

Docker e Kubernetes: cosa sono, come funzionano e perché hanno cambiato il software

Oct 3, 2026 · @Giancarlo

Docker impacchetta un'applicazione in un contenitore che gira uguale ovunque; Kubernetes decide dove e come far girare migliaia di quei contenitori, li tiene in vita e li moltiplica quando serve. Insieme sono diventati lo standard di fatto per costruire e gestire il software moderno, dal sito di un'azienda locale ai servizi delle grandi piattaforme cloud.

Per capirli conviene partire da un problema antico dell'informatica: il programma che "sul mio computer funziona" e in produzione no. Librerie diverse, versioni del sistema operativo, configurazioni dimenticate. I container risolvono questo problema; l'orchestrazione risolve quello successivo, cioè gestirne tanti insieme senza impazzire.

Docker: l'applicazione in una scatola

Docker, lanciato nel 2013, ha reso semplice e popolare il concetto di container. Un container è un processo isolato che porta con sé tutto ciò che gli serve (codice, librerie, dipendenze, configurazione) ma condivide il kernel del sistema operativo ospite. Per questo è molto più leggero di una macchina virtuale: si avvia in secondi e ne girano decine sullo stesso server.

Le caratteristiche principali:

- **Immagini e Dockerfile.** Un file di testo, il Dockerfile, descrive passo per passo come costruire l'immagine dell'applicazione. L'immagine è un modello immutabile e versionato, composto da strati riutilizzabili: se cambia solo il codice, non si ricostruisce tutto.
- **Portabilità.** La stessa immagine gira identica sul portatile dello sviluppatore, sul server aziendale, su un NAS o nel cloud. Le immagini seguono gli standard aperti OCI, quindi non sono legate a Docker.
- **Isolamento.** Ogni container ha il proprio file system, la propria rete e limiti di CPU e memoria, grazie a funzioni del kernel Linux (namespaces e cgroups).
- **Registri.** Le immagini si pubblicano e si scaricano da registri come Docker Hub o da registri privati: installare un database o un CMS diventa un comando.
- **Docker Compose.** Un unico file descrive un'applicazione fatta di più container (per esempio sito, database e cache) e la avvia con un comando.

- **Docker Desktop.** L'ambiente grafico per Windows e macOS, gratuito per uso personale e piccole aziende ma a pagamento per le organizzazioni più grandi.

Il limite di Docker da solo è la scala: gestisce bene i container su una macchina, ma non decide come distribuirli su dieci o cento server, né li riavvia altrove se un server si guasta. Qui entra Kubernetes.

Kubernetes: il direttore d'orchestra

Kubernetes (spesso abbreviato in K8s) è una piattaforma open source che gestisce applicazioni containerizzate su un gruppo di server, chiamato cluster. Nasce in Google, che lo ha reso pubblico nel 2014 sulla base dell'esperienza del suo sistema interno Borg, ed è oggi governato dalla Cloud Native Computing Foundation. Escono circa tre versioni all'anno, ciascuna supportata con aggiornamenti per circa quattordici mesi ([Kubernetes, patch releases](#)).

Il principio chiave è dichiarativo: non si dice a Kubernetes *cosa fare*, gli si descrive *lo stato desiderato* ("voglio tre copie di questo servizio, con questa memoria, raggiungibili a questo indirizzo") e il sistema lavora di continuo per mantenerlo.

Le caratteristiche principali:

- **Pod e Deployment.** Il pod è l'unità minima: uno o più container che lavorano insieme. Il Deployment dice quante copie del pod devono esistere e come aggiornarle.
- **Autoriparazione.** Se un container si blocca viene riavviato; se un server si spegne, i suoi pod vengono ricreati sugli altri nodi senza intervento umano.
- **Scalabilità automatica.** Il numero di copie cresce o cala in base al carico (CPU, memoria o metriche personalizzate). Nelle versioni recenti si possono anche cambiare le risorse di un pod in esecuzione senza riavviarlo: la funzione è diventata stabile con la 1.35 di dicembre 2025 ([Kubernetes v1.35](#)).
- **Aggiornamenti senza interruzioni.** Le nuove versioni si distribuiscono gradualmente (rolling update) e, se qualcosa va storto, si torna indietro con un comando.
- **Service e Ingress.** Danno un indirizzo stabile ai servizi e distribuiscono il traffico tra le copie, anche quando i pod vengono spostati.
- **Configurazioni e segreti.** ConfigMap e Secret separano parametri e password dal codice.
- **Storage persistente.** I volumi collegano i container a dischi locali, NAS o storage cloud, così i dati sopravvivono ai container.
- **Estendibilità.** Con le risorse personalizzate e gli operatori si insegna a Kubernetes a gestire database, code di messaggi o modelli di intelligenza artificiale; Helm installa intere applicazioni come pacchetti.

