Introduzione a Kubernetes

Architettura e Componenti principali di K8s

Sinisi

Workshop on
management of distributed resources for genomic communities

29-30 October 2024

INFN CLOUD

ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

POLICLINICO DI SANT'ORSOLA

DARE DIGITAL LIFELONG PREVENTION

Ministero dell'Università e della Ricerca

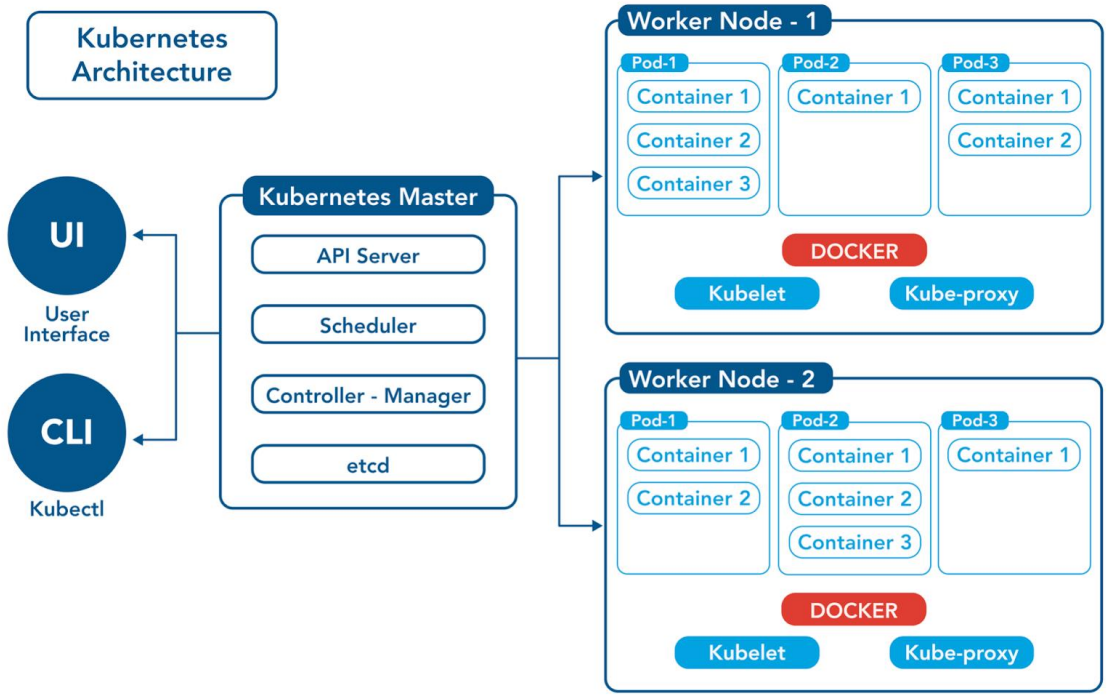
Italiadomani

PNC Piano nazionale per gli investimenti complementari al PNRR
Ministero dell'Università e della Ricerca

