ordine INGEGNERI provincia como

C3i COMITATO ITALIANO INGEGNERIA INFORMAZIONE

LINUX CONTAINER, QUESTI SCONOSCIUTI

Introduzione alla tecnologia CLOUD NATIVE
che potrebbe cambiare il tuo modo di lavorare

Documentazione per il corso

ONLINE IL 10 APRILE 2021,
DALLE H. 9.00 ALLE H. 13.00

Questo documento è liberamente tratto dal corso DO080, erogato in esclusiva da EXTRAORDY e dedicato ai **Linux Container Kubernetes** ed al mondo di **OpenShift;** è pensato per conoscere meglio quelle che sono le basi del mondo cloud poiché tutto quello che parte dal mondo cloud ready dell'iceberg hybrid cloud parte sempre da **Linux**.

INDICE

DOCUMENT INFORMATION

ORIGINATOR

EXTRAORDY Srl

OWNER

EXTRAORDY Srl – Confidential; Restricted Distribution.

DISTRIBUTION

Do not forward or copy without written permission.

CONFIDENTIALITY

This is a confidential document between **EXTRAORDY S.r.l**. and the user**.** ("Client").

All information supplied to the Client for the purpose of this project is to be considered confidential.

DISCLAIMER

This document does not represent an official Statement of Work.

UN TEAM CERTIFICATO

I servizi certificati forniti da EXTRAORDY vengono erogati da un team le cui figure senior hanno oltre 20 anni di esperienza su clienti Enterprise ed erogano regolarmente formazione e consulenza ufficiale **Red Hat**.

Tutto il team altamente specializzato di consulenti **EXTRAORDY** possiede una o più seguenti certificazioni ufficiali:

- *Red Hat Certified Architect*
- *Red Hat Certified Specialist in Ansible Automation*
- *Red Hat Certified Instructor*
- *Red Hat Certified Examiner*
- *Red Hat Certified Engineer*
- *Red Hat Certified System Administrator*
- *Red Hat Certificate of Expertise in Platform-as-a-Service*
- *Red Hat Certified Virtualization Administrator*
- *Red Hat Certified Datacenter Specialist*
- *Red Hat Certified JBoss Administrator*
- *Red Hat Certified Enterprise Application Developer*

INTRO

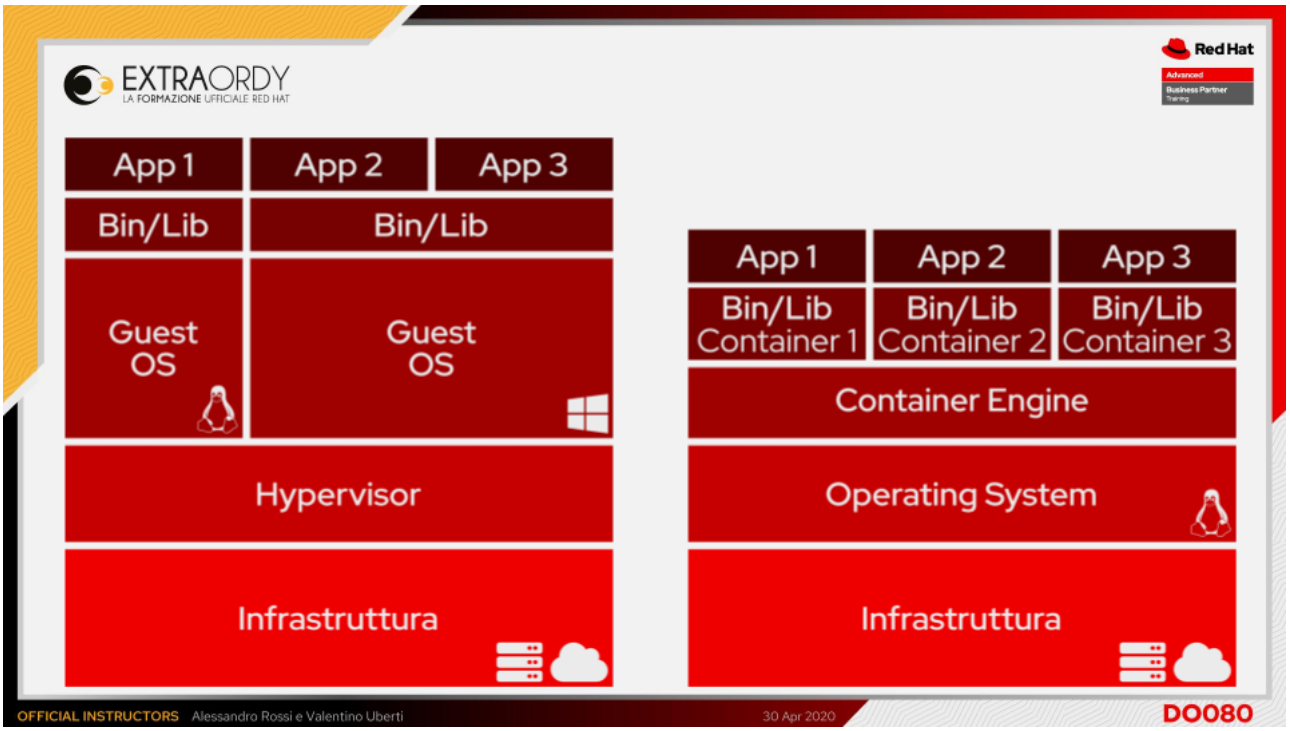
Condividere conoscenza è la mission di **EXTRAORDY**, proprio per questo è presente un canale youtube dove periodicamente vengono pubblicati contenuti inerenti a tutto quello che è l'open source, il cloud e il mondo del cappello rosso.

Gli istruttori di **EXTRAORDY**, istruttori ufficiali Red Hat, si occupano di formazione per quanto riguarda DevOps e OpenShift.

L'obiettivo è sicuramente quello di trasmettere la passione e l'entusiasmo del lavoro degli istruttori **EXTRAORDY**, che con grande impegno e lavoro di squadra quotidiano aiuteranno a comprendere e conoscere meglio quelle che sono le basi del mondo cloud.

MACCHINE VIRTUALI VS CONTAINERS

Secondo gli istruttori **EXTRAORDY** quando si introducono nuove tecnologie è sempre utile andare subito a comprendere che problema risolvono, cioè perché si dovrebbe adottare una particolare tecnologia. Per fare questo è utile fare un confronto con il deploy di un applicativo delle applicazioni utilizzando delle macchine virtuali.



A sinistra si trova un semplice stack che utilizza le macchine virtuali dove si trova un hypervisor dove si va ad installare la macchina virtuale che possono contenere due sistemi operativi differenti.

