Luca Cabibbo

Architettura dei Sistemi Software

# Orchestrazione di container

**dispensa asw670**

ottobre 2024

*You may be wondering what we mean when we say "reliable, scalable distributed systems."*

*Brendan Burns, Joe Beda, and Kelsey Hightower*



## - Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 40, Orchestrazione di container
- Luksa, M. **Kubernetes in Action**. Manning, 2018.
- Richardson, C. **Microservices Patterns**: With examples in Java. Manning, 2019.
- Nygard, M. **Release It!**: Design and Deploy Production-Ready Software, second edition. Pragmatic Bookshelf, 2018.
- Kubernetes Documentation
  https://kubernetes.io/docs/home/
- Docker – Swarm mode overview
  https://docs.docker.com/
  https://docs.docker.com/engine/swarm/

# - Obiettivi e argomenti

- Obiettivi
  - presentare l'orchestrazione di container
  - discutere i problemi affrontati e le soluzioni fornite dall'orchestrazione di container
- Argomenti
  - introduzione
  - orchestrazione di container
  - caratteristiche dell'orchestrazione di container
  - discussione



# * Introduzione

- L'orchestrazione di container ha lo scopo di supportare l'esecuzione in produzione di sistemi software a container
  - è una tecnologia complementare alla virtualizzazione basata su container, che è realizzata dai container manager
  - per applicazioni multi-servizi e multi-container – in cui ogni container incapsula un servizio software
  - in produzione, per motivi di scalabilità e disponibilità, servizi e container devono essere replicati ed eseguiti in un cluster di nodi, fisici o virtuali
    - un container manager è però in grado di occuparsi solo della gestione di container in un singolo nodo – non è adeguato per sostenere scalabilità e disponibilità su molti nodi
  - l'orchestrazione di container ha lo scopo di risolvere questi e altri problemi di interesse per l'esecuzione in produzione di sistemi software a container

# * Orchestrazione di container

- L'orchestrazione di container riguarda la gestione e l'esecuzione in produzione di sistemi software a container
  - consente di definire ed eseguire applicazioni multi-servizi e multi-container (replicati) su un cluster di (molti) nodi
    - ogni container è usato per eseguire un servizio software
    - l'intera applicazione (a container) è definita come una composizione di questi servizi e container, tra loro distribuiti
  - queste funzionalità sono fornite dagli orchestratori – strumenti software per l'orchestrazione di container, che operano a un livello di astrazione superiore rispetto ai container manager
- Presentiamo ora i concetti, gli obiettivi e le caratteristiche principali dell'orchestrazione di container e degli orchestratori di container
  - in modo indipendente (per quanto possibile) dalle implementazioni attuali



# Strumenti di orchestrazione

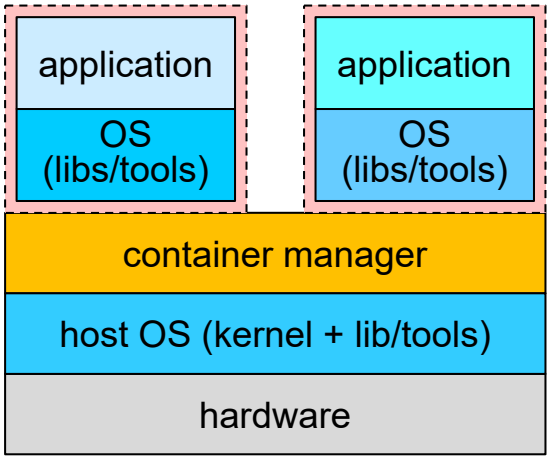
- Alcuni esempi di orchestratori
  - uno *Swarm* (*Docker in Swarm Mode*) è un gruppo di nodi che eseguono Docker e che sono uniti in un cluster – che consente l'esecuzione di applicazioni multi-container e multi-computer
  - *Kubernetes* è un sistema open-source di orchestrazione e gestione di container, inizialmente sviluppato da Google, per l'automazione del rilascio, della scalabilità e della gestione di applicazioni a container
  - *Google Kubernetes Engine* (*GKE*) e *Amazon Elastic Kubernetes Service* (*Amazon EKS*) sono servizi gestiti di orchestrazione di container, basati su Kubernetes
  - *Amazon Elastic Container Service* (*Amazon ECS*) è un servizio di orchestrazione di container altamente scalabile e completamente gestito, per semplificare l'esecuzione e la gestione di container in un cluster

# Orchestrazione di container

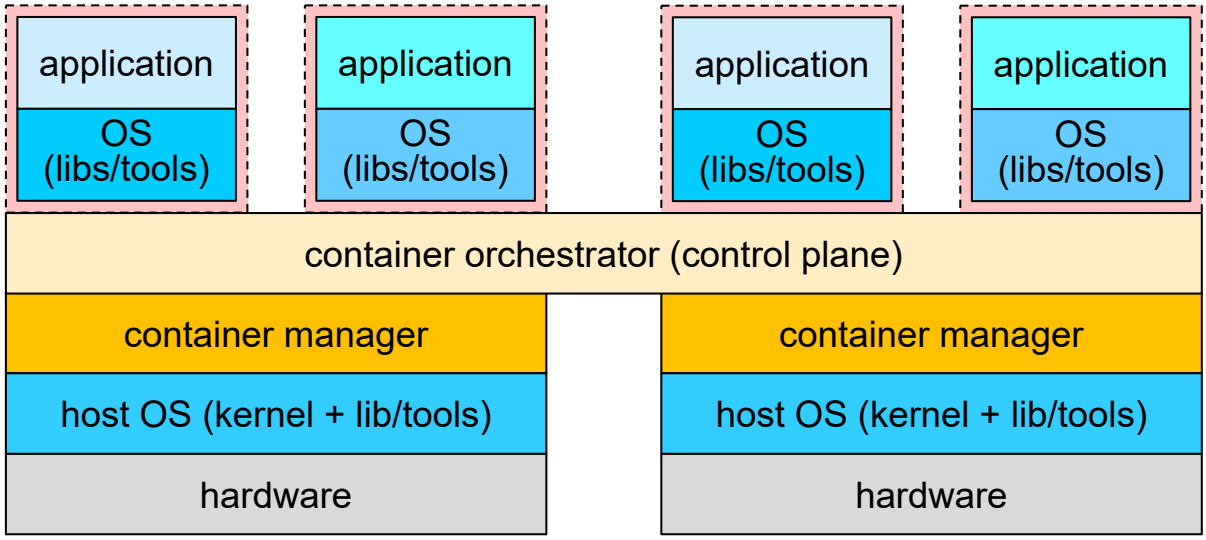
- L'*orchestrazione di container* è la possibilità di definire ed eseguire applicazioni a container (applicazioni contenitorizzate), multi-servizi e multi-container (replicati), in un cluster di (molti) nodi
  - l'orchestrazione di container è supportata dagli strumenti software di orchestrazione (*container orchestrator* o *orchestratori*) che, sulla base di opportune astrazioni, consentono di gestire applicazioni a container in produzione, in modo scalabile e disponibile, in un cluster di nodi (in un data center privato o nel cloud)
  - intuitivamente, un orchestratore è una piattaforma per gestire applicazioni a container in un data center
    - è realizzato come un *control plane* distribuito che gestisce le risorse dei nodi del cluster (in ciascuno dei quali viene eseguito un container manager) e le utilizza per l'esecuzione distribuita e coordinata dei container