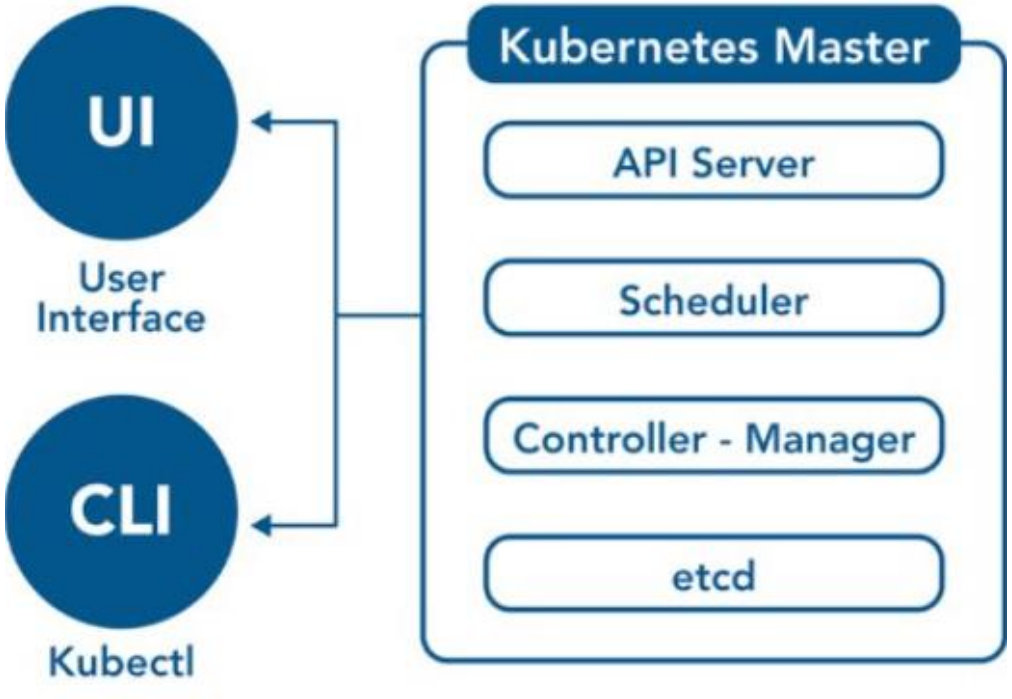
Architettura cluster K8s

Tipi di nodi e componenti principali

Architettura Kubernetes



Tipi di nodi e componenti

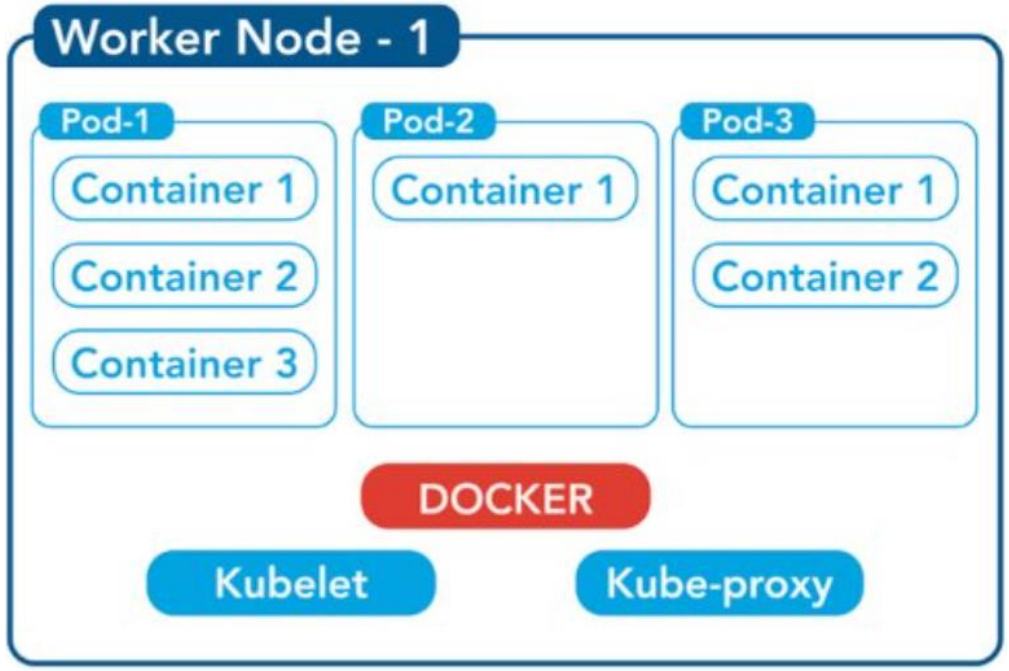


Ci sono 2 tipi di nodi: **controlPlane**, che contengono tutte le componenti principali, **worker**, i nodi che mettono a disposizione le proprie risorse per le applicazioni.

Nel controlPlane troviamo:

- **ControllerManager**: un loop infinito che controlla lo stato del cluster e verifica che corrisponda a quello desiderato dall'amministratore. Per farlo contatta continuamente le API e se non corrisponde intraprende azioni correttive.
- **API Server**: il punto di ingresso del cluster, un'interfaccia per interagire col cluster sia da se stesso (ControllerManager) sia da parte degli utenti (amministratore o developer).
- **ETCD**: affidabile database chiave-valore dove viene salvato lo stato del cluster. Ogni operazione viene scritta nell'ETCD o viene consultato quando richiediamo un'informazione.
- **Scheduler**: si occupa di distribuire i Pod sui nodi (spesso sui worker). Trova il nodi migliore che rispetta i vincoli impostati dall'amministratore, dagli utenti, dallo spazio richiesto dall'applicazione, dallo spazio disponibile sul nodo, ecc.