Ad esempio si suppone di avere due macchine virtuali, la prima con un sistema operativo Linux e la seconda con sistema operativo windows. Tipicamente quindi le applicazioni vengono installate a posteriori su questi sistemi operativi ma con questo deploy si possono avere dei problemi che vengono risolti grazie all'adozione dei container. Se si prende l'esempio del sistema operativo windows può capitare che in caso si avesse il deploy di due applicazioni sullo stesso sistema operativo, si potrebbero avere dei problemi per esempio quando si aggiorna una libreria condivisa perché tipicamente le applicazioni utilizzano le librerie condivise dal nostro sistema operativo ma in questo caso se si ha

l'aggiornamento di una libreria condivisa si può avere l'applicazione 2 completamente funzionante ma magari l'applicazione 3 che ha dei problemi.

Il primo grande problema all'utilizzo di librerie condivise.

Un altro problema è che se per caso l'applicazione numero 2 viene compromessa, l'attaccante potrà vedere i dati anche dell'applicazione 3 perché non esiste un isolamento fra l'applicazione 2 e l'applicazione 3 e anche questo problema sarà risolto grazie all'utilizzo dei container.

Un altro problema è relativo a cosa accade se si deve aggiornare il sistema operativo: per esempio, nella macchina virtuale si dovranno adottare dei meccanismi che grazie all'utilizzo dei Linux container sono decisamente più semplici da attuare.

Un altro problema è il tempo di start up perché quando la macchina virtuale viene avviata si deve aspettare del tempo di caricamento del sistema operativo scelto che gira nella macchina virtuale e il tempo effettivo necessario all'applicazione per essere messi al running.

Parlando di risorse, andando a utilizzare un approccio con macchine virtuali si richiedono più risorse a livello di ram e di spazio su disco perché oltre le risorse consumate dalla applicazione si deve tener conto anche delle risorse utilizzate dal sistema operativo sottostante e quindi tutti questi problemi sono risolti grazie all'utilizzo dei container.

Quando si parla di containers si deve tenere bene a mente che si parla di processi Linux quindi sono processi che hanno bisogno di un kernel Linux.

Parlando quindi di processi, non si ha bisogno di tutte macchine virtuali ma di un kernel Linux che possa mettere il running dei processi esattamente come tutti gli altri processi Linux.

In realtà, però, i **processi container hanno delle caratteristiche fondamentali:** prima di tutto **i processi container sono isolati dagli altri processi container** ovvero l'applicazione 1 è completamente isolata dall'applicazione 2: se un attaccante riesce ad hackerare l'applicazione 1 non avrà mai visibilità dell'applicazione 2 e questo avviene grazie a delle features del kernel Linux.

Un altro problema risolto grazie ai containers è che **si possono inserire librerie e runtime che servono solo all'applicazione.** Nella colonna di destra l'applicazione 1 utilizzerà solo le librerie di cui necessita e queste librerie non sono condivise con altre applicazioni quindi, a livello di security, è veramente un ottimo boost.

I processi container sono isolati anche dall'hardware e questo vuol dire che possono girare su piattaforme che rendono disponibile un kernel Linux e quindi un processo contener può girare sui laptop, sulle workstation e in cloud: l'unico requisito è che ci sia un kernel Linux.

Il consumo di risorse è decisamente inferiore andando a utilizzare i containers Linux perché non si hanno le macchine virtuali quindi non si ha tutto lo stack di virtualizzazione.

Un altro grande grande vantaggio sono i tempi di start up perché non si ha la necessità di attendere la fase di boot di un sistema operativo e quindi l'applicazione può essere messa in running in un tempo decisamente inferiore paragonato appunto a una classica l'infrastruttura che utilizza le macchine virtuali.

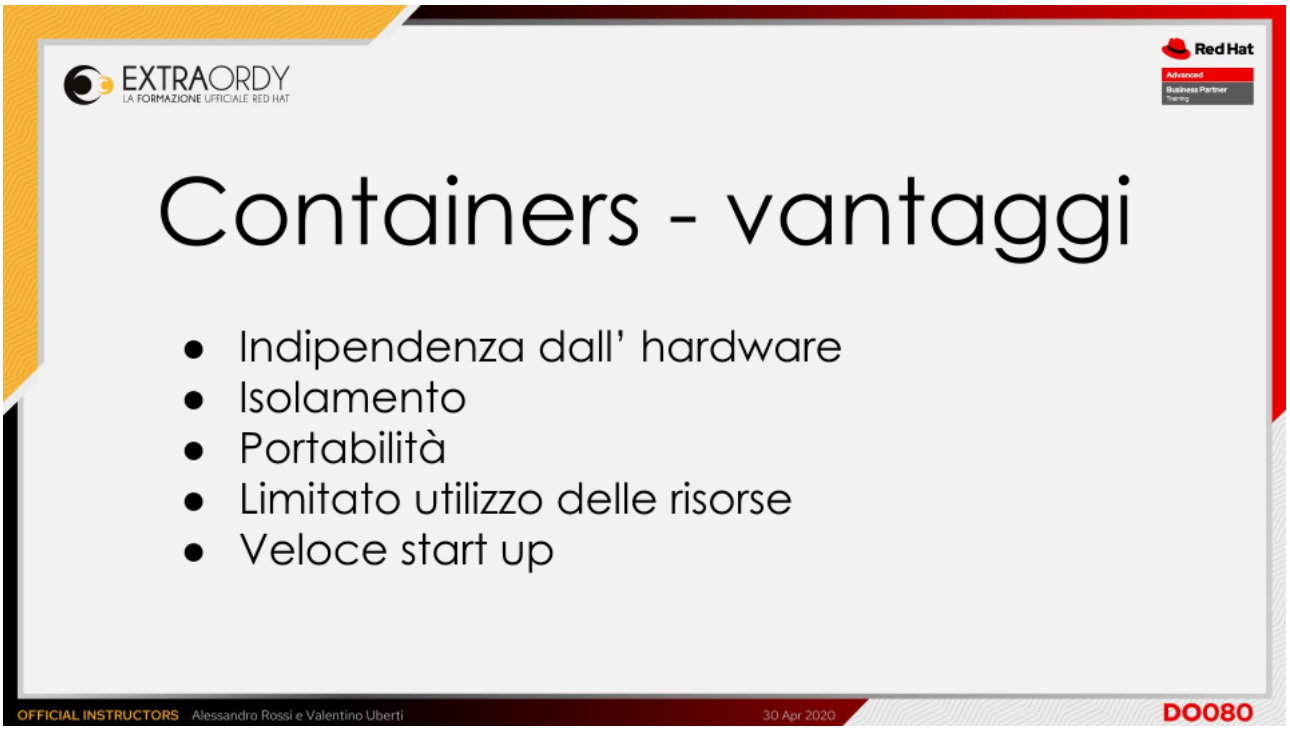
Questi sono solo alcuni dei problemi che vengono risolti grazie all'approccio dei Linux container.