# Dalla virtualizzazione basata su container all'orchestrazione di container



# Dalla virtualizzazione basata su container all'orchestrazione di container



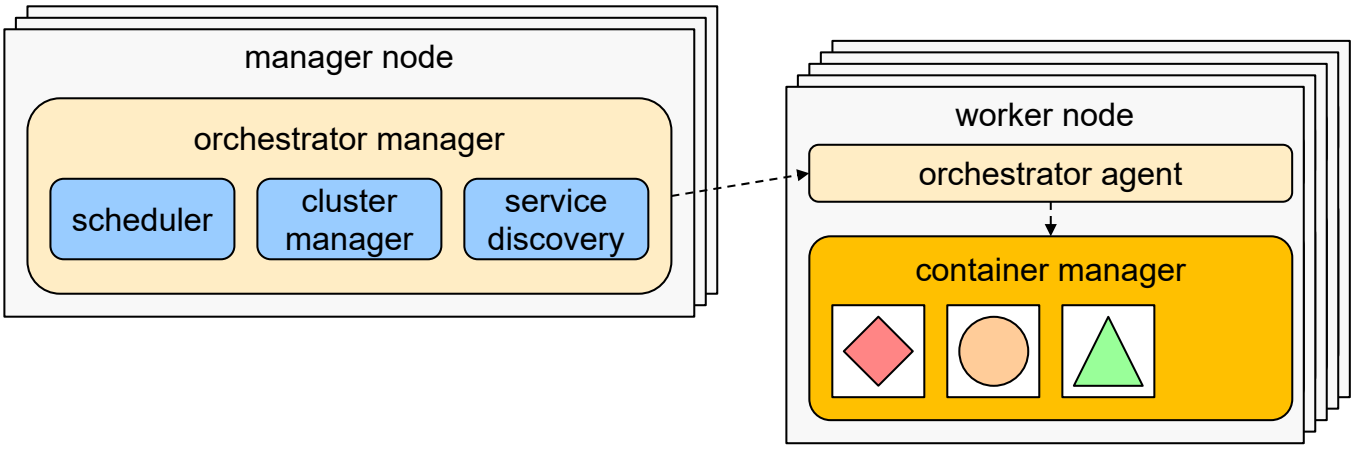



# Orchestrazione come piattaforma

- L'orchestrazione di container realizza una piattaforma per sistemi software a container
  - una piattaforma è, in generale, un ecosistema di risorse per implementare ed eseguire applicazioni software
    - un insieme di strumenti di supporto allo sviluppo di applicazioni
    - un ambiente runtime per l'esecuzione di queste applicazioni
    - le applicazioni devono avere l'architettura richiesta dalla piattaforma
  - discutiamo anche (tra le righe) in che modo l'orchestrazione di container realizza una piattaforma per applicazioni a container

# Architettura di un orchestratore

- Architettura generale di un orchestratore di container




  - due tipi di nodi – per eseguire diversi componenti software



# Architettura di un orchestratore

- Nell'orchestrazione di container, vengono utilizzati due tipi di nodi
  - *nodi manager* o *control plane*
    - si occupano dell'esecuzione dell'orchestratore
    - ricevono (dagli amministratori) richieste di rilascio di applicazioni, che gestiscono schedulando l'esecuzione di container nei nodi worker
    - forniscono anche ulteriori servizi di supporto per la gestione delle risorse del cluster
  - *nodi worker*
    - si occupano dell'esecuzione dei container delle applicazioni
    - gestiscono le richieste (fatte dai nodi manager) di esecuzione di container – che in ogni nodo vengono delegate dall'agente dell'orchestratore al container manager locale