Tipi di nodi e componenti



Ci sono 2 tipi di nodi: **controlPlane**, che contengono tutte le componenti principali, **worker**, i nodi che mettono a disposizione le proprie risorse per le applicazioni.

Nel worker troviamo:

- **Kubelet**: demone che gira all'interno dei nodi e si occupa della comunicazione verso API Server, l'implementazione dei comandi, verifica lo stato di salute dei Pod.
- **Kube-proxy**: si occupa del networking come l'esposizione delle porte, l'instradamento del traffico ed eventuali regole di networking.
- **Container Runtime**: permette l'esecuzione dei container.

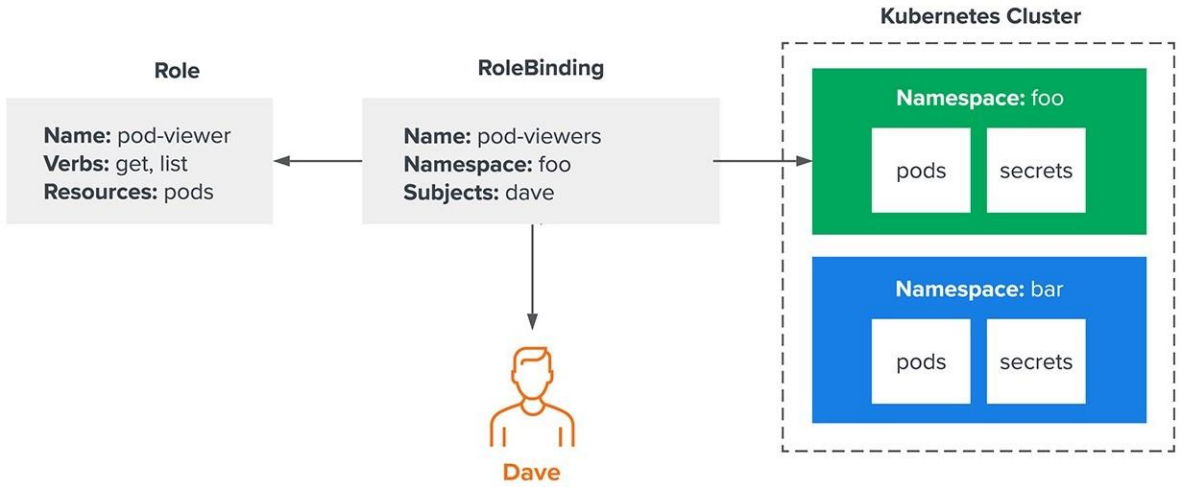
API Server

Come interagire col cluster tramite le API e RBAC

Contattare il cluster

Gli elementi per contattare un cluster sono 2:

- L'eseguibile **kubectl** (scaricabile da questa guida);
- Un file (**kubeconfig**) contenente le coordinate del cluster e la carta d'identità dell'utente che desidera contattarlo.



RBAC

Dopo aver contattato il cluster, la richiesta deve superare il vaglio delle regole RBAC (*chi può fare cosa*), formate dalla coppia:

- **Role**: lista di azioni (GET, CREATE, DELETE, ecc.) che possono essere eseguite su alcune componenti K8s (Pod, Service, Deployment, ecc.)
- **RoleBinding**: lega l'utente al ruolo

