Capitolo 1

# Primi passi

> "Il perfezionismo ci impedisce di fare doppi passi nella nostra carriera. Pensiamo di dover essere perfetti, ma non è così."
>
> – Reshma Saujani, avvocato e fondatrice di Girls Who Code

Con l'avvento dei container Docker, questa tecnologia "dimenticata" appartenente al mondo Linux torna in auge e si fa spazio in un mondo dove la portabilità è la chiave del successo di ogni applicazione: eliminare dal proprio frasario "sul mio PC funziona", per poter affermare con certezza che l'applicazione funzionerà su qualsiasi ambiente, indipendentemente dall'infrastruttura o dal sistema operativo, è stata una vera e propria rivoluzione nel settore IT.
Con il passare degli anni, è chiaro che le esigenze sono cambiate e che i limiti appartenenti al prodotto della Docker Inc. sono emersi piuttosto velocemente: in un ambiente cosiddetto di "produzione", ossia l'ambiente utilizzato direttamente dagli utenti finali, è necessario prevedere dei meccanismi di alta disponibilità, piuttosto che di recupero in caso di problematiche all'hardware o al software installato.
Senza però fare ulteriori "spoiler" sui contenuti di questo e dei prossimi capitoli, cominciamo dalle basi: questa parte del manuale sarà infatti dedicata alla comprensione del perché Docker (o altri strumenti di containerizzazione) non è sufficiente per i nostri scopi e come mai tecnologie come Kubernetes si sono fatte largo nel settore grazie alle numerose features introdotte fin dai primi rilasci.

## In questo capitolo

**SUGGERIMENTO**

Questo capitolo affronterà solo una piccola parte dei concetti di base relativi al mondo dei container: la maggior parte della terminologia specifica è stata ampiamente discussa nel volume intitolato *Docker: Sviluppare e rilasciare software tramite container*, edito sempre da Apogeo. Se si parte da zero, si consiglia quindi di dare prima un'occhiata a questo manuale, per arrivare più preparati/e ai successivi capitoli!

## Dal container all'orchestratore

I *container* nascono come tecnologia che consentiva di impacchettare e isolare le applicazioni con il loro intero ambiente di *runtime*, compresi tutti i file necessari per l'esecuzione. Ciò semplificava notevolmente lo spostamento dell'applicazione tra differenti ambienti durante le varie fasi di realizzazione del prodotto (per esempio sviluppo, test, produzione e così via) pur mantenendo la piena funzionalità. Inoltre, i container nascono come parte fondamentale della sicurezza: adottando le opportune *best practice*, quali l'utilizzo di tali oggetti come ambiente di esecuzione, si ottiene una forma di segregazione per definizione e ciò garantisce che la tua applicazione sia affidabile, scalabile e sicura.
Tuttavia i container non nascono con Docker, ma fanno parte del mondo Linux®, in quanto rappresentano un insieme di uno o più processi isolati dal resto del sistema. Tutti i file necessari per metterli in esecuzione vengono forniti da un'immagine distinta, il che significa che i *container Linux* sono portatili e coerenti mentre passano dallo sviluppo, al test e infine alla produzione. Ciò li rende molto più veloci da utilizzare rispetto a un flusso di lavoro che si basa sulla replica di ambienti di test tradizionali.
Immagina di sviluppare un'applicazione: lavori su un laptop e il tuo ambiente ha una configurazione specifica, con file `.properties`, librerie e dipendenze, ma anche programmi che hai dovuto installare per far funzionare tutto l'ambiente. Altre persone che sviluppano con te potrebbero avere configurazioni leggermente diverse: una differente versione della macchina virtuale utilizzata per interpretare e compilare il linguaggio di programmazione adottato oppure errori o sviste nei file di configurazione e così via. L'applicazione che stai sviluppando si basa però su tale configurazione e dipende da librerie, dipendenze e file specifici. Allo stesso tempo, sai che all'interno della tua azienda ci sono degli ambienti di sviluppo e produzione standardizzati, ognuno con le proprie configurazioni e relativi file, per cui il tuo ambiente dovrà emulare quegli ambienti il più possibile in locale, anche se ci saranno delle impostazioni da modificare opportunamente. Inoltre, è chiaro che sul tuo PC sarà sufficiente eseguire l'applicazione con una sola istanza e testarla, magari tramite un browser, ma sugli ambienti utilizzati dagli utenti finali bisognerà accertarsi che non ci sia alcun disservizio, anche in caso di errore applicativo.
Come fai a far funzionare la tua applicazione in questi ambienti, a garantirne un certo livello di qualità tramite un flusso di CI/CD eseguito attraverso strumenti differenti, ma soprattutto a distribuire la tua app senza sbattere la testa su configurazioni e problemi vari da risolvere? La risposta: tramite i *container*.
Il container che ospiterà la tua applicazione conterrà le librerie, le dipendenze e i file necessari al funzionamento della stessa, così da poterla spostare in produzione senza spiacevoli effetti collaterali. In effetti, il contenuto di un'immagine, creata utilizzando uno strumento open source come Buildah o Podman, può essere considerato come l'installazione *ex novo* di una qualsiasi distribuzione Linux con l'aggiunta dell'applicazione