# * Caratteristiche dell'orchestrazione di container

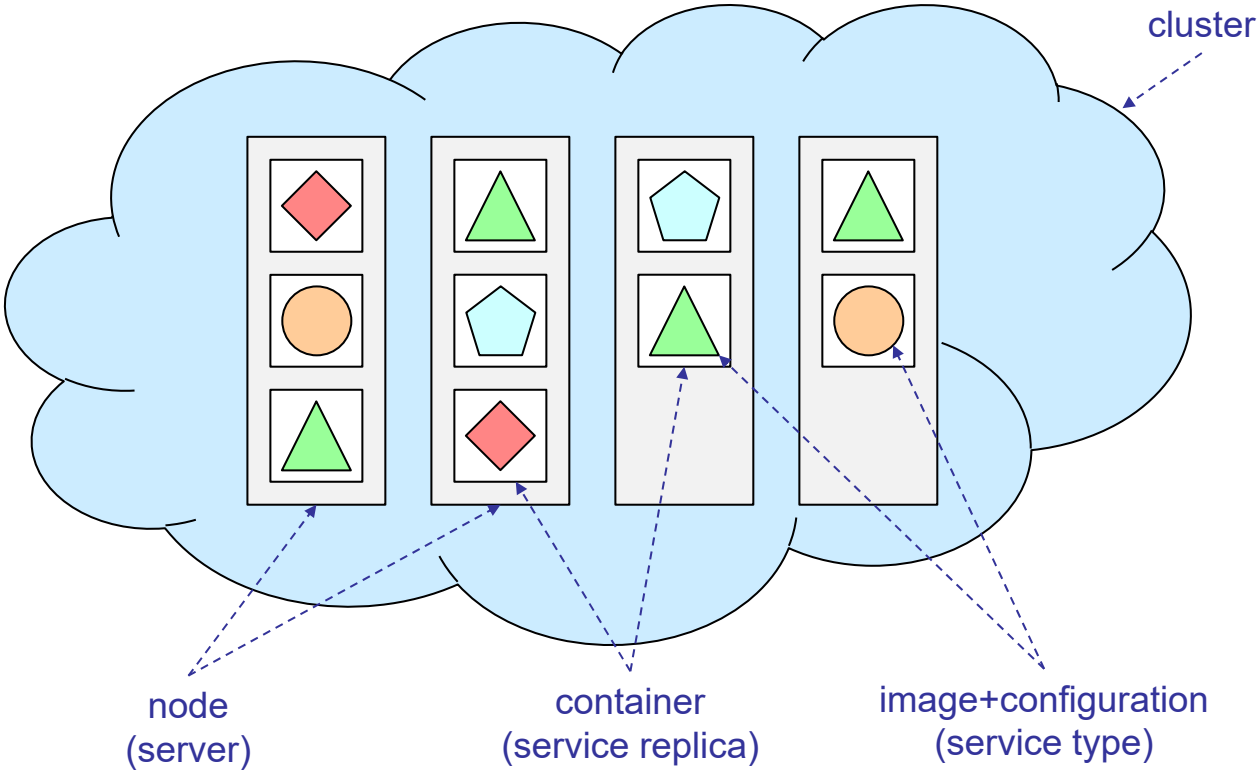
- Descriviamo ora le principali caratteristiche e capacità fornite dall'orchestrazione di container
  - è possibile comprendere queste caratteristiche ragionando sui problemi che è necessario affrontare nella gestione delle applicazioni a container in produzione – e considerando le soluzioni fornite dagli orchestratori



# - Architettura a servizi

- Le applicazioni a container hanno un'architettura a servizi
  - ogni applicazione è composta da più servizi
  - i servizi possono essere replicati
  - i servizi sono eseguiti mediante dei container
    - ogni (tipo di) servizio è rappresentato da un'immagine di container – insieme a una configurazione
    - ciascuna replica di un servizio è rappresentata da un'istanza di container
  - l'orchestratore esegue un'applicazione gestendo i container necessari per l'applicazione nei diversi nodi del cluster

# Architettura a servizi

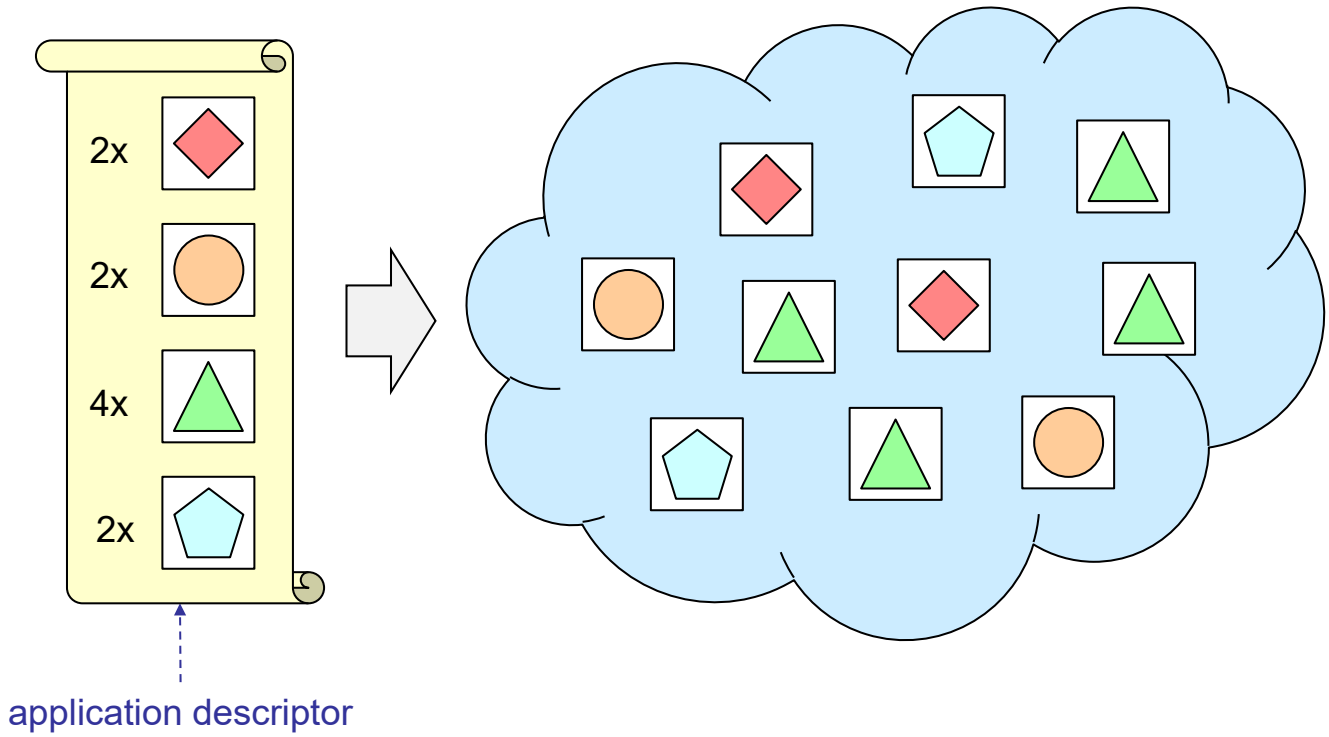





# - Configurazione (composizione) delle applicazioni

- È necessario un approccio per descrivere la configurazione di ogni applicazione di interesse, in modo flessibile e dichiarativo
  - ad es., per specificare i suoi servizi e il numero di repliche richieste per ciascun servizio
  - l'orchestratore definisce delle astrazioni per la configurazione delle applicazioni
    - ad es., "servizio" e "applicazione"
  - inoltre, fornisce un linguaggio dichiarativo per specificare queste configurazioni
  - l'orchestratore utilizza queste configurazioni per rilasciare e gestire le applicazioni nel cluster

