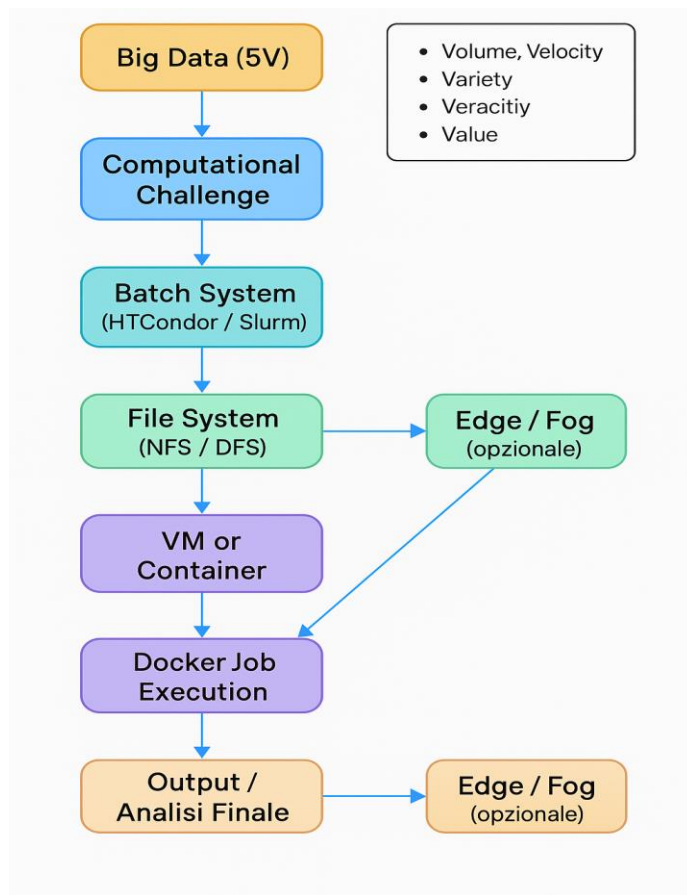
- Sequenziamento **on site** con analisi edge (es. outbreak virale in tempo reale).
- Dispositivi medici smart che analizzano e inviano solo dati critici.
- Analisi pre-filtrata nei laboratori prima del trasferimento a un supercomputer.