sviluppata. In quanto corredata di pacchetti RPM, file di configurazione, ecc., rende tutta l'operazione molto più semplice rispetto all'installazione di nuove copie dei sistemi operativi. Crisi scongiurata: tutti sono felici.

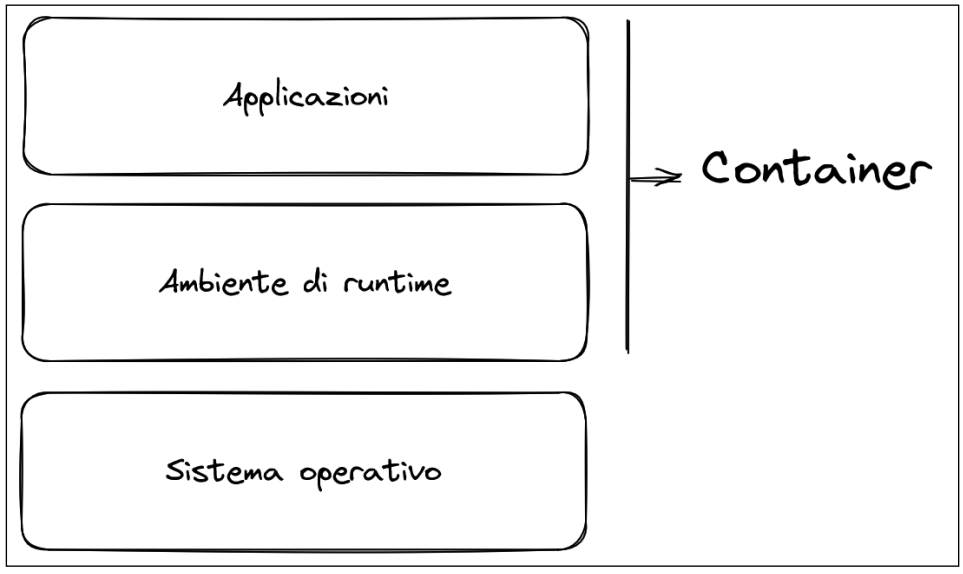


**Figura 1.1** Le responsabilità di un container nel contesto di rilascio di un'applicazione.

Questo è un esempio abbastanza comune, ma il concetto di container può essere applicato a molti problemi diversi in cui sono necessarie portabilità, configurabilità e isolamento. Lo scopo dei *container Linux* (quindi già prima di Docker) è sempre stato quello di favorire un più rapido sviluppo da parte di chi si occupa di implementare le funzionalità e di soddisfare le esigenze aziendali man mano che si presentano, senza però lasciare che questo compito "consumi" del tempo necessario ad altre attività. Non solo: il termine "sicurezza" è venuto fuori diverse volte. Perché è così importante e perché dovremmo preoccuparcene? La sicurezza dei container comporta la definizione e l'adesione a procedure di creazione, distribuzione e runtime che proteggono un container, e riguarda anche le applicazioni che supportano all'infrastruttura su cui si basano. La sicurezza dei container deve essere un processo integrato e continuo a supporto della sicurezza complessiva di un'azienda. In generale non si tratta soltanto di garantire la protezione della sola applicazione, ma anche di tutto il flusso di lavoro e degli ambienti che costituiscono l'infrastruttura che serve a distribuire le applicazioni.

I container hanno quindi rappresentato uno strumento rivoluzionario per il mondo dello sviluppo e del rilascio del software, ma hanno richiesto molto lavoro e un'evoluzione significativa verso una gestione centralizzata e semplificata del loro ciclo di vita: è qui che strumenti come Kubernetes entrano in gioco e ci permettono di avere una cassetta degli attrezzi pronta all'uso estremamente potente per poter lavorare con continuità e in maniera efficiente nella fase successiva allo sviluppo dell'applicazione. Prima però di parlare dell'orchestrazione, è importante soffermarsi sull'aspetto della sicurezza di questi oggetti: non solo in Kubernetes, ma anche in tecnologie che sfruttano Kubernetes