# Configurazione (composizione) delle applicazioni






# Configurazione (composizione) delle applicazioni

- Ecco alcune astrazioni di configurazione – Docker Swarm
  - un *service* è definito in termini di un'immagine di container, un comando da eseguire nel container e il numero di repliche richieste (o se il servizio è "globale")
  - a runtime, un *task* è un'istanza di container usata per eseguire un servizio nel cluster
  - uno *stack* è un gruppo di servizi che compone un'applicazione contenitorizzata
    - il rilascio di uno stack ne avvia tutti i servizi e tutti i task
  - l'amministratore interagisce con l'orchestratore definendo e rilasciando stack e servizi
    - a runtime, l'orchestratore schedula i task necessari e interagisce con i container manager nei nodi del cluster per avviare le relative istanze di container

# Configurazione (composizione) delle applicazioni

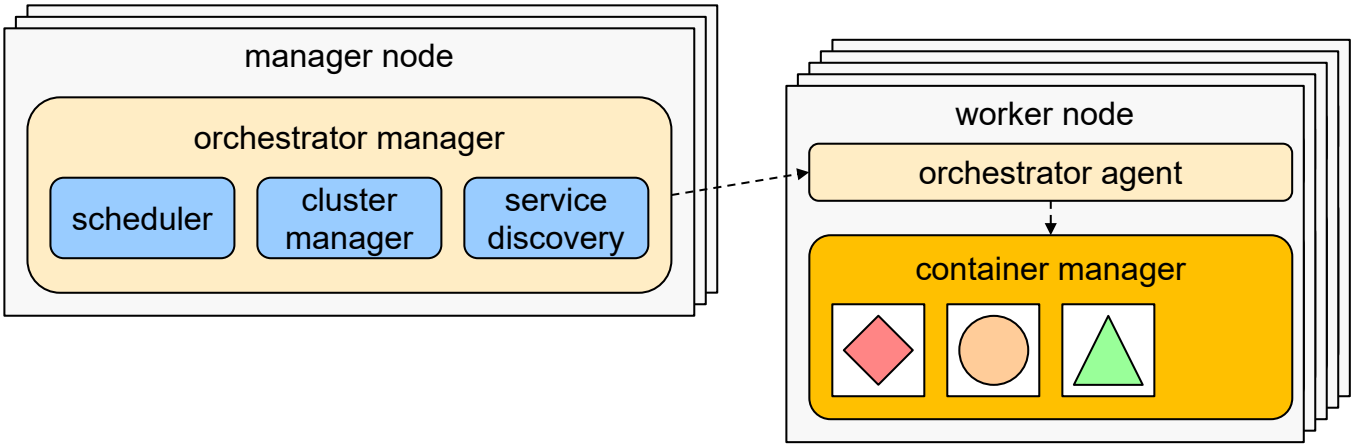
- Ecco alcune astrazioni di configurazione – Kubernetes
  - un *pod* è un'istanza di container da eseguire nel cluster – è l'unità di deployment nell'orchestratore
  - un *replica set* consente di gestire ed eseguire una o più repliche di un pod – un *daemon set* consente di avere una replica di un pod per ciascun nodo
  - un *deployment* è una risorsa di alto livello per gestire in modo dichiarativo il rilascio e l'aggiornamento di un'applicazione
  - un *service* è un punto d'ingresso unico e stabile per un insieme di pod che forniscono uno stesso servizio



# - Scalabilità e disponibilità dell'orchestratore

- È necessario sostenere disponibilità e scalabilità dell'orchestratore
  - a fronte del guasto di nodi e di variazioni del carico del cluster
  - nel cluster vengono utilizzati due tipi di nodi – *nodi manager* e *nodi worker*
    - i nodi manager vengono replicati per sostenere la disponibilità e la scalabilità dell'orchestratore

# Scalabilità e disponibilità dell'orchestratore



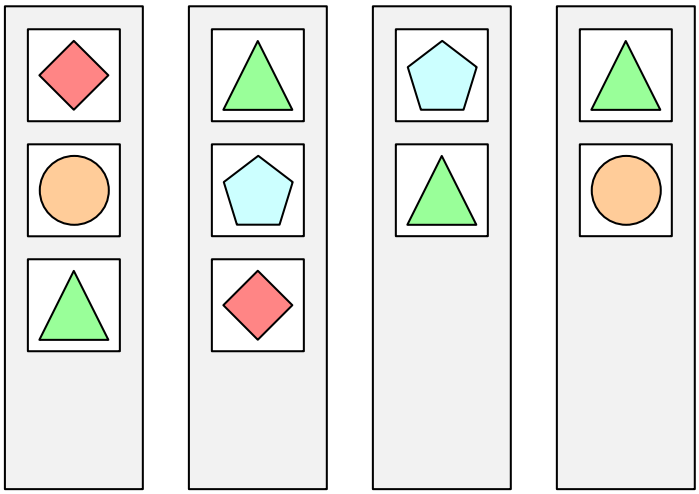



# - Disponibilità delle applicazioni

- È necessario sostenere la disponibilità delle applicazioni a container
  - a runtime, lo stato di un'applicazione deve corrispondere a quello richiesto dalla sua configurazione – anche a fronte del fallimento di alcuni container e del guasto di alcuni nodi
  - l'orchestratore effettua il monitoraggio dei container in esecuzione, e riavvia automaticamente i container che non funzionano più o che smettono di rispondere
  - se un nodo worker fallisce, allora l'orchestratore rischedula i suoi container nei nodi sopravvissuti del cluster

# Disponibilità delle applicazioni

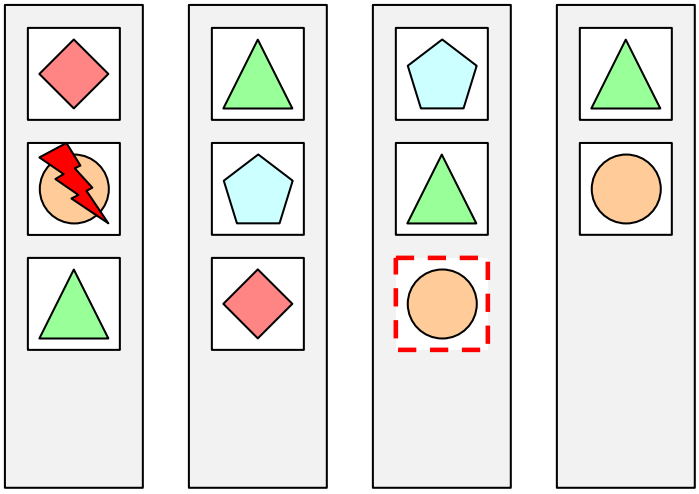
- Fallimento di un container (prima)