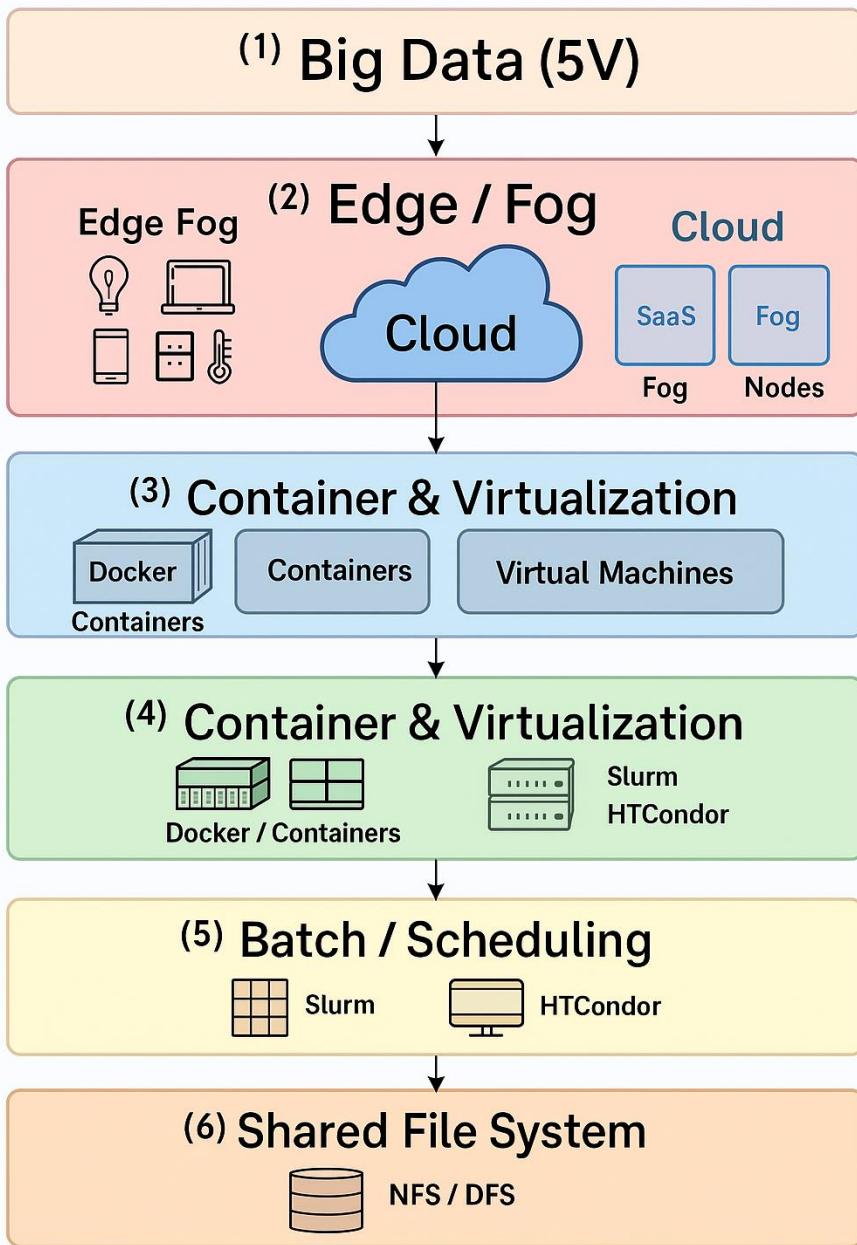
6. Collegamenti con altri concetti

- **Supercomputing e cloud** = elaborazione centralizzata.
- **Edge/Fog** = decentralizzazione → elaborazione vicino ai dati.
- **Docker su edge**: possibile eseguire container anche su dispositivi ARM-based.
- **DFS/NFS**: meno adatti in ambienti edge → preferiti sistemi locali o stream processing.

Modello	Posizione elaborazione	Latenza	Uso tipico	
Cloud	Remoto	Alta	Analisi intensive post raccolta	
Fog	Locale (vicino ai dati)	Media	Filtraggio/aggregazione	
Edge	Sul dispositivo	Bassa	Diagnosi in tempo reale	

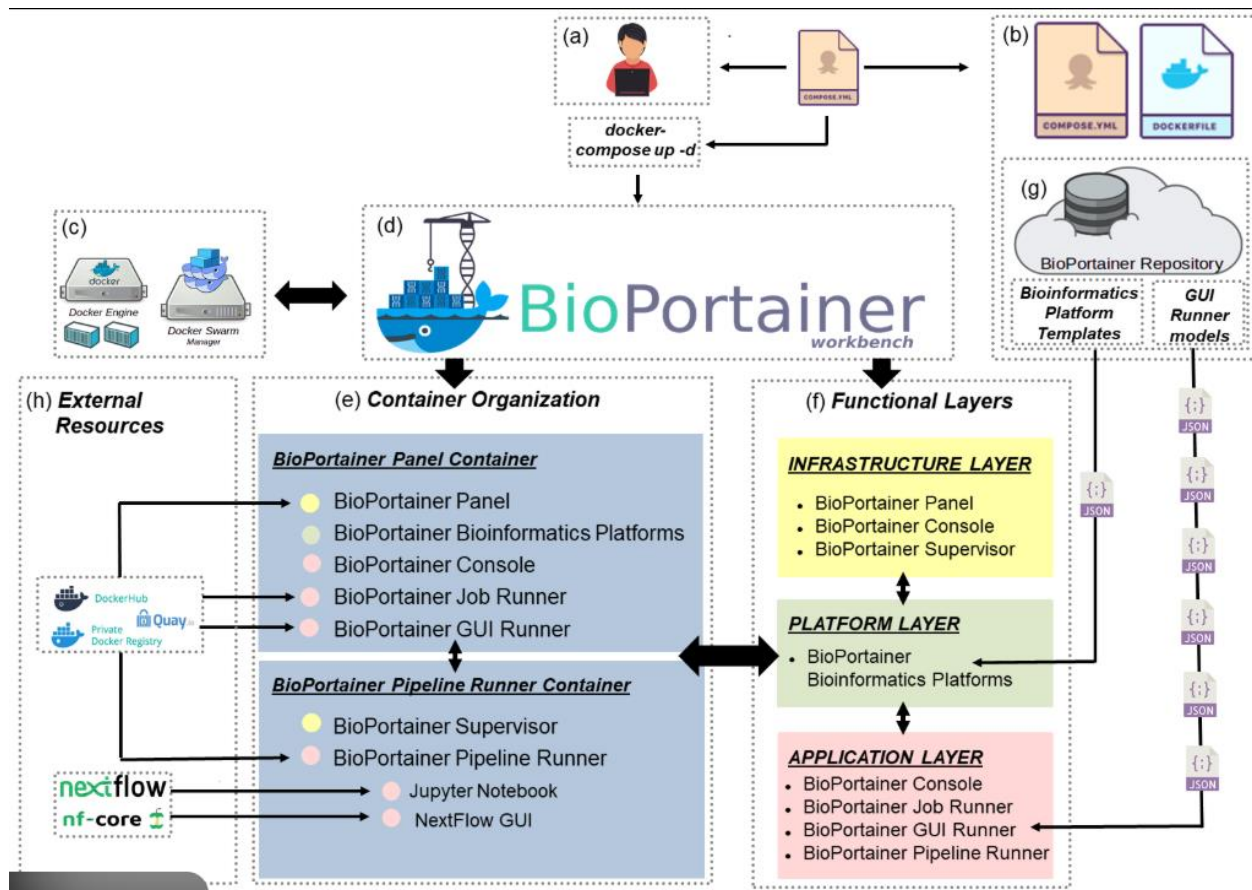
1. **Big Data (5V)**
→ origine del problema: grandi quantità di dati biologici, veloci, variabili e da interpretare.
2. **Computational Challenge**
→ necessità di strumenti efficienti per elaborare dati genomici (es. allineamento reads).
3. **Cloud / HPC / HTC**
→ infrastruttura scelta in base al tipo di analisi: batch massivo (HTC) o simulazione pesante (HPC).
4. **Batch System (HTCondor / Slurm)**
→ gestisce i job distribuiti, automatizzando l'esecuzione su cluster.
5. **File System (NFS / DFS)**
→ permette l'accesso condiviso ai dati da più nodi o container.
6. **VM o Container**
→ scelta dell'ambiente esecutivo: VM per isolamento completo, container per portabilità.
7. **Docker Job Execution**
→ esecuzione vera e propria delle pipeline bioinformatiche (es. BWA, GATK, FastQC).
8. **Output / Analisi Finale**
→ produzione dei risultati e loro aggregazione, visualizzazione, interpretazione.
9. **Edge / Fog (opzionale)**
→ nei casi in cui serve analisi real-time o in loco (sequenziamento portatile, AI su dispositivi locali).