- **Sicurezza e multi-utenza.** Namespace, controllo degli accessi basato sui ruoli (RBAC) e policy di rete separano team e applicazioni sullo stesso cluster.

Una precisazione utile: dal 2022 Kubernetes non usa più Docker come motore interno, ma runtime come containerd o CRI-O. Le immagini costruite con Docker però funzionano esattamente come prima, perché seguono lo stesso standard OCI.

Docker e Kubernetes: complementari, non rivali

Il confronto "Docker contro Kubernetes" è fuorviante: lavorano a due livelli diversi. Docker costruisce e fa girare il container; Kubernetes decide quanti, dove e come, su molte macchine. Il flusso tipico è: lo sviluppatore crea l'immagine con Docker, la pubblica in un registro, e Kubernetes la scarica e la esegue nel cluster.

Aspetto	Docker	Kubernetes
Compito	Creare ed eseguire container	Orchestrare container su più server
Scala tipica	Un singolo host	Da pochi nodi a migliaia
Unità di lavoro	Container	Pod (uno o più container)
Guasto di un server	I container si fermano	I pod ripartono su altri nodi
Scalabilità	Manuale	Automatica in base al carico
Curva di apprendimento	Bassa	Alta
Ideale per	Sviluppo, test, piccoli servizi	Produzione, microservizi, alta disponibilità

In mezzo c'è Docker Swarm, l'orchestratore integrato in Docker: più semplice di Kubernetes, ma con un ecosistema molto più piccolo. Per chi ha pochi servizi su una o due macchine, Docker con Compose resta spesso la scelta più sensata.

Potenzialità e casi d'uso

La forza della coppia sta nel rendere il software ripetibile, spostabile e automatico. Ecco dove questo fa la differenza.

- **Microservizi.** Un'applicazione grande si divide in servizi piccoli e indipendenti (catalogo, pagamenti, utenti), ognuno nel suo container, aggiornabile senza fermare il resto.

- **DevOps e rilascio continuo.** Ogni modifica al codice produce una nuova immagine che passa test automatici e arriva in produzione in modo controllato, anche più volte al giorno.
- **Cloud ibrido e multi-cloud.** La stessa configurazione gira nel data center aziendale e sui servizi gestiti dei grandi provider (Amazon EKS, Google GKE, Microsoft AKS), riducendo la dipendenza da un singolo fornitore.
- **Edge computing e telecomunicazioni.** Versioni leggere come K3s portano Kubernetes su piccoli server in negozi, fabbriche e antenne di rete; gli operatori telefonici lo usano per le reti 5G e Open RAN.
- **Intelligenza artificiale e dati.** Kubernetes è diventato la piattaforma di riferimento per addestrare e servire modelli di IA, distribuendo il lavoro sulle GPU disponibili.
- **Piccole realtà e laboratori.** Anche senza Kubernetes, Docker semplifica la vita: su un NAS o un piccolo server si installano in pochi minuti strumenti come CMS, database, sistemi di automazione o di backup, ognuno isolato e facile da aggiornare.

Il vantaggio economico è la densità: più applicazioni sugli stessi server, risorse assegnate in base all'uso reale, meno tempo speso a configurare macchine a mano.

Esempio pratico: un CRM su Docker e su Kubernetes

Per vedere la differenza sul campo prendiamo [EspoCRM](#), un CRM open source per gestire contatti, aziende, opportunità e attività. Ha un'immagine Docker ufficiale e ha bisogno di un database MySQL: due componenti, quindi un caso perfetto. Le configurazioni seguono la [documentazione ufficiale di EspoCRM per Docker](#); le password qui sotto sono segnaposto da cambiare.