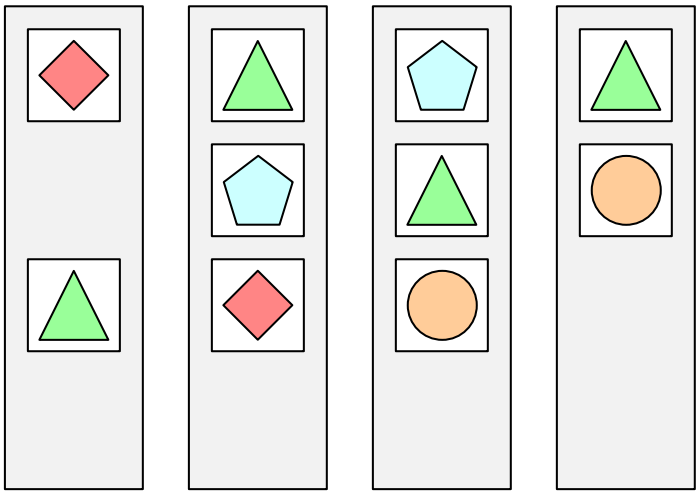
# Disponibilità delle applicazioni

- Fallimento di un container (dopo)



# Disponibilità delle applicazioni

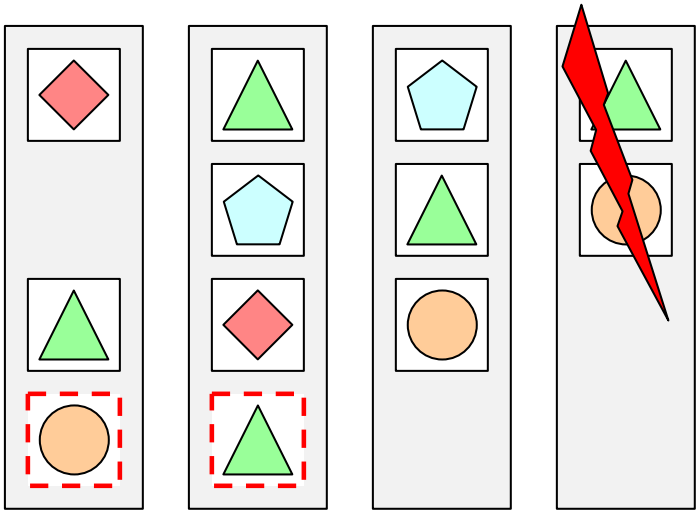
- Fallimento di un nodo (prima)





# Disponibilità delle applicazioni

- Fallimento di un nodo (dopo)



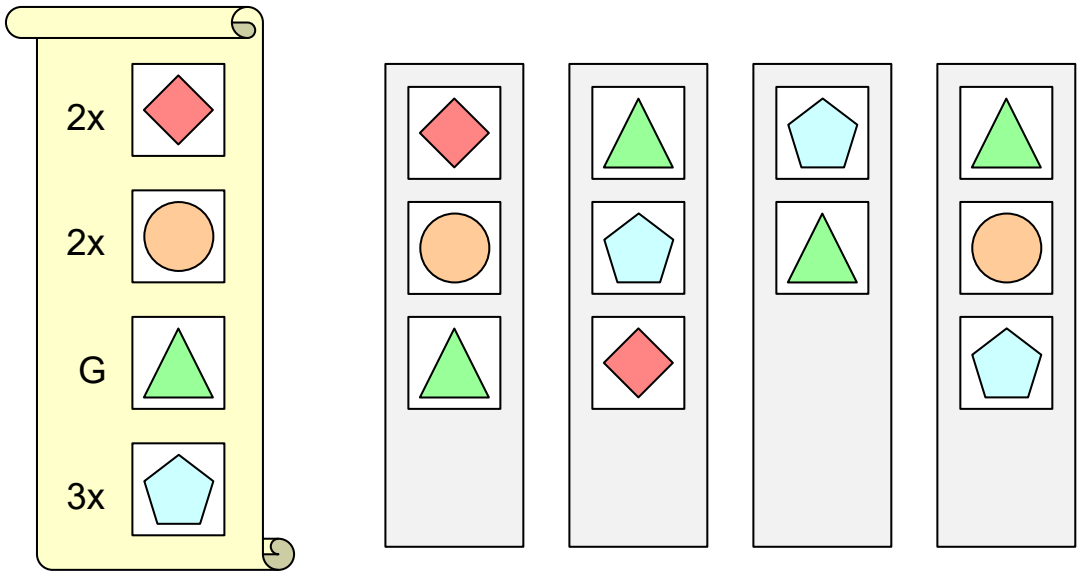
# - Scalabilità delle applicazioni

- È necessario sostenere la scalabilità, in modo elastico, delle applicazioni a container
  - ad es., se varia il carico di un'applicazione
  - l'amministratore può modificare (anche a runtime) la configurazione dell'applicazione per variare il numero di repliche di ciascun servizio
    - l'orchestratore avvia nel cluster i container aggiuntivi richiesti oppure arresta i container in eccesso
  - l'orchestratore può anche controllare automaticamente il numero di repliche per ciascuno dei servizi, monitorando le applicazioni
  - è anche possibile scalare la dimensione del cluster (nel cloud, anche automaticamente)