schema finale

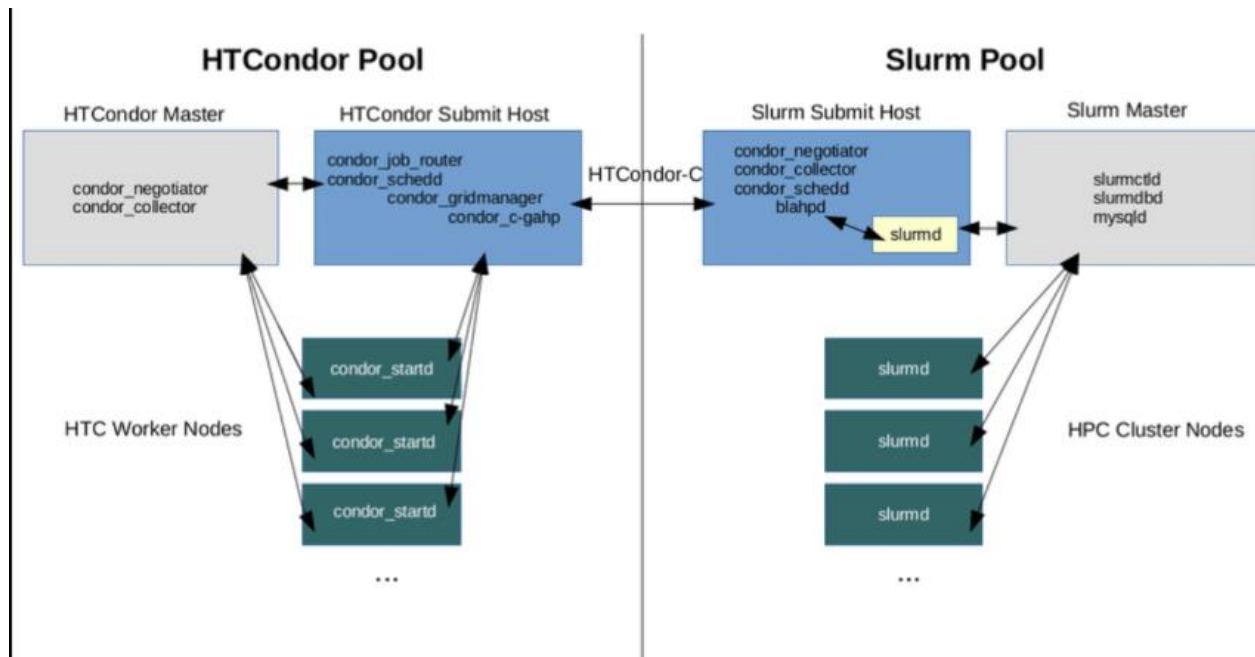
architettura corso



Cosa mostra:

- Un'infrastruttura **containerizzata** basata su **Docker**.
- Include componenti per gestire e visualizzare pipeline bioinformatiche tramite interfaccia grafica (GUI Runner, Jupyter, Nextflow).
- **Usa Docker Compose** per avviare in modo automatico l'intero sistema.

→architetture cloud-native → utile per collegare a **IaaS** e **container orchestration**.