Installazione con Docker Compose

Serve un computer o un server con Docker installato (Docker Desktop su Windows e macOS, Docker Engine su Linux; anche molti NAS lo offrono).

1. Crea una cartella per il progetto ed entraci:

```
mkdir espocrm-docker
cd espocrm-docker
```

2. Crea al suo interno il file `docker-compose.yml`. Descrive tre container: il database, il CRM e un processo di servizio (daemon) che esegue le attività pianificate come l'invio delle email.

```

services:
  mysql:
    image: mysql:8
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: cambia-root
      MYSQL_DATABASE: espocrm
      MYSQL_USER: espocrm
      MYSQL_PASSWORD: cambia-db
    volumes:
      - mysql:/var/lib/mysql
    restart: always

  espocrm:
    image: espocrm/espocrm
    container_name: espocrm
    environment:
      ESPOCRM_DATABASE_PLATFORM: Mysql
      ESPOCRM_DATABASE_HOST: mysql
      ESPOCRM_DATABASE_USER: espocrm
      ESPOCRM_DATABASE_PASSWORD: cambia-db
      ESPOCRM_ADMIN_USERNAME: admin
      ESPOCRM_ADMIN_PASSWORD: cambia-admin
      ESPOCRM_SITE_URL: "http://localhost:8080"
      ESPOCRM_LANGUAGE: it_IT
    volumes:
      - espocrm:/var/www/html
    restart: always
    ports:
      - 8080:80

  espocrm-daemon:
    image: espocrm/espocrm
    container_name: espocrm-daemon
    volumes:
      - espocrm:/var/www/html
    restart: always
    entrypoint: docker-daemon.sh

volumes:
  mysql:
  espocrm:

```

3. Avvia tutto in background:

```
docker compose up -d
```

4. Controlla che i container siano attivi con `docker compose ps` e segui l'installazione con `docker compose logs -f espocrm`. Al primo avvio il CRM crea il database: servono uno o due minuti.
5. Apri il browser su `http://localhost:8080` (o sull'indirizzo IP del server, aggiornando `ESPOCRM_SITE_URL` di conseguenza) ed entra con l'utente `admin` e la password scelta.
6. Per aggiornare in futuro: `docker compose pull && docker compose up -d`. I dati restano nei volumi `mysql` ed `espocrm`.

Con un file di una cinquantina di righe e un comando il CRM è operativo, con dati persistenti e riavvio automatico. Il limite è quello già visto: tutto gira su una sola macchina.

Installazione su Kubernetes

Su Kubernetes la stessa applicazione si descrive con più oggetti, ognuno con un compito preciso. Per provare basta un cluster locale: quello integrato in Docker Desktop, minikube o K3s (leggero, adatto anche a un piccolo server). Serve lo strumento a riga di comando `kubectl`.

Oggetto Kubernetes	A cosa serve nel nostro CRM
Namespace <code>crm</code>	Raccoglie tutte le risorse del CRM, separate dal resto del cluster
Secret	Custodisce le password fuori dai file di configurazione
PersistentVolumeClaim	Chiede lo spazio disco per database e file del CRM
Deployment	Tiene in vita i container e li ricrea se si fermano
Service	Dà un nome di rete stabile (<code>mysql</code> , <code>espocrm</code>) ai componenti

1. Crea il file `database.yaml` con namespace, password, disco e MySQL:

```
apiVersion: v1
kind: Namespace
metadata:
  name: crm
---
apiVersion: v1
kind: Secret
metadata:
```

```

    name: crm-secret
    namespace: crm
stringData:
  mysql-root-password: cambia-root
  db-password: cambia-db
  admin-password: cambia-admin
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-data
  namespace: crm
spec:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 5Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: crm
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:8
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef: {name: crm-secret, key: mysql-root-password}
            - name: MYSQL_DATABASE
              value: espocrm
            - name: MYSQL_USER
              value: espocrm

```

```

        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef: {name: crm-secret, key: db-password}
      ports:
        - containerPort: 3306
      volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
    volumes:
      - name: data
        persistentVolumeClaim:
          claimName: mysql-data
---
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: crm
spec:
  selector:
    app: mysql
  ports:
    - port: 3306