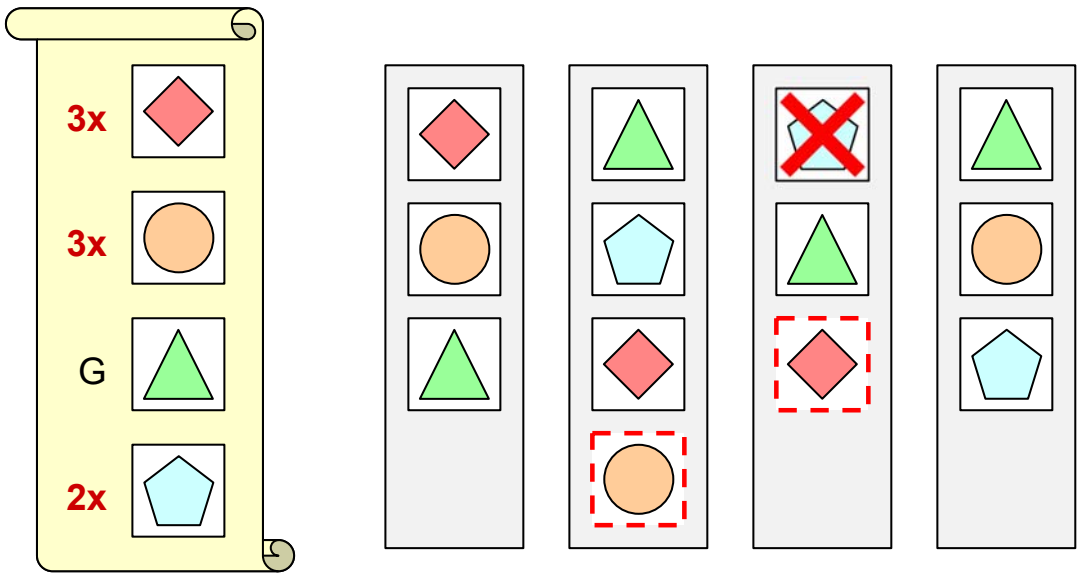
# Scalabilità delle applicazioni

- Scalabilità (prima)



# Scalabilità delle applicazioni

- Scalabilità (dopo)



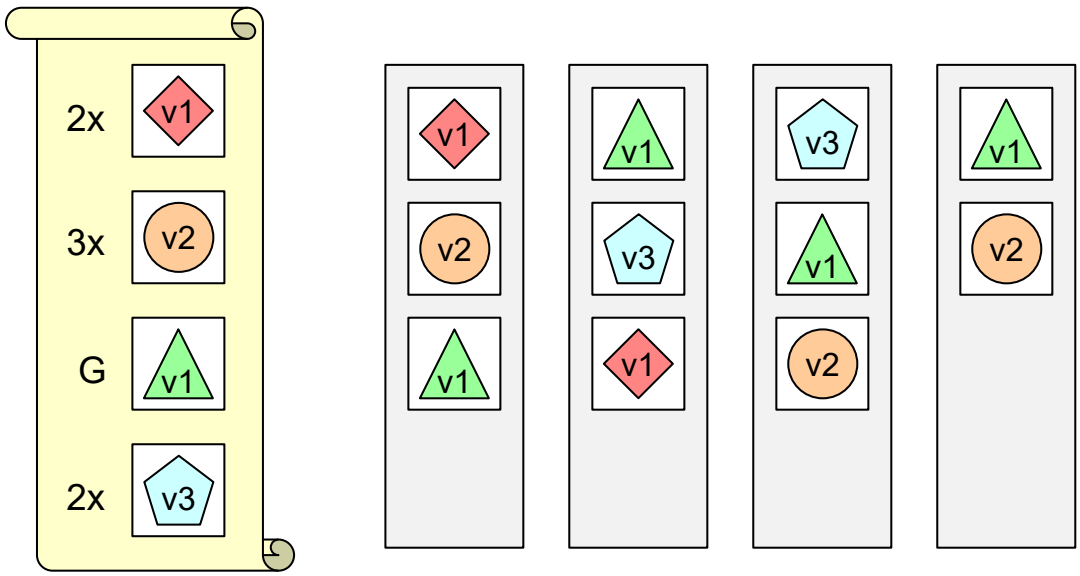



# - Rilascio di aggiornamenti

- È necessario un supporto per l'aggiornamento dei servizi e delle applicazioni a container in esecuzione
  - deve essere possibile il rilascio di una nuova versione di un servizio o di un'intera applicazione – in modo dichiarativo e, se possibile, anche senza interruzioni di servizio (*zero-downtime release*) per gli utenti finali
  - l'orchestratore implementa una o più tecniche di deployment automatico
    - ad es., un *rolling update* dei servizi in esecuzione
  - inoltre, fornisce la possibilità di effettuare il rollback di un servizio o un'applicazione a una versione precedente

# Rilascio di aggiornamenti

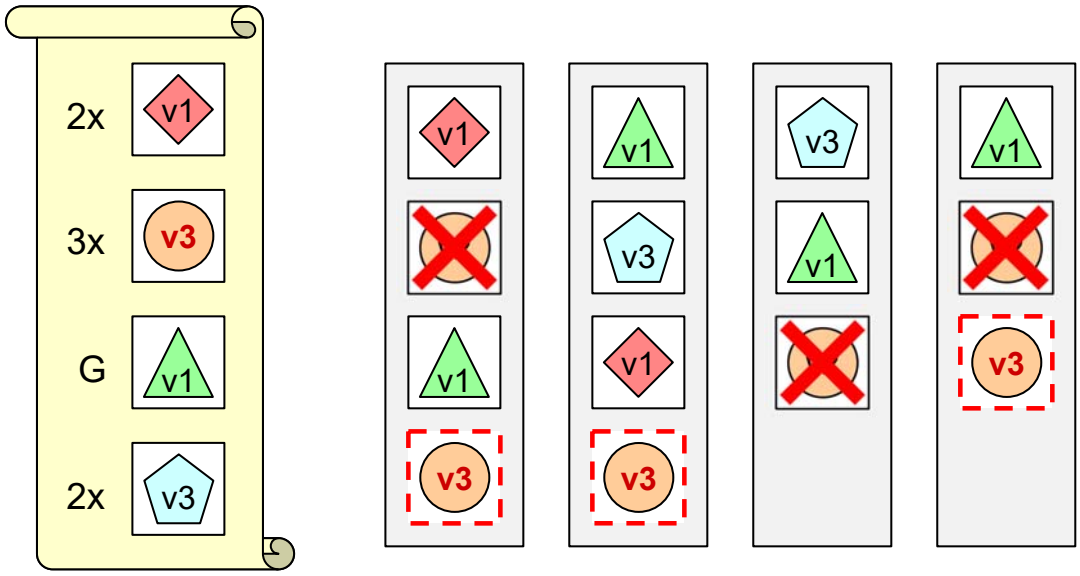
- Rolling update (prima)






# Rilascio di aggiornamenti

- Rolling update (dopo)