Riassumendo quindi i vantaggi, i Linux container essendo processi, vengono definiti in maniera indipendente dall'hardware ma anche dalle distribuzioni Linux.
Sono processi isolati a livello di networking e a livello di processo e tutto questo porta a un enorme portabilità.

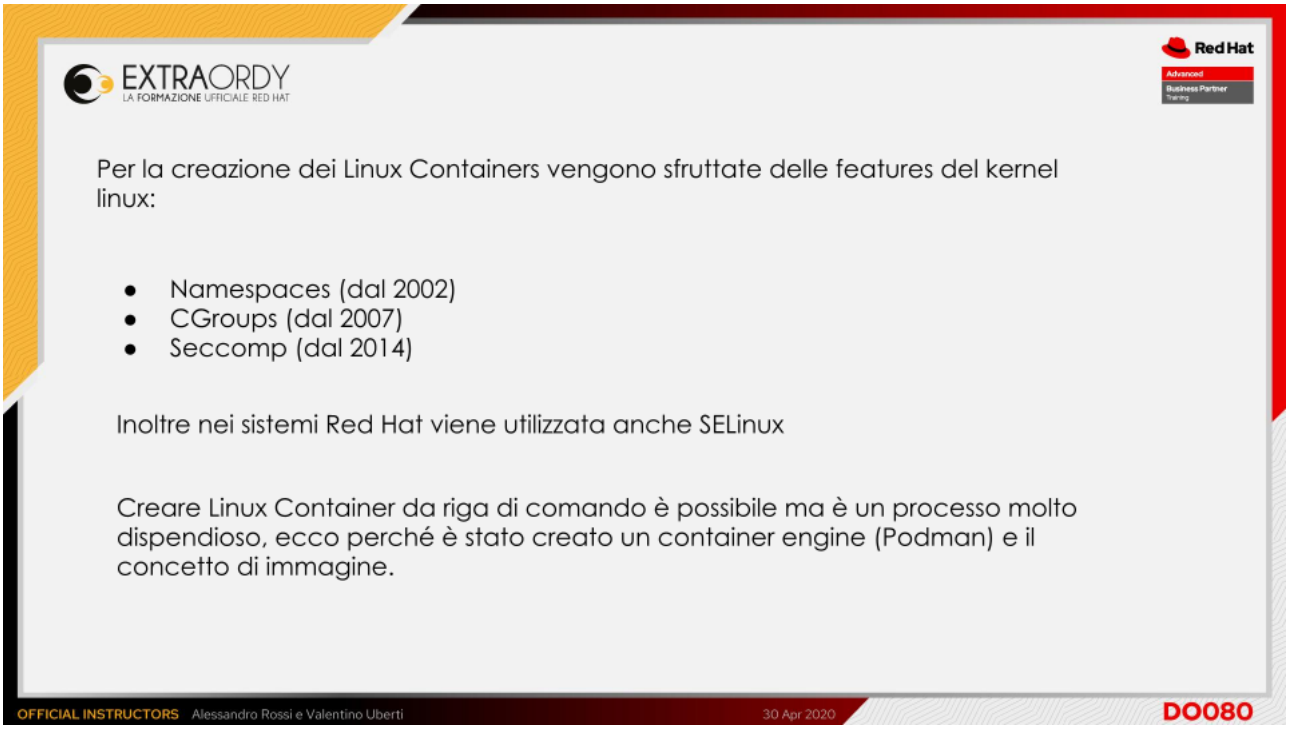
Un effetto molto importante dell'utilizzo del containers è come vengono distribuite le applicazioni: non ci si deve più preoccupare se una distribuzione usa apt o altri pact manager perché una volta inserita di un container si saprà che potrà essere messa in running su tutte le piattaforme che hanno un kernel Linux.

Tutti i processi containers possono essere limitati nell'utilizzo delle risorse: se si decide che una applicazione, all'interno di containers, al massimo potrà utilizzare 12 gigabyte di ram e se per caso si presenta un errore applicativo e quindi l'applicazione ha un bug e incomincia a fare dei forti di processi e ad utilizzare tutte le risorse, queste risorse vengono limitate e quindi anche a livello security, grazie al containers, questo problema sostanzialmente non si pone.

Poiché non si ha tutto lo stack delle macchine 2, **il tempo di start up è decisamente più veloce.**



Un altro aspetto importante è che **i processi containers vanno ad utilizzare delle tecnologie che sono presenti nel kernel Linux** e questa è una lista di tecnologie utilizzate per creare dei processi containers.



Alcune di queste caratteristiche, utilizzate dai containers, sono innanzitutto **la presenza dei Namespaces che sono presenti fin dal 2002 e la cui funzionalità è quella di isolare un processo,** ovvero isolarlo per la parte di networking e che non avrà visibilità a livello di networking. Sarà un processo isolato anche a livello di host name, di utenti e di process-id. **Il CGroups è un'altra feature del kernel Linux ed è grazie ad esso che un processo Linux può essere limitato a livello di risorse** ovvero si può, per esempio, dire che un certo processo al massimo può utilizzare 1 gb di ram.

Un'altra funzionalità importantissima è la Seccomp, grazie alla quale si riesce a filtrare le SystemCall che un processo può utilizzare. E' molto utile perché, per esempio, in caso di attacco, l'attaccante non potrà mai mettersi in ascolto per scambiare informazioni.

La seconda è piuttosto complessa perché va a utilizzare dei filtri livello kernel ma si hanno delle utility molto semplici da utilizzare che si preoccupano per gli istruttori di andare a utilizzare tutte queste features del kernel Linux.

Si possono creare dei Namespaces a mano per un processo con un'utility in Linux che si chiama ip netns che permette di creare dei processi isolati. Si possono creare a mano anche un processo e limitarlo grazie ai CGroups e dei profili Seccomp.