dietro le quinte; questo fattore è cruciale e rappresenta, per i clienti che adottano queste soluzioni, un requisito fondamentale per il rilascio dei propri prodotti. Vediamo quindi come integrare la sicurezza nel contesto dei container, ancor prima di "atterrare" nel mondo dell'orchestrazione, e quindi di Kubernetes.

## Integrare la sicurezza

Non è un mistero che negli ultimi anni il termine *hacker* sia entrato nella quotidianità: solo nel 2022 sono stati riportati 236 milioni di attacchi causati da *ransomware* con conseguenze più o meno gravi. Questo ha fatto sì che per la maggior parte delle aziende la sicurezza informatica diventasse una priorità da non relegare in fondo alla catena di rilascio del software, ma al contrario da porre in cima alla lista. In tale contesto, come si integra il mondo dei container? I container sono composti da diversi strati di immagini e sono ormai uno standard diffuso per il rilascio delle applicazioni, sia che si parli di un ambiente "tradizionale" (on-premise in gergo) che di un ambiente cloud. Docker ha segnato un punto su una tecnologia che ha di fatto rivoluzionato il deploy, e ha rappresentato l'inizio della diffusione di altri motori come Buildah o Podman per la definizione e messa in esecuzione delle immagini. Le immagini hanno quindi creato uno standard per tutte le persone che sviluppano, lasciando un vuoto non trascurabile: come gestire flussi di lavoro che esulano da questi standard? Chiariamo questo aspetto: non sempre andremo a lavorare su un ambiente "chiuso" e (quasi) sempre il lavoro che facciamo non è pura e semplice sperimentazione, ma un qualcosa che verrà messo a disposizione degli utenti finali. Questo vuol dire *esporre* l'applicazione verso l'esterno e di conseguenza a potenziali rischi. Una riflessione importante riguarda l'introduzione della sicurezza a monte di questo flusso, quindi ancor prima che l'applicazione sia a disposizione degli utenti finali: questo ci permette di non dover *rimediare*, ma di *prevenire* rispetto al perimetro di vulnerabilità che ci troviamo a dover difendere.