# - Comunicazione interna tra servizi

- Nelle applicazioni a container, i container per i servizi devono poter comunicare tra di loro in rete
  - i container per i servizi sono però effimeri (volatili) – così come la loro locazione in rete
  - i container sono inoltre distribuiti nei diversi nodi del cluster
  - inoltre, i container per ciascun servizio sono in genere replicati
  - per consentire la comunicazione tra i servizi, è necessario indirizzare ogni richiesta per un servizio a un container tra quelli che erogano il servizio richiesto



# Comunicazione interna tra servizi

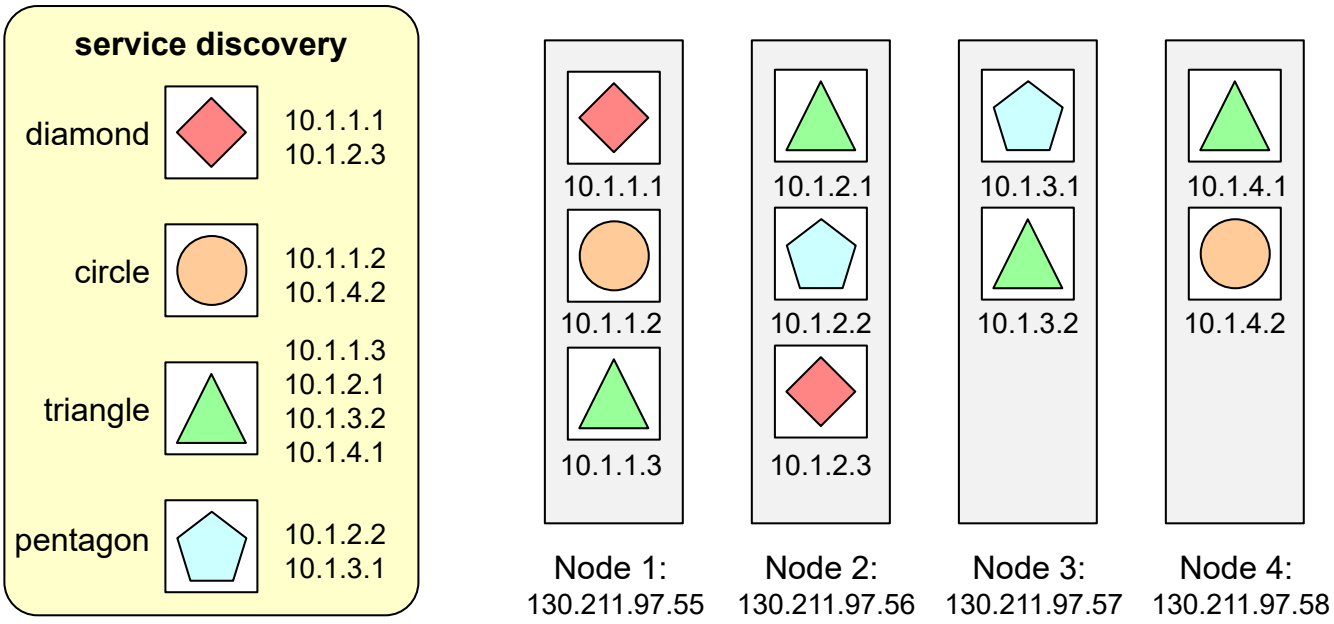
- Nelle applicazioni a container, i container per i servizi devono poter comunicare tra di loro in rete
  - l'orchestratore gestisce la comunicazione in rete tra servizi
    - gestisce un'overlay network sui nodi del cluster, e definisce uno spazio degli indirizzi di rete singolo per tutti i container – che fornisce un'infrastruttura per far comunicare i container in modo sicuro
    - inoltre, l'orchestratore definisce una rete privata (o namespace) per ciascuna applicazione
      - una tale rete consente la comunicazione tra i servizi dell'applicazione – inoltre l'orchestratore opera da DNS e da load balancer ("internal load balancing")
      - nel cluster è possibile eseguire più applicazioni a container, in ambienti isolati

# Comunicazione interna tra servizi

- Nelle applicazioni a container, i container per i servizi devono poter comunicare tra di loro in rete
  - l'orchestratore gestisce la comunicazione in rete tra servizi
    - l'orchestratore fornisce anche un servizio di service discovery
      - registra i servizi in esecuzione e ne effettua il monitoraggio
      - utilizza queste informazioni per indirizzare e distribuire le richieste per un servizio ai container attivi per il servizio
  - complessivamente, l'orchestratore fornisce una funzionalità di brokeraggio delle richieste (interne) tra servizi



# Comunicazione interna tra servizi



# Parentesi: Service discovery

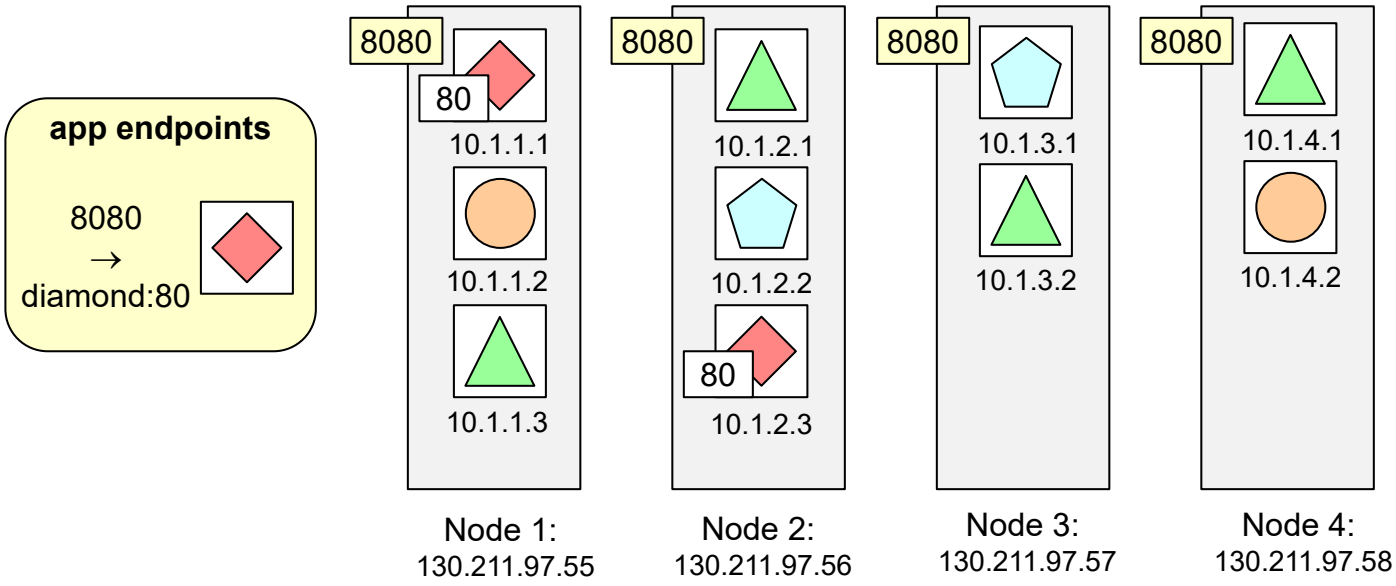
- Nella realizzazione di un'applicazione a servizi è in genere necessario un servizio di service discovery – utilizzando un orchestratore di container, sono possibili più soluzioni
  - soluzione applicativa – usando uno specifico servizio infrastrutturale (ad es., Consul) e le sue librerie
    - è indipendente dalla piattaforma
    - ma può essere compatibile solo con alcuni linguaggi di programmazione
  - soluzione fornita dalla piattaforma di orchestrazione
    - è indipendente dai linguaggi di programmazione
    - ma è legata alla piattaforma utilizzata