```

2. Applicalo e aspetta che il database sia pronto:

```

kubectl apply -f database.yaml
kubectl wait -n crm --for=condition=available deployment/mysql --timeout=180s

```

3. Crea il file `espocrm.yaml`. Il CRM e il suo daemon stanno nello stesso pod, così condividono lo stesso disco come nella versione Compose:

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: espocrm-data
  namespace: crm
spec:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 5Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:

```

```

name: espocrm
namespace: crm
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: espocrm
  template:
    metadata:
      labels:
        app: espocrm
    spec:
      containers:
        - name: espocrm
          image: espocrm/espocrm
          env:
            - name: ESPOCRM_DATABASE_PLATFORM
              value: Mysql
            - name: ESPOCRM_DATABASE_HOST
              value: mysql
            - name: ESPOCRM_DATABASE_USER
              value: espocrm
            - name: ESPOCRM_DATABASE_PASSWORD
              valueFrom:
                secretKeyRef: {name: crm-secret, key: db-password}
            - name: ESPOCRM_ADMIN_USERNAME
              value: admin
            - name: ESPOCRM_ADMIN_PASSWORD
              valueFrom:
                secretKeyRef: {name: crm-secret, key: admin-password}
            - name: ESPOCRM_SITE_URL
              value: "http://localhost:8080"
            - name: ESPOCRM_LANGUAGE
              value: it_IT
          ports:
            - containerPort: 80
          volumeMounts:
            - name: html
              mountPath: /var/www/html
        - name: espocrm-daemon
          image: espocrm/espocrm
          command: ["docker-daemon.sh"]
          volumeMounts:
            - name: html

```

```

        mountPath: /var/www/html
    volumes:
      - name: html
        persistentVolumeClaim:
          claimName: espocrm-data
---
apiVersion: v1
kind: Service
metadata:
  name: espocrm
  namespace: crm
spec:
  selector:
    app: espocrm
  ports:
    - port: 80
      targetPort: 80

```

4. Applicalo e controlla lo stato dei pod:

```

kubectl apply -f espocrm.yaml
kubectl get pods -n crm
kubectl logs -n crm deployment/espocrm -c espocrm -f

```

5. Per una prova in locale, collega la porta del servizio al tuo computer e apri `http://localhost:8080`:

```

kubectl port-forward -n crm svc/espocrm 8080:80

```

6. Prova l'autoriparazione: cancella il pod del CRM con `kubectl delete pod -n crm -l app=espocrm`. Kubernetes ne crea subito uno nuovo, collegato allo stesso disco, e il CRM torna disponibile con tutti i dati.

Cosa cambia passando a Kubernetes

La versione Kubernetes richiede più righe, ma il risultato è diverso nella sostanza: le password stanno in un Secret, i componenti si trovano tramite nomi di servizio, i guasti si riparano da soli e lo stesso pacchetto di file si può portare su qualunque cluster, in sede o nel cloud.

Per andare in produzione servono ancora alcuni passi:

- **Accesso dall'esterno.** Al posto del port-forward si usa un Ingress con un nome di dominio e un certificato HTTPS (per esempio con cert-manager), aggiornando `ESPOCRM_SITE_URL`.

- **Database robusto.** MySQL andrebbe gestito con uno StatefulSet o un operatore dedicato, oppure affidato a un database gestito del provider cloud.
- **Backup.** Vanno salvati con regolarità sia il database sia il volume dei file.
- **Più copie del CRM.** Qui `replicas` è 1 perché il disco `ReadWriteOnce` si collega a un solo nodo alla volta. Per far girare più copie in parallelo serve uno storage condiviso `ReadWriteMany` (per esempio NFS, anche su un NAS) e una configurazione adatta delle sessioni.
- **Pacchetti Helm.** Per installazioni ripetute conviene raccogliere questi file in un chart Helm, così un'intera installazione si crea o si aggiorna con un comando.

Il confronto riassume bene la regola vista prima: per un'azienda con un solo server, Docker Compose basta e avanza; quando il CRM diventa critico, deve restare sempre acceso e crescere con gli utenti, Kubernetes ripaga la complessità in più.

Un ecosistema intero costruito su Kubernetes

Kubernetes non è solo un software: attorno a lui è nato un mercato. Lo racconta bene un reportage di Antony Savvas su *Information Age* ([5 Kubernetes technology vendors hot right now](#), febbraio 2023), scritto dopo un giro tra le aziende della Silicon Valley. Il giornalista descrive Kubernetes come un pilastro del movimento DevOps, attorno al quale è cresciuto un ecosistema completamente nuovo, e presenta cinque fornitori che mostrano quanto lontano arrivi la sua influenza.

Azienda	Cosa fa con Kubernetes
Rakuten Symphony	Piattaforma Kubernetes per l'edge e le reti delle telco (Open RAN), nata dall'acquisizione di Robin.io; ambisce a diventare la "VMware dell'edge"
CloudFabrix	Piattaforma AIOps che raccoglie l'inventario di pod, nodi e servizi tramite kubectl per automatizzare la gestione IT
Pulumi	Infrastructure as code open source: l'infrastruttura, cluster compresi, si descrive con normali linguaggi di programmazione
Solo.io	Service mesh basato su Istio (Gloo Platform, Ambient Mesh) per rete, API e sicurezza tra servizi

Azienda	Cosa fa con Kubernetes
Sysdig	Monitoraggio, sicurezza e controllo dei costi per container e cluster, anche con lo strumento open source Falco

Il punto più attuale dell'articolo riguarda i costi. Savvas cita una ricerca di Civo su 1.000 sviluppatori: il 57% aveva visto aumentare il numero di cluster nella propria organizzazione e il 47% segnalava costi di Kubernetes in crescita, per la maggior parte fino al 25% in un anno. Secondo l'amministratore delegato di Sysdig, molti cluster sono sovradimensionati per mancanza di competenze, e spostare le applicazioni nel cloud senza progettazione porta a costi alti e prestazioni deludenti.

L'articolo ha qualche anno e alcuni nomi nel frattempo sono cambiati (VMware, citata come concorrente di riferimento, è stata acquisita da Broadcom), ma la tesi di fondo regge: l'ecosistema si è allargato e il controllo dei costi è diventato una disciplina a sé, chiamata FinOps.

Limiti, costi e complessità

Kubernetes risolve problemi da grande scala, e porta con sé una complessità da grande scala. Prima di adottarlo conviene pesare alcuni aspetti.

- **Curva di apprendimento ripida.** Concetti, file YAML, rete e storage richiedono competenze specifiche, spesso un team dedicato.
- **Costi nascosti.** Cluster sovradimensionati, servizi gestiti a consumo e strumenti di monitoraggio possono far crescere la spesa, come segnala la ricerca citata da Information Age.
- **Sicurezza.** Immagini con vulnerabilità, permessi troppo larghi e segreti mal gestiti sono gli errori più comuni; servono scansione delle immagini e policy chiare.
- **Applicazioni con stato.** Database e sistemi che conservano dati funzionano su Kubernetes, ma richiedono più attenzione delle applicazioni senza stato.
- **Eccesso di soluzione.** Per un sito o pochi servizi, Kubernetes è spesso sproporzionato: Docker con Compose, o un servizio gestito, costano meno fatica.

La regola pratica: Docker conviene quasi a tutti; Kubernetes conviene quando i servizi sono molti, devono restare sempre accesi e scalare con il traffico.

Conclusioni

Docker ha reso il software portabile; Kubernetes lo ha reso autonomo e scalabile. Il primo è ormai uno strumento di base per chiunque sviluppi o gestisca server, anche in piccolo. Il secondo è la fondazione su cui girano il cloud moderno, le reti delle telco e una parte crescente dell'intelligenza artificiale, con un ecosistema di aziende che continua ad allargarsi. La sfida dei prossimi anni non è più adottarli, ma usarli con competenza e tenere sotto controllo i costi.

Fonti

- Antony Savvas, [5 Kubernetes technology vendors hot right now](#), Information Age, 28 febbraio 2023
- [Kubernetes v1.35: Timbernetes](#), blog ufficiale Kubernetes, dicembre 2025
- [Patch Releases](#), documentazione ufficiale Kubernetes
- [EspoCRM with Docker](#), documentazione ufficiale EspoCRM