Non a caso, si è passati da un approccio DevOps, dove sviluppo e rilascio (semplificando) sono ben integrati e connessi insieme, al movimento DevSecOps: il lavoro congiunto tra il team che sviluppa l'applicazione, quello che la rilascia e quello che si occupa della sicurezza è necessario e fortemente voluto dalle aziende che vedono nella gestione della sicurezza non solo una mitigazione di problemi futuri, ma un investimento redditizio. Ormai le immagini distribuite tramite container sono divenute il formato standard di rilascio delle applicazioni negli ambienti nativi del cloud, ma anche in quelli basati su un approccio ibrido. Come indicano molte aziende leader nel settore, tra cui RedHat, bisogna prestare particolare attenzione alle immagini di base utilizzate: ai fini della sicurezza, questa è la fase più delicata perché questa immagine (in gergo tecnico è chiamata anche *base image* o *golden image*) viene utilizzata come punto di partenza da cui si creano tutte le immagini che ne derivano. Questo vuol dire che ogni pacchetto installato e ogni strumento utilizzato comporta un incremento della superficie esposta a potenziali vulnerabilità; per questo, la ricerca dovrebbe iniziare da fonti attendibili per le immagini di base: verifica sempre che l'immagine provenga da un'azienda nota o da un gruppo open source, sia ospitata su un registry affidabile e che il codice sorgente di tutti i componenti dell'immagine sia disponibile per poter verificare le risorse utilizzate. Tuttavia la sicurezza non riguarda solo questo aspetto: è infatti fondamentale giocare di anticipo e correggere le vulnerabilità, tenendo conto degli aggiornamenti forniti sia da chi distribuisce le immagini, sia dagli stessi sistemi operativi utilizzati; inoltre, è buona

prassi utilizzare dei registry privati per le immagini che contengono informazioni sensibili o applicativi proprietari, che poi possono essere distribuite gestendo l'accesso al repository con delle utenze *ad hoc*.
Inoltre, integrare test di sicurezza e meccanismi di automazione per la loro distribuzione rende il lavoro più snello e sposa anche quanto descritto all'interno della filosofia DevOps; ogni manualismo deve lasciare il posto a un lavoro di sviluppo o di miglioramento delle funzionalità già presenti, e non deve togliere tempo ad attività ripetitive che possono essere trattate in maniera automatica. Ultima cosa, ma non meno importante, quando abbiamo a che fare con dei container e con tutte le risorse che ne derivano e di cui questo manuale offre un'ampia descrizione, è fondamentale avere una visione a 360 gradi: non basta utilizzare delle fonti attendibili per le proprie immagini, ma occorre proteggere in maniera opportuna tutta l'infrastruttura. Infatti, il sistema operativo che ospiterà il cluster viene abilitato utilizzando uno strumento di esecuzione dei container, idealmente gestito tramite un sistema di orchestrazione. Per rendere resiliente la piattaforma, soprattutto per ambienti complessi come quelli di produzione dedicati agli utenti finali, è fondamentale seguire alcune *best practice*, tra cui prevedere dei meccanismi di autenticazione e autorizzazione, isolamento dei progetti, e così via.
Da quanto descritto finora, sono emerse delle esigenze importanti, che Docker non sempre riesce a soddisfare: sappiamo che l'adozione di un pattern architetturale a microservizi implica la necessità di lavorare con più di un container, e quindi di monitorare più "ambienti". Avere uno strumento che ne possa centralizzare la gestione e ci fornisca un aiuto quando (e se) ci saranno errori, diventa fondamentale. In più, possiamo anche affermare che difficilmente ci capiterà di avere a che fare con una sola applicazione contenuta in un container che non abbia bisogno di comunicare con altri servizi e che quindi richieda una configurazione più complessa: non parliamo solo del modo in cui l'applicazione viene eseguita, ma anche di come si coordina con altri componenti applicativi. E se poi il nostro volume di traffico o di utenti crescesse? Come possiamo rendere l'applicazione adatta a gestire un flusso di lavoro che cambia nel tempo? L'orchestrazione rappresenta la configurazione, la gestione e il coordinamento automatizzato di sistemi informatici, applicazioni e servizi. L'orchestrazione, nel senso più generico del termine, aiuta l'IT a gestire più facilmente attività e flussi di lavoro complessi. I team IT devono gestire molti server e applicazioni, ma farlo manualmente non è una strategia scalabile: più complesso è un sistema informatico, più complessa può diventare la gestione di tutti gli attori. In questo senso, automazione e orchestrazione sono concetti diversi, ma correlati. L'automazione aiuta a rendere più efficiente la tua azienda, riducendo o sostituendo l'interazione umana con i sistemi IT e utilizzando invece il software per eseguire attività al fine di ridurre costi, complessità ed errori. In generale l'automazione si riferisce a una singola attività e questo è ben diverso dall'orchestrazione, che è il modo in cui è possibile automatizzare un processo o un flusso di lavoro che prevede molti passaggi su più sistemi disparati. L'orchestrazione in ambito IT aiuta anche a semplificare e ottimizzare i processi e i flussi di lavoro che si verificano di frequente, soprattutto se supportano un approccio DevOps, con lo scopo di aiutare il team a distribuire le applicazioni più rapidamente.
Come si incastra Kubernetes in tutto ciò? In questo caso, parliamo di orchestrazione delle applicazioni o anche dei container, ossia quando si integrano insieme due o più applicazioni software distribuite tramite container e nel cui contesto è necessario avere dei meccanismi di automazione per la loro gestione complessiva. Questo processo consente di gestire e monitorare in maniera centralizzata le tue integrazioni e di aggiungere