# - Comunicazione con i client esterni

- Anche i client esterni (utilizzati dagli utenti finali) devono poter comunicare in rete con le applicazioni e i servizi di loro interesse
  - l'orchestratore consente di esporre in rete degli endpoint alle applicazioni e ai servizi
    - ad es., come porte pubblicate, hostname o path HTTP
  - quando riceve la richiesta per un endpoint, la inoltra a uno dei container per il servizio richiesto
    - le richieste vengono di solito accettate su qualunque nodo del cluster ("ingress load balancing")
    - un componente esterno (ad es., un load balancer per il cloud) può accedere a un endpoint per un servizio da un qualunque nodo del cluster
  - l'orchestratore gestisce dunque anche il routing delle richieste (esterne) alle applicazioni e ai servizi

# Comunicazione con i client esterni






# Parentesi: API gateway

- Nella realizzazione di un'applicazione a servizi, la funzionalità di routing delle richieste dei client esterni ai servizi interni di un sistema viene di solito svolta da un API gateway
  - la funzionalità di routing fornita da un orchestratore può certamente semplificare la realizzazione dell'API gateway
  - è però in genere necessario utilizzare comunque un API gateway applicativo
    - ad es., per gestire l'autenticazione e per la composizione di API

# - Gestione di volumi

- È necessario un supporto per la memorizzazione persistente dei dati dei servizi, nonché per la condivisione di dati tra container
  - infatti, anche lo storage dei container è effimero
  - il container manager (o il container engine) può supportare la persistenza dei dati dei container in modo nativo – ad es., con i volumi
    - tuttavia, alcune soluzioni non sono applicabili in un cluster o nel cloud
  - l'orchestratore può fornire delle modalità aggiuntive di gestione dei dati persistenti
    - ad es., la possibilità di memorizzare dati persistenti condivisi in un servizio di storage per il cloud



# - Dati di configurazione e segreti

- È necessario un supporto per la gestione dinamica dei dati di configurazione delle applicazioni
  - quasi tutte le applicazioni richiedono dei dati di configurazione, che possono anche variare dinamicamente
  - alcuni dati di configurazioni sono sensibili, e devono essere gestiti in modo sicuro
  - queste configurazioni devono poter essere definite esternamente alle applicazioni
  - l'orchestratore può fornire delle astrazioni per la definizione dei dati di configurazioni – per poi mappare queste astrazioni su dei meccanismi concreti
    - inoltre, può notificare ai container variazioni nelle configurazioni

# - Discussione

- Ecco le principali caratteristiche e capacità fornite da un orchestratore di container per sostenere la gestione e l'esecuzione in produzione di applicazioni a container
  - architettura a servizi o a microservizi
    - specifica dichiarativa della composizione delle applicazioni
  - rilascio e aggiornamento delle applicazioni (senza interruzione di servizio)
    - scheduling dei container
  - scalabilità e disponibilità dell'orchestratore
  - scalabilità e disponibilità delle applicazioni
    - monitoraggio dei container e dei nodi
  - comunicazione tra servizi e routing delle richieste esterne
    - DNS, service discovery, ingress load balancing
  - gestione di volumi e dati di configurazione



# Discussione

- È possibile vedere delle analogie tra le responsabilità di un orchestratore di container e quelle di un contenitore per componenti ("Container") nell'architettura a componenti
  - ad es., entrambi forniscono la possibilità di specificare la composizione di applicazioni in modo dichiarativo, l'aggiornamento di applicazioni, sostengono la comunicazione sicura, sostengono scalabilità e disponibilità nell'esecuzione in un cluster
  - tuttavia, anche se i problemi affrontati sono analoghi, le soluzioni sono certamente diverse
    - le soluzioni realizzate da un contenitore per componenti sono mono-tecnologiche e proprietarie
    - l'orchestrazione di container supporta invece l'esecuzione di applicazioni e servizi realizzati anche con tecnologie diverse – ma omogenee nell'uso dei container – e inoltre sostiene la scalabilità delle applicazioni nel cloud

# * Discussione

- I container sono una possibile opzione per il rilascio di sistemi software distribuiti – in particolare, l'orchestrazione di container
  - sostiene il rilascio e la gestione di applicazioni distribuite multi-servizi e multi-container in produzione, in modo scalabile e affidabile
    - in un cluster di nodi, on premises oppure nel cloud
  - è supportata da strumenti specifici di orchestrazione
    - che offrono delle astrazioni di livello più alto rispetto a quelle fornite direttamente dai container manager
  - costituisce oggi l'infrastruttura o piattaforma preferita per l'esecuzione di applicazioni a microservizi
    - consente di sfruttare la disponibilità e l'elasticità delle piattaforme virtualizzate e nel cloud
  - costituisce un altro importante fattore di successo dei container e della piattaforma Docker