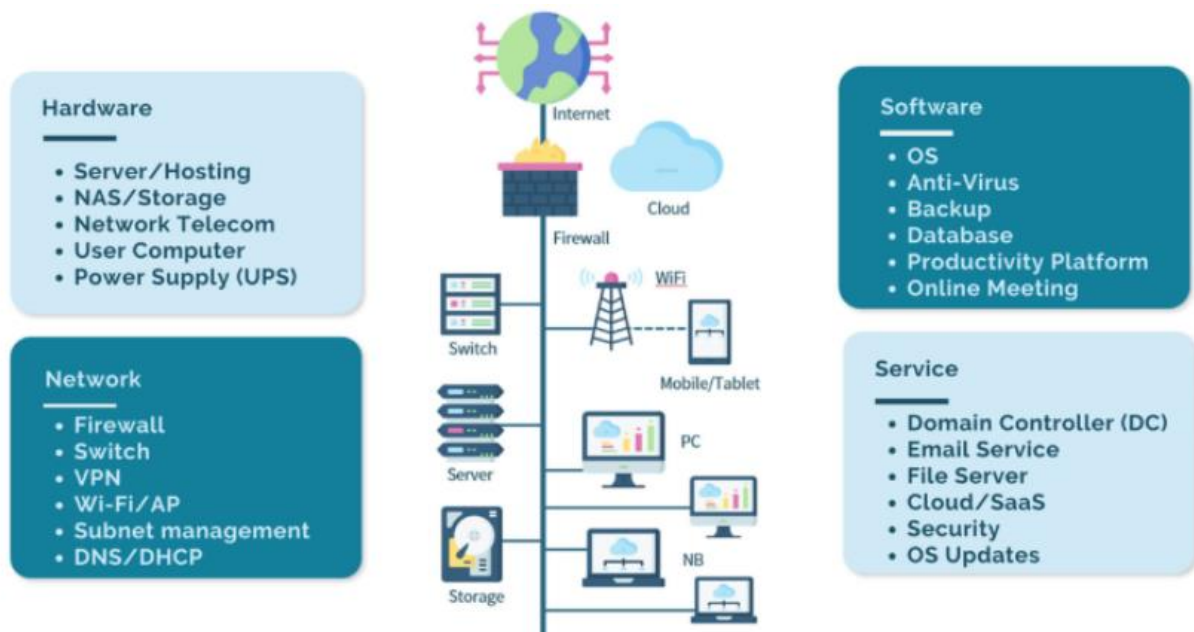


Cosa mostra:

- Architettura comparativa tra **HTCondor (HTC)** e **Slurm (HPC)**.
- Illustra il ruolo di **submit host**, **master**, **worker node**, e i vari demoni (`condor_schedd`, `slurmd`, ecc.).

Da notare:

- HTCondor lavora bene con job piccoli e distribuiti (HTC), mentre Slurm è ottimo per job paralleli su supercomputer.
- L'elemento **HTCondor-C** (bridge tra Slurm e Condor) è fondamentale per ambienti misti.



Cosa mostra:

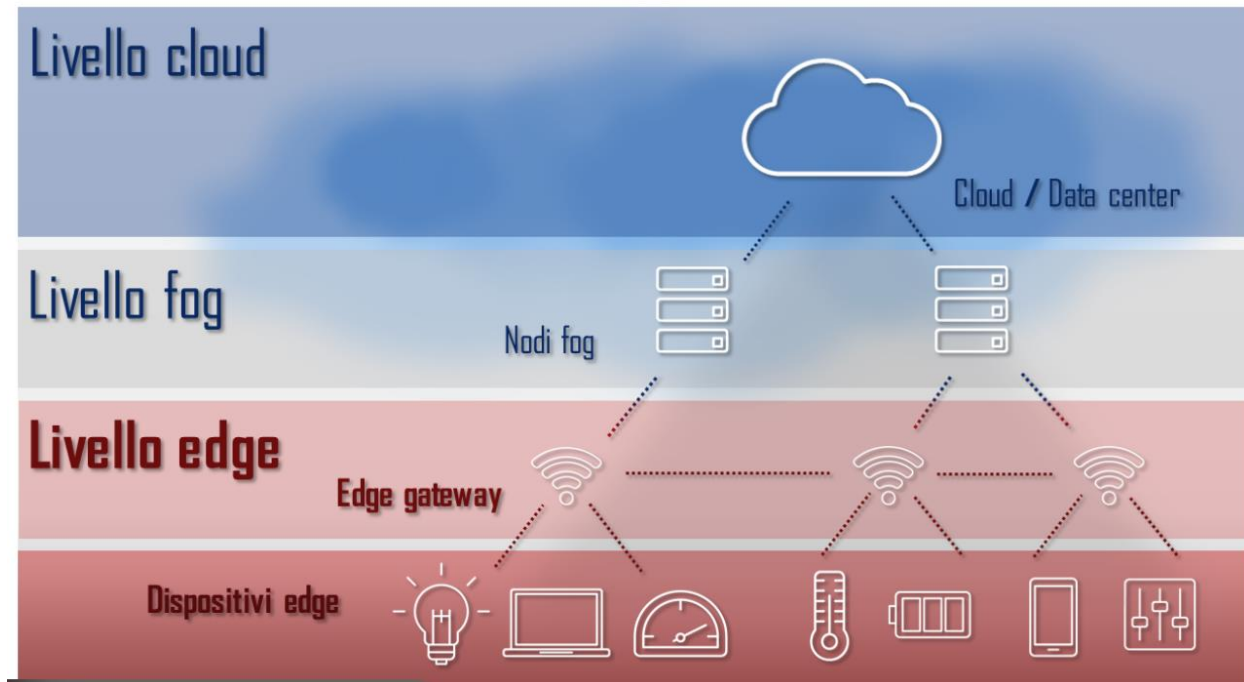
- L'intera architettura IT: **hardware (server, switch), rete, servizi cloud, software.**
- Mostra come tutto è connesso tramite internet e firewall.