funzionalità per il routing delle applicazioni, per la sicurezza e per l'integrità delle stesse su cui Kubernetes è focalizzato. Ma che cos'è esattamente l'orchestrazione dei container? Tra le funzionalità che ci mette a disposizione uno strumento come questo c'è la pianificazione della distribuzione dei container nel cluster, nonché la gestione del ciclo di vita del container in base alle specifiche stabilite nella loro definizione. Ma perché abbiamo bisogno dell'orchestrazione dei container? E qual è lo scopo dell'automazione e dell'orchestrazione? Ebbene, queste funzionalità consentono di ridimensionare le applicazioni con un singolo comando, creare rapidamente nuove applicazioni containerizzate per gestire il traffico in crescita e semplificare il processo di installazione, migliorando anche la sicurezza. Kubernetes nasce proprio con questo intento: semplificare una gestione altrimenti complessa in cui soluzioni come Docker Swarm o Docker Machine avevano delle "mancanze" in un contesto enterprise.
Ma chi c'è dietro a Kubernetes?

## Kubernetes: breve storia

Le grandi aziende di Internet come Google, Twitter ed eBay hanno utilizzato architetture basate su container per anni; mentre il container, secondo quanto visto finora, offre una soluzione elegante per la creazione di applicazioni basate su microservizi, Docker non include tutto il necessario per distribuire e gestire diversi container che devono lavorare insieme e scalare all'aumentare della domanda. Quando ognuna di queste aziende ha iniziato a creare i propri strumenti per lo sviluppo, a gestirne l'implementazione e il ridimensionamento, ci si è resi conto che il lavoro era eccessivo perché Docker potesse gestirne tutto il carico.
Senza voler rendere questo capitolo noioso, riportiamo solo alcuni dei punti fondamentali che ci permettono di comprendere al meglio la nascita e la crescita di questo ambizioso progetto che, in qualche modo, ci ha portato alla scrittura di questo manuale.