Tutte queste features vengono controllate da delle utility come Podman. Per chi ha utilizzato docker, Podman ha gli stessi comandi a tal punto che viene effettuato un link simbolico chiamato docker e che offre molti vantaggi rispetto a docker, come ad esempio il fatto che Podman non richiede un demone in esecuzione. **Grazie a Podman si possono creare e gestire i Container.**

IMMAGINI

Le applicazioni, per poter "girare" nel container hanno bisogno di librerie a runtime. Si può crearne a mano o andare a utilizzare in concetto di immagine: **si può pensare ad un'immagine come ad un template dove all'interno sono inserite tutte le librerie, le runtime, il file di configurazione ed eventuali codici sorgente che servono all'applicazione per essere messa in running.**

Le immagini sono proprio come se fossero un progetto infatti, molto spesso, si trova la parola blueprint per indicare il progetto dell'immagine. Per chi ha un background di programmazione semplicemente si potrebbe pensare alle immagini come se fossero una classe e il container come l'istanza di questa classe.

Le immagini sono inserite in cataloghi che possono essere pubblici o privati chiamati registry e questo perché nel mondo della community sono state create migliaia e migliaia di immagini che gli utenti possono riutilizzare o possono utilizzarle come immagini di partenza per poi andare ad inserire le librerie.

I registry hanno diverse categorie, soprattutto in ambito enterprise, quindi si deve avere una fonte certa e controllata delle immagini. Nell'immagine sono presenti delle librerie del codice che verrà eseguito e in ambito enterprise questo codice deve essere controllato non si può per esempio inserire il running di un'immagine proveniente da un registro pubblico.

Red Hat mette a disposizione un proprio catalogo per i clienti Red Hat dove le immagini vengono controllate e verificate: viene effettuata un'analisi di sicurezza delle librerie interne e vengono corrette da un'eventuale presenza di qualche bug e quindi il cliente enterprise riceve un prodotto controllato e pronto per la produzione.

Red Hat ha creato un registro pubblico chiamato "Quay" dove ciascun utente può registrarsi e le immagini però non vengono controllate come nel Red Hat Container Catalog, ma viene utilizzato uno scanner di vulnerabilità molto efficace che si chiama Clair in fase di upload ovvero quando l'utente utilizza il registro Quay e carica un'immagine che ha creato, in automatico viene controllata da questo scambio che si chiama Quay.

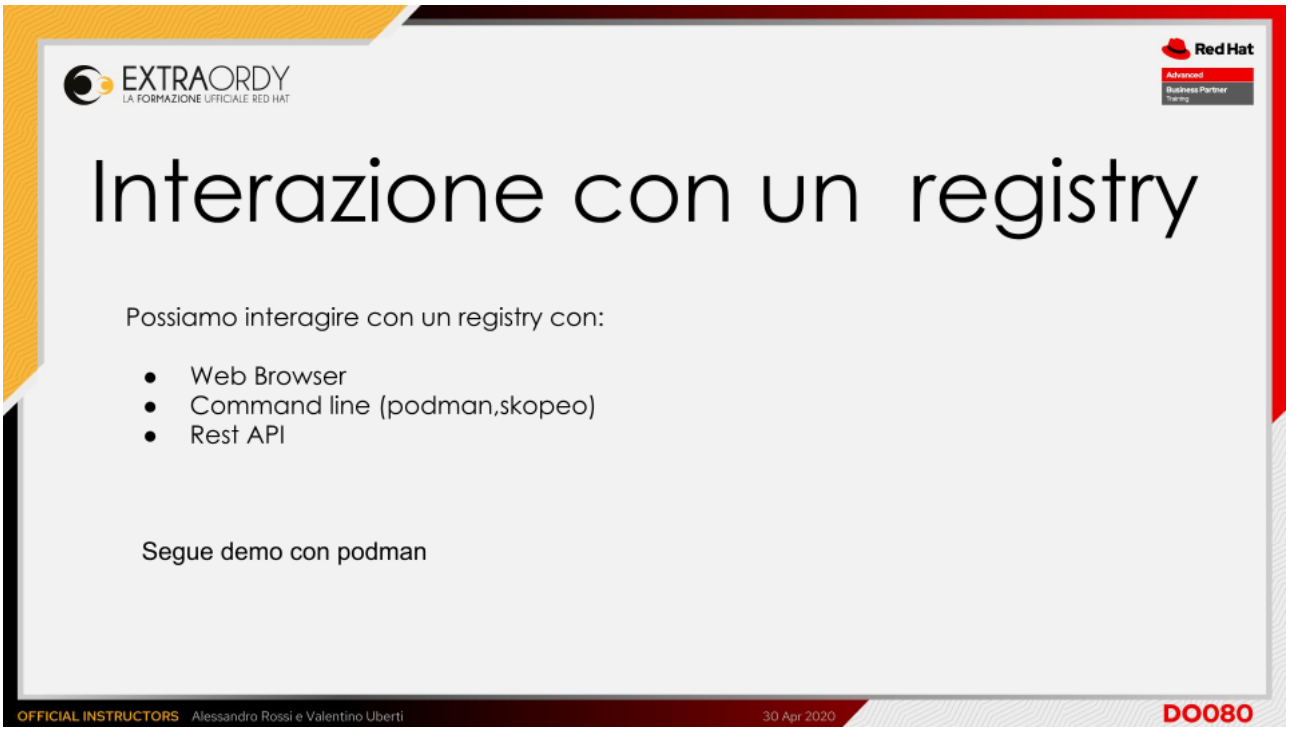
Infine, un altro esempio è **il registro storico Docker Hub, aperto a tutto il pubblico dove non vi è un controllo di provenienza dell'immagine quindi non va bene per un ambito enterprise.**

Questi registry si presentano come dei siti web su dei cataloghi a cui ci si può interfacciare in due modi, ma ne verranno illustrati due:

1- ci si può collegare al sito web cercare l'immagine di cui si ha bisogno;
2- il secondo modo è tramite la linea di comando andando a utilizzare l'utility Podman.

Esistono registry pubblici o privati e registry che richiedono l'autenticazione.