Dove si inserisce:

- **Sezione 3 – Cloud Computing (IaaS):** qui vengono istanziati **server, storage e reti** nel cloud.
- **Sezione 5 – Distributed File Systems (DFS):** i file server e NAS si collegano tramite switch/router e sono condivisi via rete.
- **Sezione 9 – Virtualizzazione (VM/Container):** le VM e i container girano su queste infrastrutture.

Da notare:

- Collegamento diretto tra **hardware fisico, servizi cloud (SaaS, IaaS)** e strumenti bioinformatici.
- Il layer infrastrutturale rappresentato è ciò che permette l'esecuzione di Condor, Docker, Slurm, ecc.



Cosa mostra:

- **I tre livelli di elaborazione dati:**
 - **Edge:** dispositivi locali (sensori, computer, laptop)
 - **Fog:** nodi intermedi (gateway)
 - **Cloud:** data center remoto

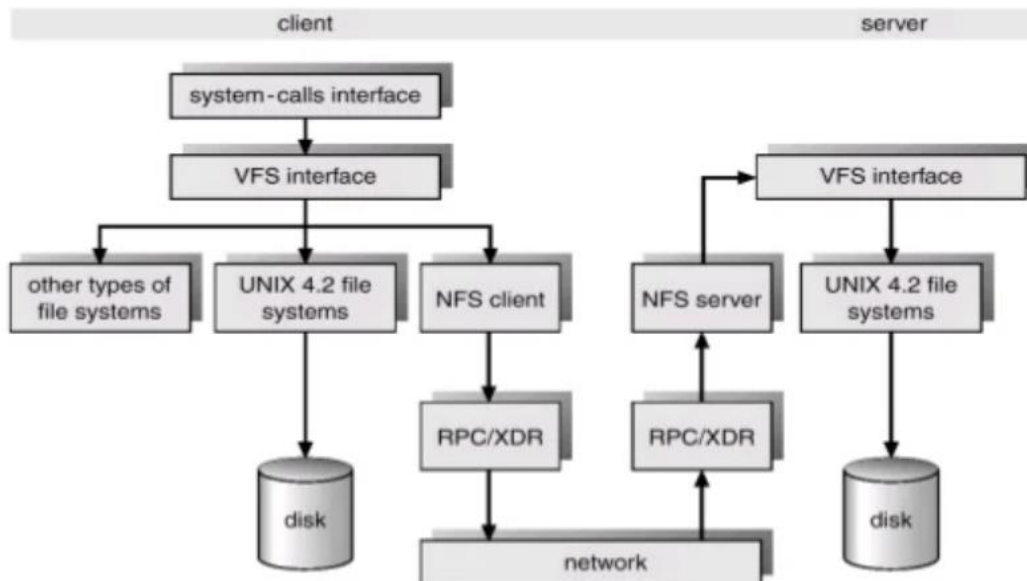
Mostra bene come **i dati si muovono** dal sensore alla nuvola → utile per spiegare latenza, privacy, velocità e analisi real-time.

DISTRIBUTED FILE SYSTEMS

SUN Network File System

NFS ARCHITECTURE:

Follow local and remote access through this figure:



29

Cosa mostra:

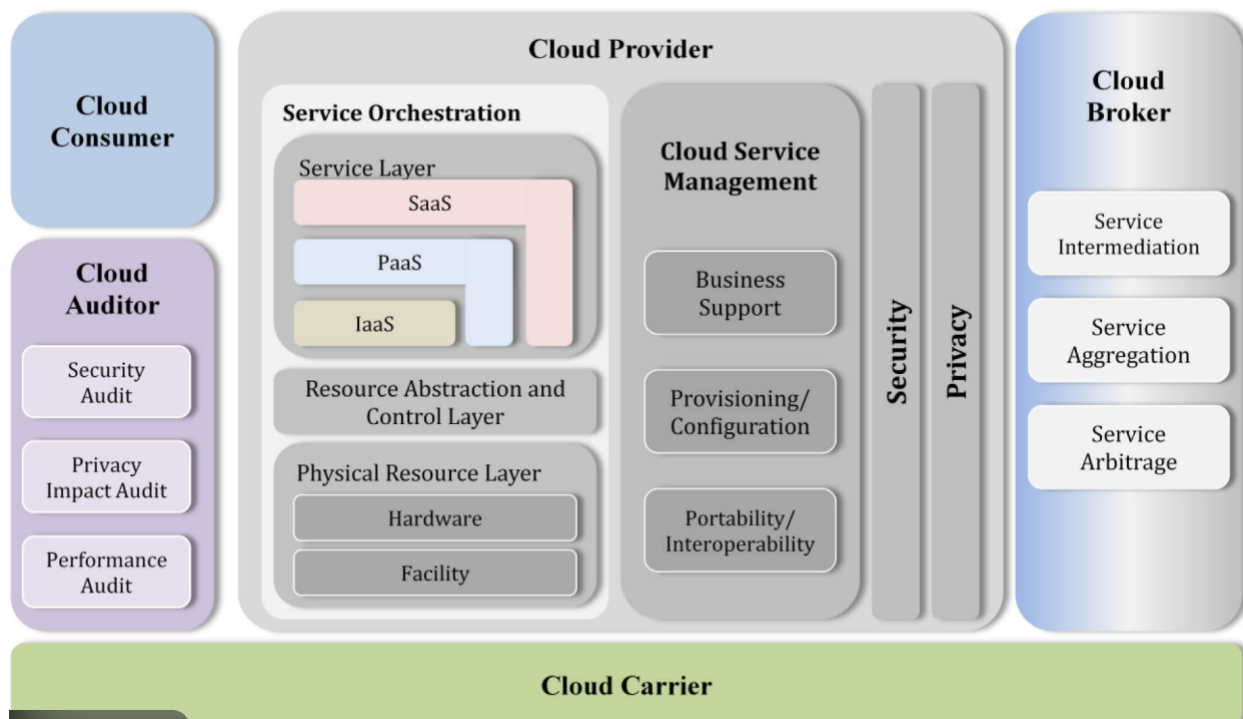
- Architettura di un **NFS (Network File System)**:
 - Lato client/server
 - Interfacce VFS
 - RPC/XDR per trasporto dati

🔗 Collega a:

- **Sezione 5 – Shared File Systems (NFS)**
- **Sezione 8 – Supercomputing** (dove DFS come Lustre sono usati)
- **Glossario:** VFS, RPC, mount

🔗 Da integrare:

- Fondamentale per capire il **modello client/server nei file system condivisi**, che è la base per accessi multi-nodo nei job HTCondor o Slurm.



Cosa mostra:

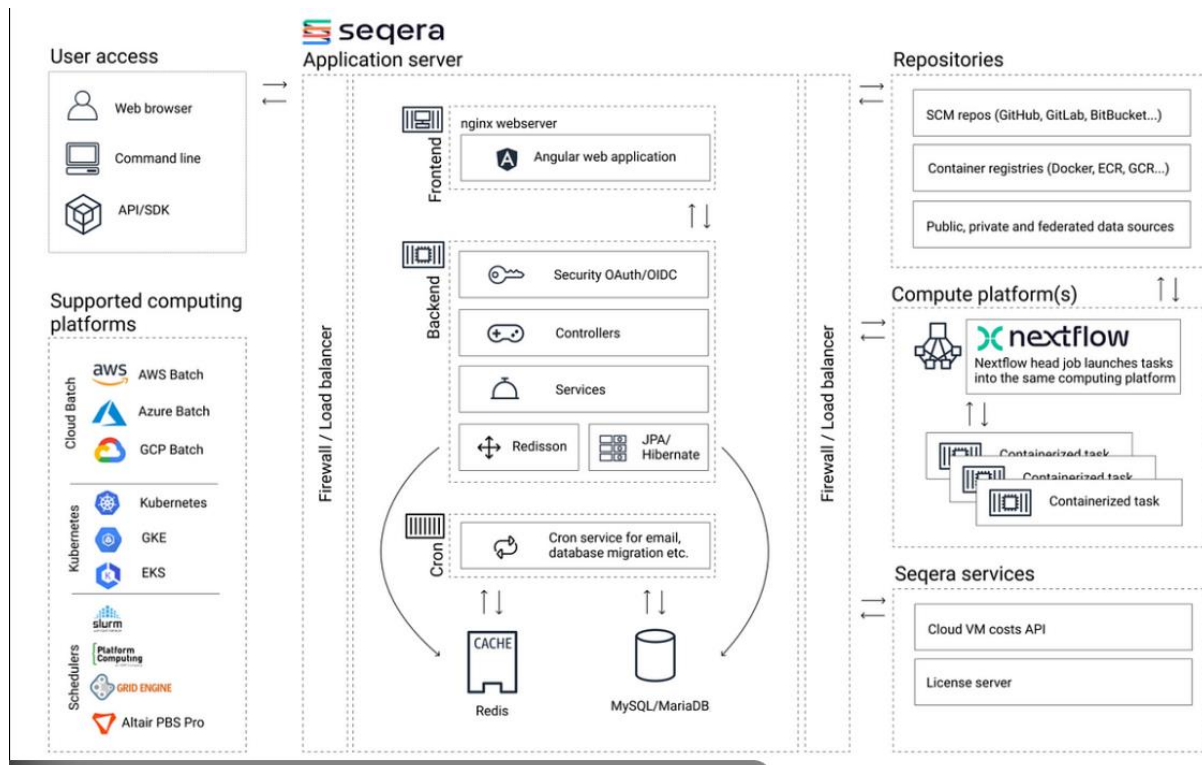
- Tutti gli attori coinvolti in un **ecosistema cloud**: consumer, provider, auditor, broker.
- Il ruolo di **orchestrazione e sicurezza**.