### Nascita di Borg

Google presenta il sistema Borg (`https://research.google/pubs/pub43438/`) intorno al 2003-2004: è iniziato come un progetto su piccola scala, con circa 3-4 persone inizialmente in collaborazione per produrre una versione aggiornata del nuovo motore di ricerca di Google. Borg, invece, era un sistema interno di gestione dei cluster su larga scala, che eseguiva centinaia di migliaia di *task*, provenienti da molte migliaia di applicazioni diverse, su molti cluster, ciascuno con un massimo di decine di migliaia di macchine. Dall'esperienza di Borg, si passa a Omega (`https://research.google/pubs/pub41684/`), un sistema di gestione dei cluster con uno *scheduler* per pianificare il rilascio delle applicazioni flessibile e scalabile indirizzato a cluster di calcolo di grandi dimensioni. Questo strumento, quindi, si integra con Borg, introducendo uno strumento che permette la gestione del parallelismo delle applicazioni, di uno stato condiviso e anche della concorrenza, tutti aspetti fondamentali per il ciclo di vita di un cluster. Questo finché nel 2014 non annunciano il progetto Kubernetes: ormai Google esegue i suoi carichi di lavoro in produzione sfruttando Borg da più di un decennio e praticamente tutto in Google viene eseguito sotto forma di container. Kubernetes traccia una linea diretta con Borg, tanto che molte delle persone che hanno lavorato originariamente al progetto sono le

protagoniste della nascita di uno degli orchestratori più diffusi al mondo. Il 7 giugno del 2014 viene effettuato il primo commit su GitHub relativo al progetto dedicato a Kubernetes ed è subito un successo; appena un mese dopo, aziende del calibro di Red Hat, Microsoft e IBM si uniscono alla community che si crea intorno a Kubernetes.

**NOTA**

In omaggio a una delle serie più geek di sempre, la tecnologia Borg prende il nome dal gruppo di cyborg *Borg*, una specie immaginaria che fa parte dei personaggi più ricorrenti della saga di Star Trek: il fatto che i Borg siano organizzati come una sorta di "mente centralizzata" collettiva ben rappresenta il senso del progetto!

## Arriva il KubeCon

Ben presto nasce anche la prima conferenza a tema: il 9-11 novembre del 2015 si tiene il primo KubeCon a San Francisco, dove si inaugura la creazione della community dedicata a questa tecnologia: il suo obiettivo era (ed è ancora) quello di tenere discorsi tecnici di esperti con l'obiettivo di stimolare la creatività e promuovere l'istruzione su Kubernetes. Nel 2016 diventa quindi un fenomeno, una vera e propria rivoluzione: questo anno segna moltissime novità per questo progetto, come il rilascio della prima versione di Helm, il gestore di pacchetti per Kubernetes, la prima KubeCon EU, ossia la prima conferenza europea di Kubernetes con quasi 500 partecipanti. Man mano che la risonanza aumenta, Kubernetes inizia a sfornare i primi oggetti che rappresentano una novità nel mondo dell'orchestrazione: nascono i PetSet, una risorsa che oggi assume un nome diverso (no spoiler, lo vedremo più in là!), così come lo sviluppo e l'implementazione di Minikube, uno strumento che semplifica l'esecuzione di Kubernetes in locale.
In questo contesto, bisogna parlare anche del caso Pokemon GO! (`https://cloud.google.com/blog/products/containers-kubernetes/bringing-pokemon-go-to-life-on-google-cloud`): infatti, sebbene possa sembrare strano, questo caso d'uso rappresenta la più grande implementazione Kubernetes mai realizzata su Google Container Engine. I Pokemon sono stati nel cuore di ogni millennial fin dall'infanzia. Poiché Pokemon è stato creato dalla nota azienda Nintendo Co., la maggior parte dei giochi è sempre stata disponibile solo su piattaforme Nintendo Switch. Pokemon GO!, invece, è stato il primissimo gioco Nintendo a essere disponibile su dispositivi diversi dai Nintendo Switch, portandolo di fatto sui dispositivi mobili, come cellulari e tablet: questa novità ha generato un enorme scalpore nel mercato prima che il gioco uscisse e, dopo il rilascio, le stime fatte dall'azienda hanno superato di molto qualsiasi aspettativa. Entro 15 minuti dal lancio in Australia e Nuova Zelanda, il traffico di giocatori è aumentato di 6 volte, con un risultato migliore di qualsiasi aspettativa del team. Il clamore è stato tale da indurre i suoi creatori alla pubblicazione di un caso di studio per illustrare la dimensione e diffusione di questo fenomeno e il contributo fondamentale di Kubernetes alla scalabilità delle risorse necessarie per sostenere rapidamente le continue richieste di risorse da parte dei giocatori di Pokemon GO! L'aspetto ancora più interessante è stato poter dedicare a ogni cliente del gioco un "pezzo" del cluster dedicato alle sue attività, isolandolo dagli altri giocatori.