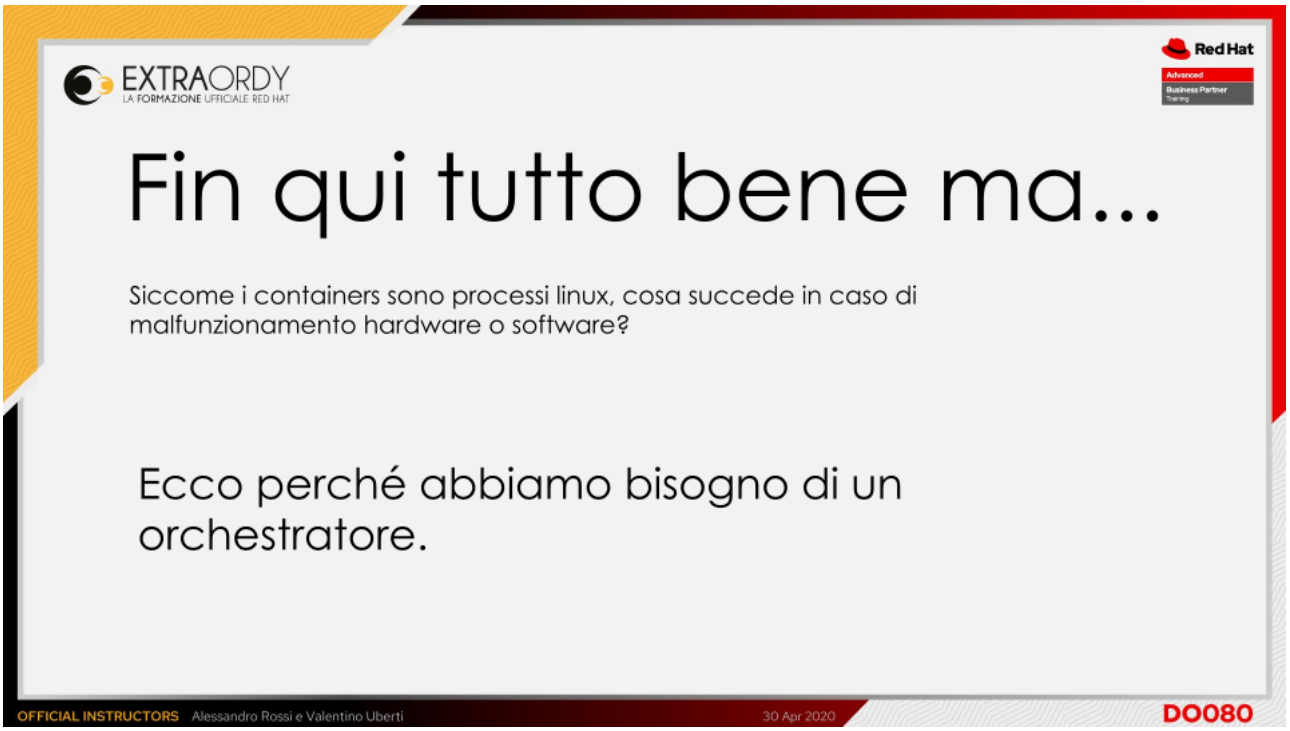
In fase di installazione Podman crea un files conf dove vengono elencati i registry che devono essere utilizzati per la ricerca. Si può scegliere di inserire il registro di red Hat.
Si ha, in realtà, anche una terza opzione come ormai tutti i prodotti, specialmente i prodotti Red Hat, e ci si può interfacciare con prodotti rivelate tramite interfaccia restful API.



Podman è utilissimo per l'integrazione con applicazioni, per esempio, di terze parti e quindi il web browser, la Command line o, se si vuole approfondire, con API rest.

ORCHESTRATORI

I container sono dei processi Linux che girano sulla macchina ma bisogna ricordarsi che ci si trova in ambito enterprise e quindi non bisogna dare un disservizio ai Clienti.
Bisogna avere dei tool, che rientrano nella categoria degli orchestratori, che permettono di usufruire di strumenti per evitare, fra le altre cose, anche il disservizio dell'applicazione.



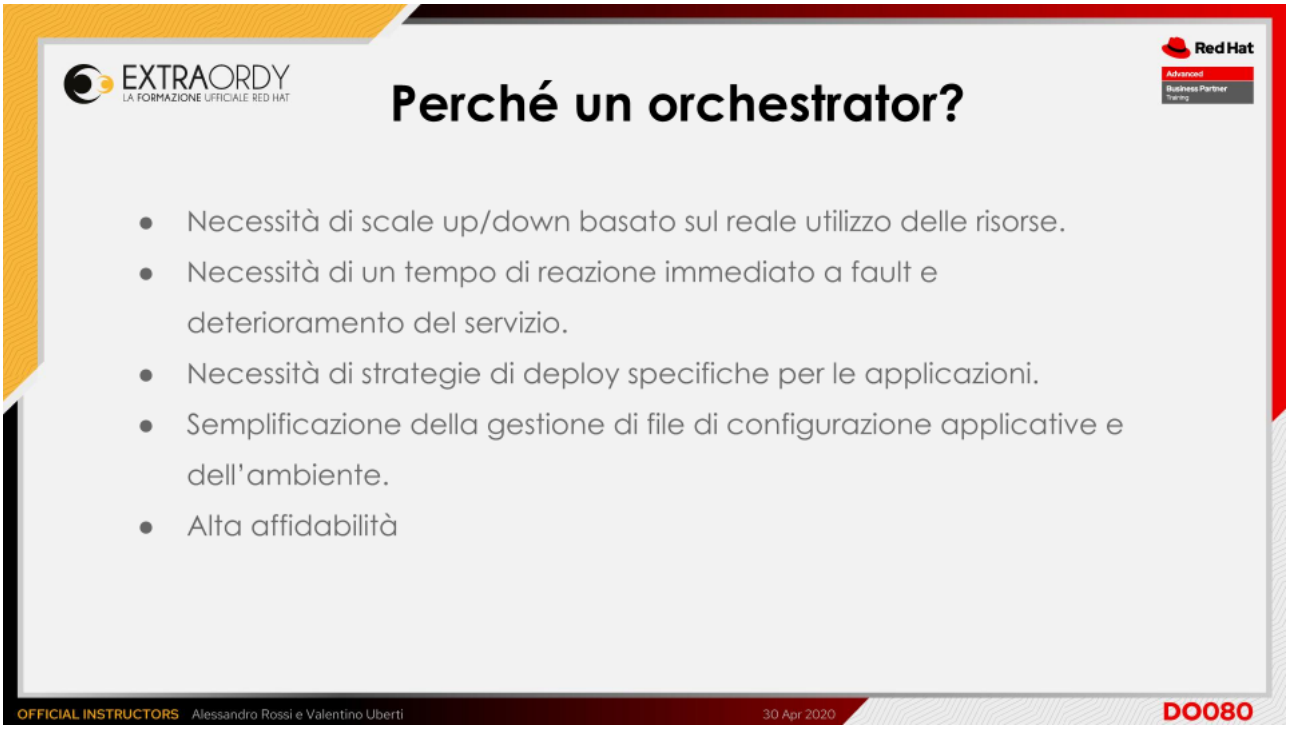
Dopo una overview di quelle che sono le differenze di un container rispetto alle virtual machine e dei vantaggi dati dai Containers, si potrebbero però avere dei problemi nel momento in cui si vanno a deployare le applicazioni. A meno che non siano delle applicazioni stand-alone che non si interfacciano quindi con nessun altro servizio, potrebbe andare tutto bene.