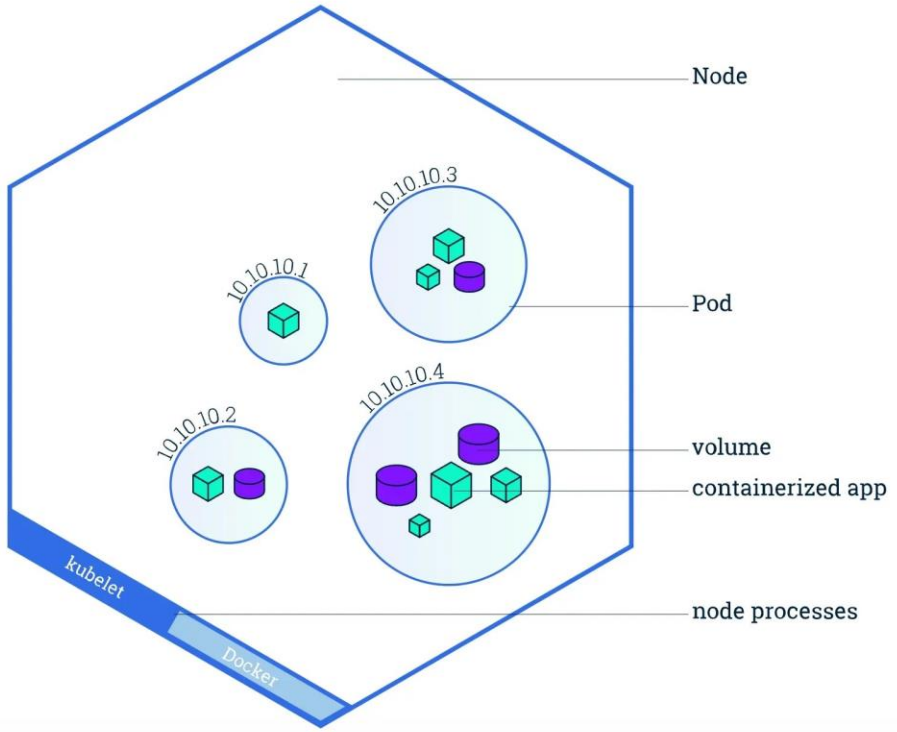
Pod e controller

L'atomo di K8s ed i suoi controllori

Il Pod

I Pod sono lo strumento di base, l'unità più piccola, il quanto di un cluster K8s. Essi:

- Creano un ambiente per i container, che girano al loro interno, mettendo a disposizione rete (indirizzo IP e porte), l'accesso ad uno storage (PVC), CPU e RAM.
- Spesso ospitano un solo container, ma possono contenerne di più, condividendo le risorse tra tutti i container.
- sono un'entità effimera.



Pod (definizione base)

```
apiVersion: v1
kind: Pod
metadata:
  name: <nome_Pod>
spec:
  containers:
    - name: <nome_container>
      image: <immagine>
```

- **apiVersion**: indica la struttura dei campi (nome e alberatura) e dei valori attesi
- **kind**: tipologia di risorsa
- **metadata**: informazioni descrittive della risorsa
- **spec**: descrive gli oggetti ospitati
- **containers**: elenco dei container ospitati all'interno del Pod

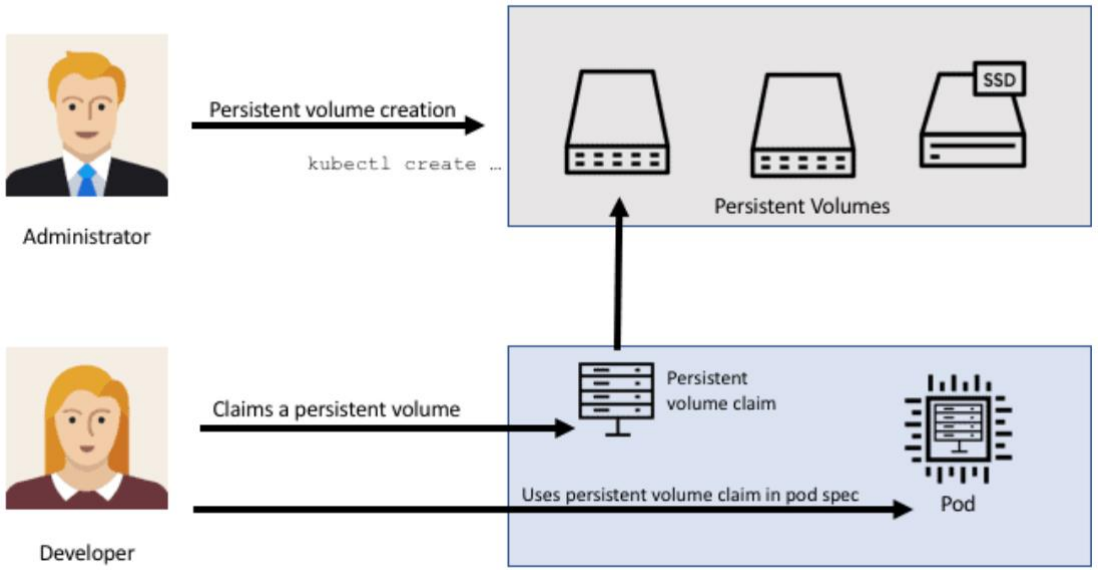
Pod (arricchito)