## Kubernetes su cloud

Questa esperienza ha finalmente consolidato il ruolo di Kubernetes come giocatore di punta nel mercato degli orchestratori, rendendolo di fatto l'apripista per altri progetti che si integrano con questa tecnologia, come Prometheus, Fluentd e OpenTracing. Nel 2017 Kubernetes viene adottato in maniera diffusa dalle aziende, tanto che Google e IBM annunciano Istio, una tecnologia open che fornisce un modo per connettere, gestire e proteggere senza problemi reti costituite da diversi microservizi, indipendentemente dalla piattaforma e dall'origine.
Un esempio di grande azienda che adotta Kubernetes per le sue applicazioni è GitHub: tutte le richieste Web e le API vengono servite dai container in esecuzione nei cluster distribuiti on cloud. E, mentre la CNCF incrementa il numero dei suoi membri grazie alla presenza di grandi aziende come Oracle, Kubernetes vede le prime installazioni open source anche su sistemi Windows o Oracle systems e soprattutto le prime implementazioni di cluster completamente o parzialmente gestiti sui principali cloud provider. Amazon annuncia nel 2017 Elastic Container Service per Kubernetes, un servizio per distribuire, gestire e ridimensionare le applicazioni containerizzate direttamente tramite AWS. Nello stesso anno, abbiamo la presentazione di Kubeflow, uno stack di strumenti per attività di machine learning altamente componibile, portatile e scalabile, creato appositamente per Kubernetes.
Nel 2018, DigitalOcean si tuffa nel mondo Kubernetes e annuncia un nuovo prodotto che ospita questa tecnologia per fornire una piattaforma di gestione e orchestrazione dei container come servizio gratuito in aggiunta alle opzioni di archiviazione e cloud computing esistenti. Anche Amazon fa progressi e lancia EKS, un sistema gestito che semplifica il processo di creazione, protezione, funzionamento e manutenzione dei cluster Kubernetes offrendo i vantaggi dell'elaborazione basata su container alle organizzazioni che desiderano concentrarsi sulla creazione di applicazioni invece di configurare un cluster Kubernetes da zero. Anche Microsoft lavora in questo senso, e annuncia quasi contemporaneamente il servizio Azure Kubernetes (AKS), per distribuire e gestire le proprie app Kubernetes di produzione.

### La squadra dietro Kubernetes

Kubernetes nasce come strumento open source e ha una community molto attiva per quanto riguarda la manutenzione e la gestione dei registri pubblici e privati, oltre che della documentazione. Infatti, più di 1400 collaboratori di diverse aziende come Red Hat, Google e Microsoft fanno parte dell'elenco di collaboratori "stabili" del progetto Kubernetes. Recentemente, anche Alibaba e Amazon sono entrate a far parte della cerchia delle grandi aziende che adottano questa tecnologia. La fondazione del cloud computing supervisiona comunque questa tecnologia nel suo complesso. Inoltre, aziende leader come Intel, Mozilla, Pivotal, Oracle e molte altre stanno contribuendo con il loro codice a questa tecnologia dalla connotazione fortemente infrastrutturale.

Google ha rilasciato Kubernetes come progetto open source con l'obiettivo di standardizzare la gestione delle applicazioni containerizzate; Kubernetes, spesso abbreviato in K8S, funge da *overlay* rispetto al data center di un'azienda e i diversi progetti potrebbero

lavorare sulla risoluzione di altri aspetti della gestione dei *container* (come la creazione di sistemi operativi semplificati o strumenti di amministrazione grafica).