Il problema è che in ambito enterprise si deve andare a garantire un certo livello di servizio e un determinato tipo di reazione a eventuali problemi e agli eventuali fault delle applicazioni.

Si potrebbe avere un problema perché per ogni pod che viene creato viene generato un determinato ip e ogni singolo pod vive di vita propria.

Si può farlo relazionare con altre applicazioni coesistenti come ad esempio nello stesso ambito applicativo e quindi immaginarsi un front end e un back end e un IP che potrebbero essere tre pod diversi che parlano attraverso di loro magari referenziando il nome del servizio.

Un grosso problema potrebbe essere quello di come fare a gestire ad esempio più repliche delle applicazioni poiché essa non deve essere basata esclusivamente su un pod che sta girando perché se va giù quel pod si ha un grosso problema.



Si ha quindi bisogno di un qualcosa a un livello superiore che vada a gestire, orchestrare e garantire che effettivamente tutta l'infrastruttura debba essere gestita da un qualcosa che sia in grado di farlo. In questo momento vengono in soccorso quelli che sono gli orchestratori perché si potrebbe aver bisogno ad esempio di scalare la nostra applicazione a 10, 15, 20 repliche a seconda dell'effettivo traffico e utilizzo dell'applicazione.

L'orchestratore aiuta, attraverso dei controlli cadenzati, a tenere sotto controllo lo stato di salute dell'applicazione e va, eventualmente, ad avvisare attraverso un alert, ad esempio al sistema di monitoraggio, per segnalare che effettivamente c'è stato qualche problema.

Potrebbe essere utile andare a costituire quella che è una strategia di deploy diversa ed eseguire lo switch solamente una volta che si è sicuri che il deploy sia andato a buon fine
Si potrebbe aver bisogno di fare A/B testing ad esempio mantenere più applicazioni in parallelo nello stesso momento con diverse versioni e magari bilanciarne il traffico per vedere effettivamente come risponde ai test.

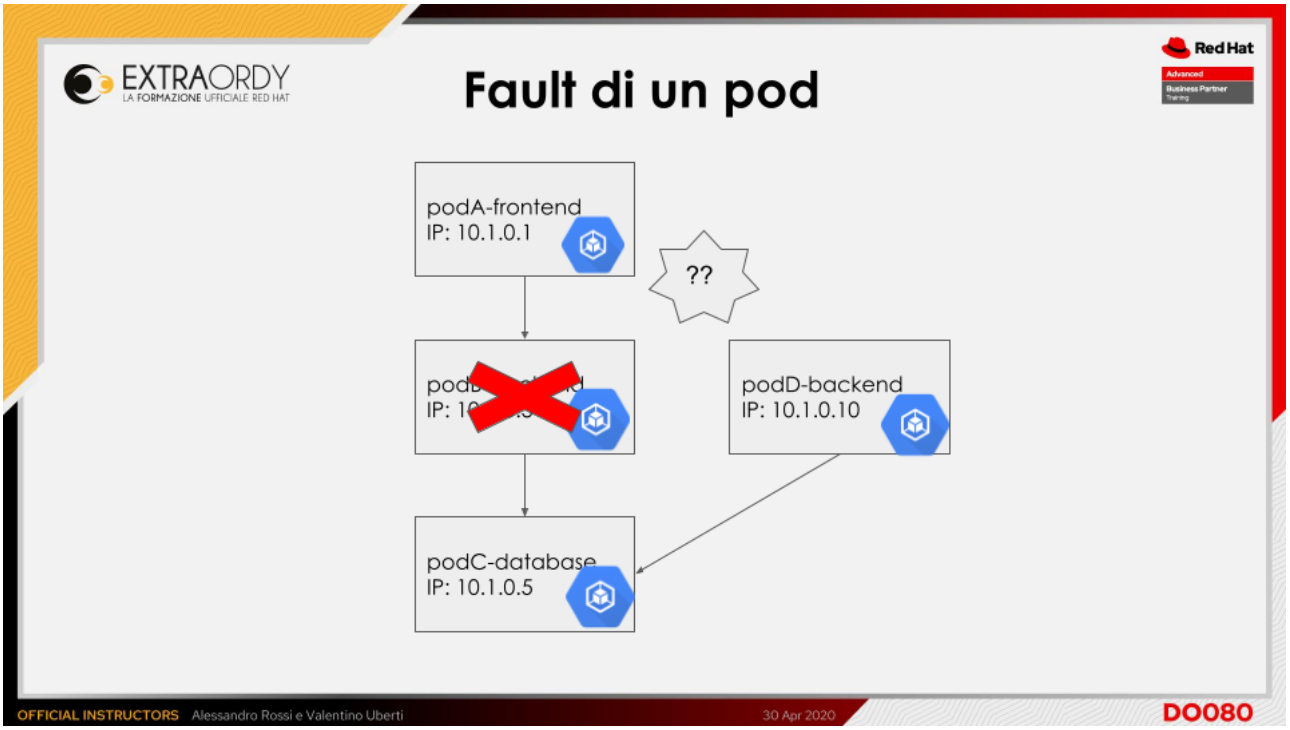
Si potrebbe aver bisogno di qualcuno che gestisca la parte di file di configurazione: si possono avere ad esempio delle applicazioni che portano dietro o all'interno dell'ambiente in cui vanno ad operare numerosi file di continuazione e variabili d'ambiente. Questo è un po un limite alla portabilità nel senso che avere un qualcosa che consenta di andare a salvare le informazioni di configurazione per poterle poi replicare e magari salvarle all'interno di un repository.
Un orchestratore garantisce che, configurando opportunamente degli alert e delle strategie di monitoraggio, si può andare a tenere sotto controllo l'infrastruttura.

KUBERNETES

L'applicazione è composta da un front end da back end e da un IP. Si decide di portarla all'interno di un container per poterla deployare sulla infrastruttura.