Collega a:

- **Sezione 3 – Cloud**
- **Sezione 9 – Virtualizzazione**
- **Glossario**: Cloud Broker, Cloud Auditor

Da integrare:

- Molto utile per capire **le responsabilità nei modelli cloud** (chi gestisce cosa) → concetto chiave per comprendere differenze IaaS/PaaS/SaaS.



Cosa mostra:

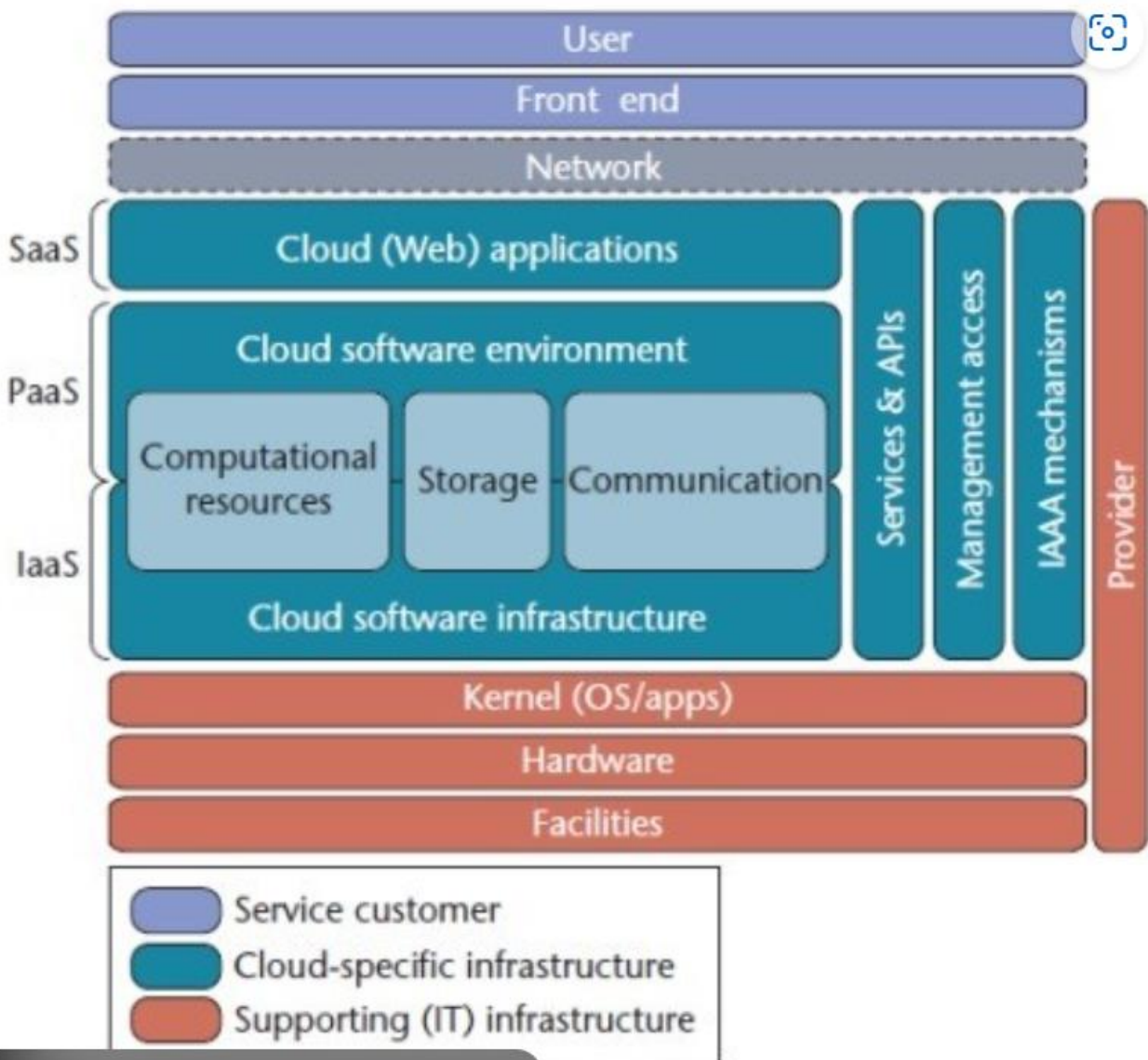
- L'interazione tra **utenti, frontend, backend**, e sistemi di calcolo (Nextflow, Kubernetes, Slurm, AWS Batch...).
- Integrazione con container registries (DockerHub, GitLab) e database (MariaDB, Redis).