- **namespace** (**NS**): partizionamento logico dello spazio ed i Pod (e non solo) appartengono ad un NS soltanto. I NS non possono essere annidati.
- **labels**: metadati associabili a oggetti K8s. Sono utilizzate per filtrare, raggruppare delle risorse.
- **resources**: meccanistmi usati da K8s per controllare risorse CPU e RAM
  - **request**: quantità minima di risorse necessarie all'avvio
  - **limits**: tetto massimo di risorse raggiungibili
- **configMap** (**CM**): configurazioni di piccole dimensioni di tipo chiave-valore, che permettono la separazione tra applicativo e configurazione (portabilità). Esse vengono esposte con variabili d'ambiente o, dopo essere montate nel Pod, come file su un certo path.
- **secret**: CM specializzati nel custodire informazioni confidenziali. Essi possono essere di tipi diversi: opaco (generico), chiaviSSH, certificati, ecc.

Volumi

I Pod, così come il loro FS, sono effimeri. Per garantire la persisitenza dei dati usare i volumi:

- **L'amministratore** si occupa di fornire uno storage e di agganciarlo al cluster
- L'**utente** si limita a richiedere un pezzo di storage *(claim)* e montarlo nella propria applicazione



Pod controller

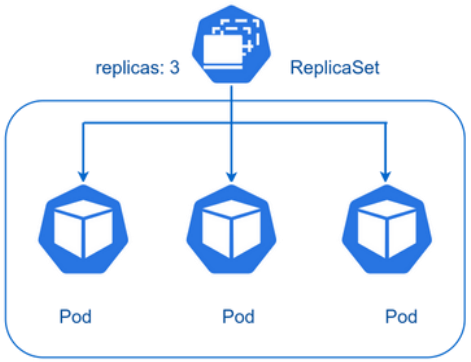


Fig: ReplicaSet

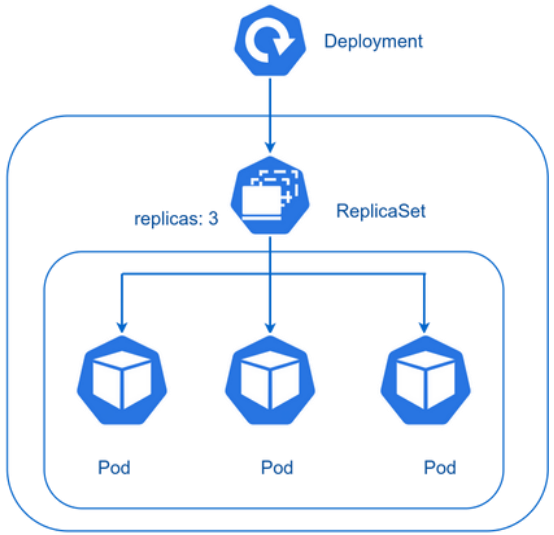


Fig: Deployment #goglides

La natura dei Pod è effimera, perciò sono stati creati dei controller per monitorarne **numero** e **stato di salute**.

- **ReplicaSet** (**RS**): lo scopo è quello di moltiplicare il numero di copie di un certo applicativo (Pod). Questo serve ad aumentare l'affidabilità dell'applicativo in caso di malfunzionamento di una replica e/o per gestire le richieste che non sarebbero digeribili da una singola istanza.
- **Deployment**: si assicura che ci siano un certo numero di repliche (sfruttando il RS) e permette strategie di rollout. Grazie al rollout è possibile sostituire in modo graduale i Pod appartenenti alla vecchia versione del Deployment, coi nuovi. Inoltre è sempre possibile effettuare un rollback alla versione precedente.

Networking

Raggiungere un'applicazione dall'esterno tramite ingress

Ingress

Per raggiungere la nostra applicazione dall'esterno, bisogna seguire il flusso seguente:

- le richieste dell'utente raggiungono l'*ingress controller* presente nel cluster;
- l'ingress controller, sulla base delle istruzioni presenti nell'*ingress resource*, smista il traffico al Service;
- il *Service* (LB interno) reindirizza la richiesta ad uno dei Pod dell'applicazione