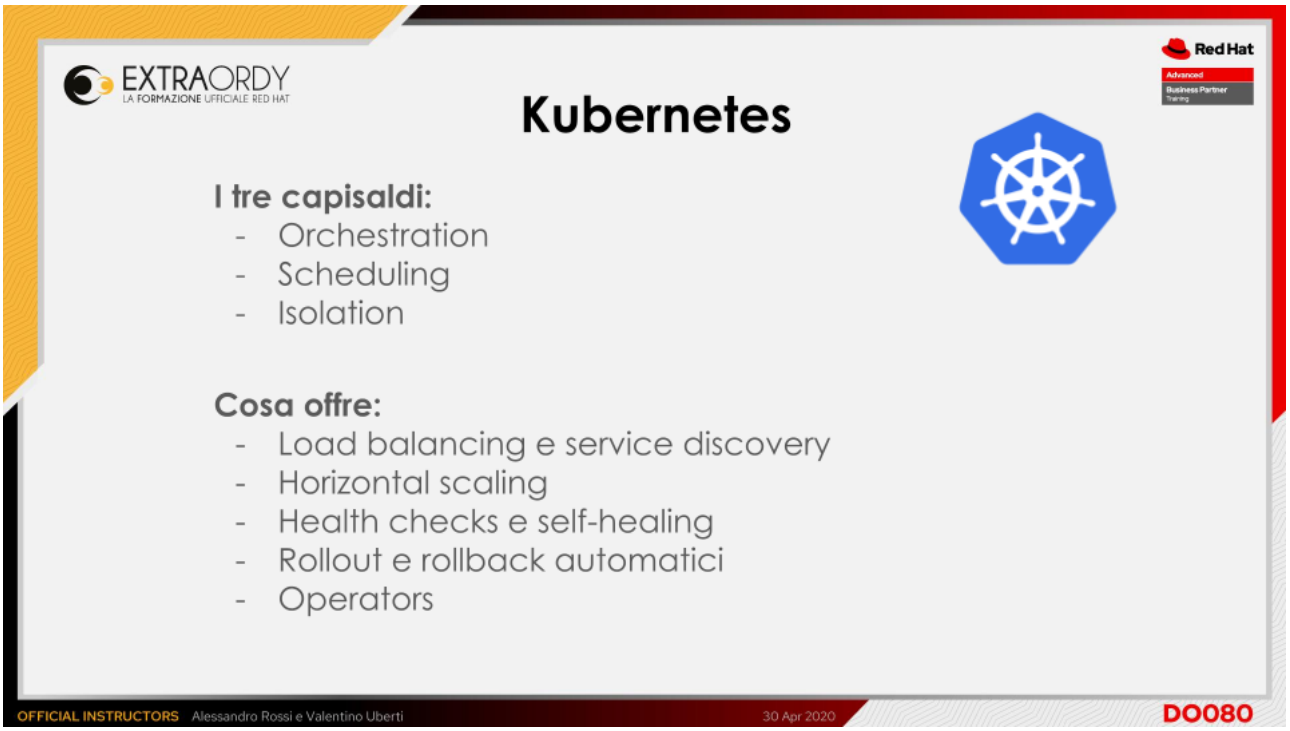
In questo caso non si gestisce la situazione con un orchestratore ma sono stati creati tre container. Ad ogni container è stato assegnato un indirizzo IP ed essi interagiscono tra di loro referenziandosi con l'indirizzo IP.

Immaginiamo che il back end vada giù, quello che si può fare è creare un nuovo pod di backend.



Il problema sarà quello che il front end non conoscerà più qual è l'indirizzo di backend.

Il problema si può risolvere tramite **Kubernetes** che consente 3 capisaldi:



1. **Orchestrazione: farsi carico di gestire quella che è l'interazione tra le diverse risorse**

2. **Scheduling: per creare nuovi pod ed assicurarsi che lo stato dell'applicazione sia quello desiderato.** Ad esempio, se si vogliono 10 repliche della nuova applicazione, l'orchestratore deve essere in grado di garantire che le 10 repliche rimangano effettivamente attive per tutto il tempo e potrà farsi carico di andare a gestire una situazione di upscaling in base all'utilizzo di risorse. Ad esempio quando l'applicazione avrà più bisogno di risorse, l'orchestratore potrebbe decidere, attraverso policy stabilite dall'utente, di andare ad aumentare il numero di pod o diminuirne il numero a seconda dell'utilizzo.

3. **Isolamento: per essere sicuri che l'applicazione, qualsiasi sia il suo stato e qualsiasi sia la sua possibilità di fallire o meno, non deve inficiare quello che è il funzionamento delle altre applicazioni che coesistono all'interno della struttura.**

Tutto ciò offre innanzitutto il load balancing e service discovery, creando delle entità superiori ovvero i service, che permettono di andare a bilanciare il traffico all'interno dei

progetti in maniera dinamica e con risorse allocate e gestite direttamente dall'orchestratore.

Con l'horizontal scaling vi è sia la possibilità di scalare orizzontalmente all'applicazione e quindi richiedere un numero maggiore o minore di repliche e sia di gestire la salute le applicazioni.

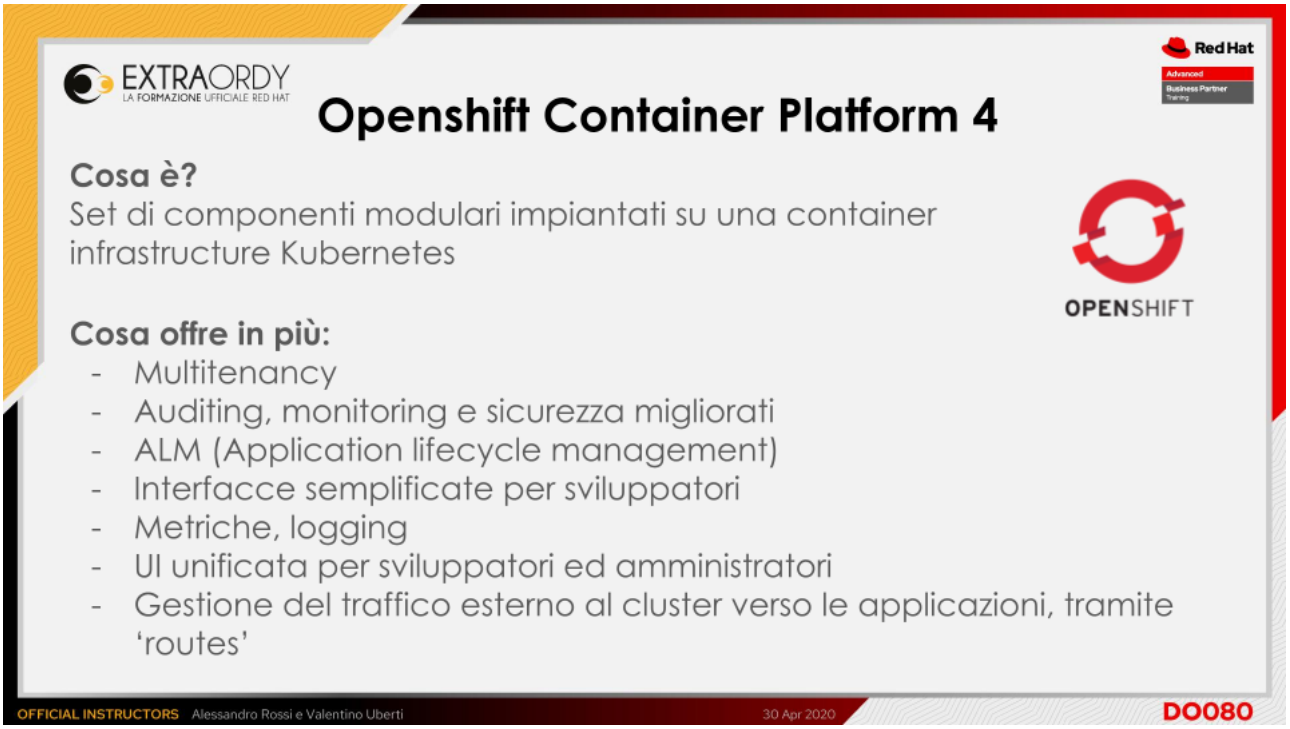
Ciò che si fa con gli Healt checks e self healing non è un controllo puramente applicativo. L'orchestratore, infatti, non sa quello che viene portato all'interno dei container quindi non può sapere come funziona l'applicazione e perché potrebbe essere considerata pronta ad erogare servizio oppure funzionante o non funzionante. Si potrebbe, all'interno dell'applicazione, ad esempio prevedere degli end point dedicati.