Collega a:

- **Sezione 6 – Docker**
- **Sezione 4 – Batch Systems**
- **Sezione 10 – Pipeline/Edge Computing (opzionale)**

Da integrare:

- Illustra come **workflow bioinformatici moderni** vengano orchestrati con strumenti cloud-native (Nextflow), containerizzati e schedulati su Slurm/Condor.



Cosa mostra:

- La **stratificazione dei servizi cloud**: IaaS → PaaS → SaaS.
- IaaS fornisce **infrastruttura virtuale** (es. AWS EC2), PaaS ambienti di sviluppo (es. Google App Engine), SaaS applicazioni pronte all'uso (es. Google Drive).

🔗 Collega a:

- **Sezione 3 – Cloud Computing (IaaS)**
- Glossario (IaaS, API, kernel, frontend/backend)

🔑 Da integrare:

- Utile per distinguere i **livelli di astrazione** nel cloud e capire dove si posizionano strumenti come Docker (su IaaS) o Nextflow (su PaaS/SaaS).
-

LOGICA GENERALE DEL PROGETTO BDP1: DAI BIG DATA ALL'INFRASTRUTTURA ATTIVA

1. IL PROBLEMA: Big Data bioinformatici

- I dati NGS (Next-Generation Sequencing) sono enormi → **volume**, arrivano rapidamente → **velocità**, hanno variabilità biologica → **varietà**.
- Da qui nasce la necessità di:

"Costruire un'infrastruttura scalabile, accessibile e automatizzabile per processare questi dati in modo riproducibile."

→ **Lezione corrispondente**: Sezione 1 + le 5V

→ **Effetto**: Dobbiamo distribuire il carico su più risorse.

2. L'INFRASTRUTTURA: il Cloud (AWS) come base fisica e logica

- L'infrastruttura viene costruita su **AWS**, tramite:
 - **VM (EC2)** → macchine personalizzabili.
 - **Snapshot + volumi** → recupero e riutilizzo di dati.
 - **Sicurezza**: gruppi, porte, autenticazione SSH.

→ **Lezione**: Sezione 3 (IaaS) + 9 (Virtualizzazione)

→ **Scopo**: Avere "mattoni" pronti e flessibili su cui costruire.

3. LA CONDIVISIONE: NFS come sistema nervoso

- Per permettere a più macchine di "vedere gli stessi dati", viene montato un **file system condiviso**:
 - master condivide una directory → worker montano in locale via IP.
 - Controllo via exportfs, chmod, fstab.

→ **Lezione**: Sezione 2 (NFS) + 5 (DFS vs NFS)

→ **Funzione**: Coordinazione tra nodi → tutti accedono ai dati giusti.

□ 4. L'INTELLIGENZA: HTCondor gestisce i job

- Il sistema batch HTCondor agisce da **cervello**:
 - Riceve job (.job)
 - Li assegna ai nodi disponibili
 - Gestisce output, log, errori

→ **Lezione**: Sezione 4 + 7 (HTC vs HPC)

→ **Obiettivo**: Automatizzare il carico di lavoro.

👤 5. L'ISOLAMENTO: Docker containerizza l'ambiente

- Per assicurare che il job sia **riproducibile** su qualsiasi nodo, si tenta l'uso di **Docker**:
 - Si crea un'immagine (Dockerfile)
 - Contiene bwa + Python + align.py

→ **Lezione**: Sezione 6 (Container)

→ **Ostacolo**: Su HTCondor, i container (Docker/udocker) non riescono ad accedere ai dati indicizzati → test fallito.

🌐 6. L'ESTENSIONE: multi-sito e trasferimento dati

- Per simulare un'**architettura geografica distribuita**, si connettono 3 cloud (AWS, GCP):
 - Si usano tool come **WebDav** per trasferire dati tra i siti.
 - Si analizza la **latenza** e il costo per l'allineamento distribuito.

→ **Lezione**: Sezione 10 (Edge/Fog, data mobility)

→ **Visione**: Simulare come una struttura globale possa funzionare su diverse regioni cloud.

🔗 Tutto insieme: un sistema bioinformatico distribuito

1. **Dati biologici grezzi** (big data)

2. **Cloud computing (IaaS)** per creare risorse
 3. **Storage condiviso (NFS)**
 4. **Sistema di job distribuiti (HTCondor)**
 5. **Container per portabilità (Docker)**
 6. **Estensione multi-cloud + WebDav**
 7. **Analisi finale:** tempi, costi, performance
-

Conclusione: cosa hai costruito

Hai realizzato:

- Un **sistema scientifico distribuito**.
- Che può essere **scalato** (più nodi), **replicato** (snapshots), **portato altrove** (cloud).
- Che è **automatizzato, documentato, modulare**.

Ogni **capitolo del corso** ti ha fornito un pezzo del puzzle.