### Kubernetes = K8S?

Ti sei chiesto perché Kubernetes viene abbreviato in K8S? I cosiddetti "numeronimi" sono apparsi alla fine degli anni Ottanta. Ci sono diverse storie su come le persone abbiano effettivamente iniziato a usarli; tuttavia, l'idea condivisa è che, per semplificare la comunicazione, il settore IT abbia iniziato a utilizzare i numeronimi per abbreviare le parole: prendere la prima e l'ultima lettera di una parola e collocarvi nel mezzo un numero che ne desse il suono corretto riduceva di molto il lavoro. Per esempio, il termine "*i18n*" deriva dall'ortografia di "*internationalization*", che inizia con la lettera "i", continua con18 lettere e finisce con la lettera "n". In modo simile, Kubernetes è composta dalla lettera "k" più 8 lettere e infine la lettera "s".

A oggi, Kubernetes è molto popolare. Secondo lo State of Cloud Native Development Report di CNCF (Cloud Native Computing Foundation) per il primo trimestre del 2021, ci sono 6,8 milioni di persone che sviluppano in ambito cloud in tutto il mondo, di cui 5,6 milioni utilizzano Kubernetes, con una crescita del 67% rispetto all'anno precedente, con 10,2 milioni di container in esecuzione. La CNCF ha inoltre stimato che ci sono 26,8 milioni di persone che sviluppano a livello globale, quindi circa 1 utente su 5 nel mondo utilizza attualmente Kubernetes.

### Il timoniere di cui Docker aveva bisogno

Il nome "Kubernetes" deriva da un'antica parola greca che sta per "timoniere" (qualcuno che dirige una nave, come una nave per il trasporto di container) e ritroviamo questo nome nel logo del progetto che mostra il timone di una nave. Questo ha sette lati, in omaggio al nome originale del progetto; Beda, McLuckie e il suo team inizialmente intendevano chiamare la tecnologia Kubernetes *Project Seven*, che prende il nome da Seven of Nine, un simpatico personaggio di *Star Trek* appartenente al gruppo dei Borg.

## Perché Kubernetes

Ormai è chiaro: i container sono un modo semplice e valido per raggruppare ed eseguire le tue applicazioni. In un ambiente di produzione, però, è necessario gestire i container che eseguono le applicazioni e assicurarsi che non vi siano tempi di inattività. Per esempio, se un container si arresta, deve essere avviato un altro container in sostituzione del primo. Per farlo, potresti trovarti nelle condizioni di dover entrare tramite SSH sul server, avviare il container e poi verificare che tutto stia funzionando a regola d'arte. È evidente quanto sarebbe più semplice se questo comportamento fosse gestito da un opportuno sistema: Kubernetes si occupa della scalabilità e del *failover* per la tua applicazione, fornisce dei modelli per agevolare il rilascio delle applicazioni secondo

delle policy predefinite e molto altro ancora. Tra le diverse funzionalità, abbiamo uno storage gestito per il cluster, *rollout* e *rollback* automatizzati per adattarne lo stato effettivo a quello desiderato a una velocità controllata, resilienza tramite dei test che controllano l'integrazione degli applicativi e molto altro.
Senza indugiare ulteriormente, voliamo verso il mondo dell'orchestrazione, dando prima un'occhiata alle soluzioni presenti sul mercato: non vogliamo che Kubernetes rappresenti la soluzione a tutti i nostri problemi, ma che sia una scelta consapevole e adatta al nostro caso d'uso!

## Che cosa abbiamo imparato

- Abbiamo visto come avviene storicamente il passaggio da strumenti di containerizzazione al contesto di orchestrazione.
- Abbiamo ripercorso i punti salienti del successo di Kubernetes, attraverso le aziende che hanno sposato il progetto e i suoi casi d'uso.
- Abbiamo introdotto le principali funzionalità che lo rendono uno strumento perfetto per i nostri scenari di lavoro in determinati contesti, che verranno approfonditi a breve.