Con il rollout e rollback dell'applicazione se si vanno a deployare le applicazioni all'interno di un pod, il problema che potrebbe verificarsi è che se la nuova applicazione per qualsiasi motivo non funziona, non si ha un modo per fare un rollback immediato o comunque un rollback che non ci permetta di non interrompere il servizio.

Inoltre in Kubernetes e OpenShift si trova il concetto di Operator: gli operatori sono delle super applicazioni che permettono di andare ad inserire all'interno del pacchetto applicativo anche della logica per quanto riguarda la parte di gestione e controllo delle risorse o di eventuali risorse create proprio dall'applicazione. L'operator ha quindi il pieno controllo sulle risorse avendo il modo di andare a gestire quelle che sono le eventuali problematiche o eventuali punti di necessità di miglioramento a livello di stabilità.

OCP - OPENSHIFT CONTAINER PLATFORM

Sono state costruite in maniera egregia delle **risorse e delle tecnologie aggiuntive che vanno a migliorare di molto quella che è l'esperienza a livello di orchestratore**



Viene introdotto il concetto di Multitenancy dove vi è il completo isolamento tra i progetti permettendo di andare a gestire l'accesso alle risorse e la sicurezza.

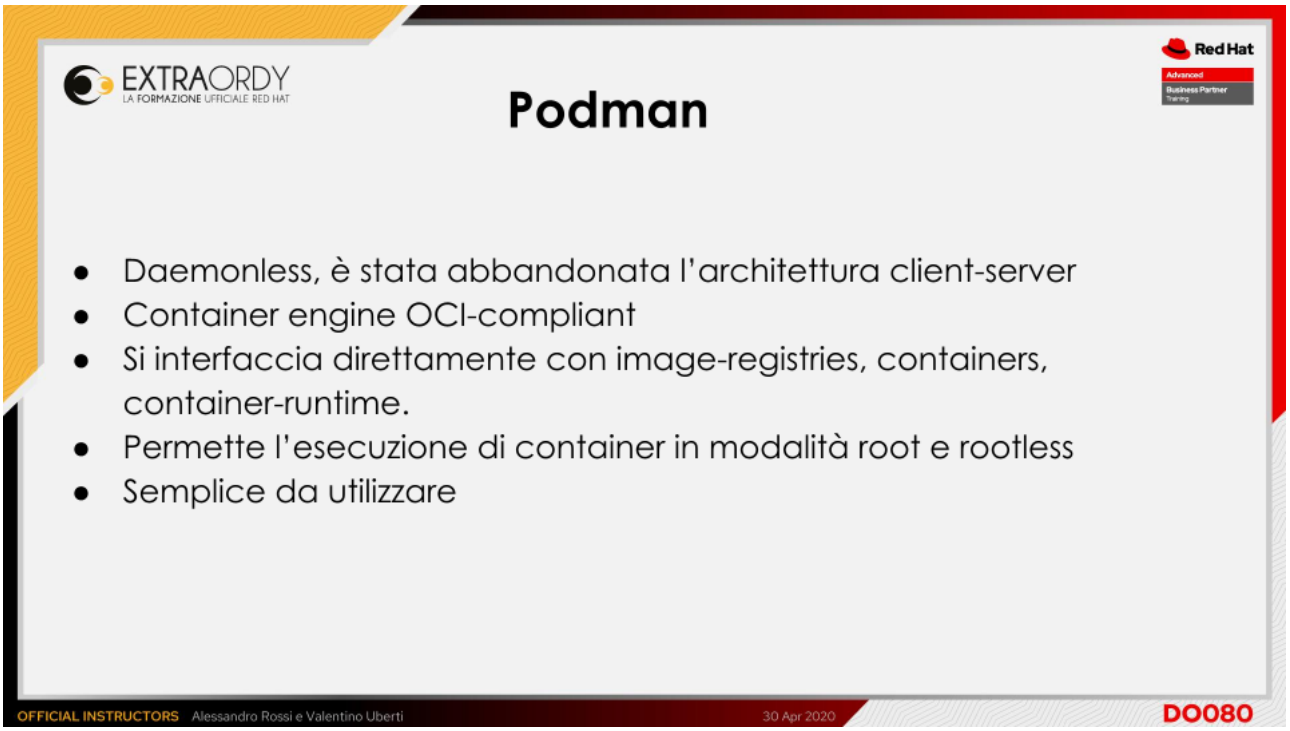
Con OpenShift 4 è stata decisamente migliorata la parte di interfaccia grafica del webconsole ed è diventato uno strumento essenziale e molto comodo per trovare senza problemi le impostazioni di base.

Un'altra feature molto utile è quella della gestione del del traffico esterna all'interno del cluster e le rotte che vengono messe a disposizione da OpenShift
E' l'orchestratore che si fa carico di andare a dirigere il traffico applicativo verso il Pod più scarico e si va a ristabilire la catena: ciò è possibile perché esiste un'entità, il service, che va a dirigere il traffico verso gli altri Pod.

INTRODUZIONE A PODMAN

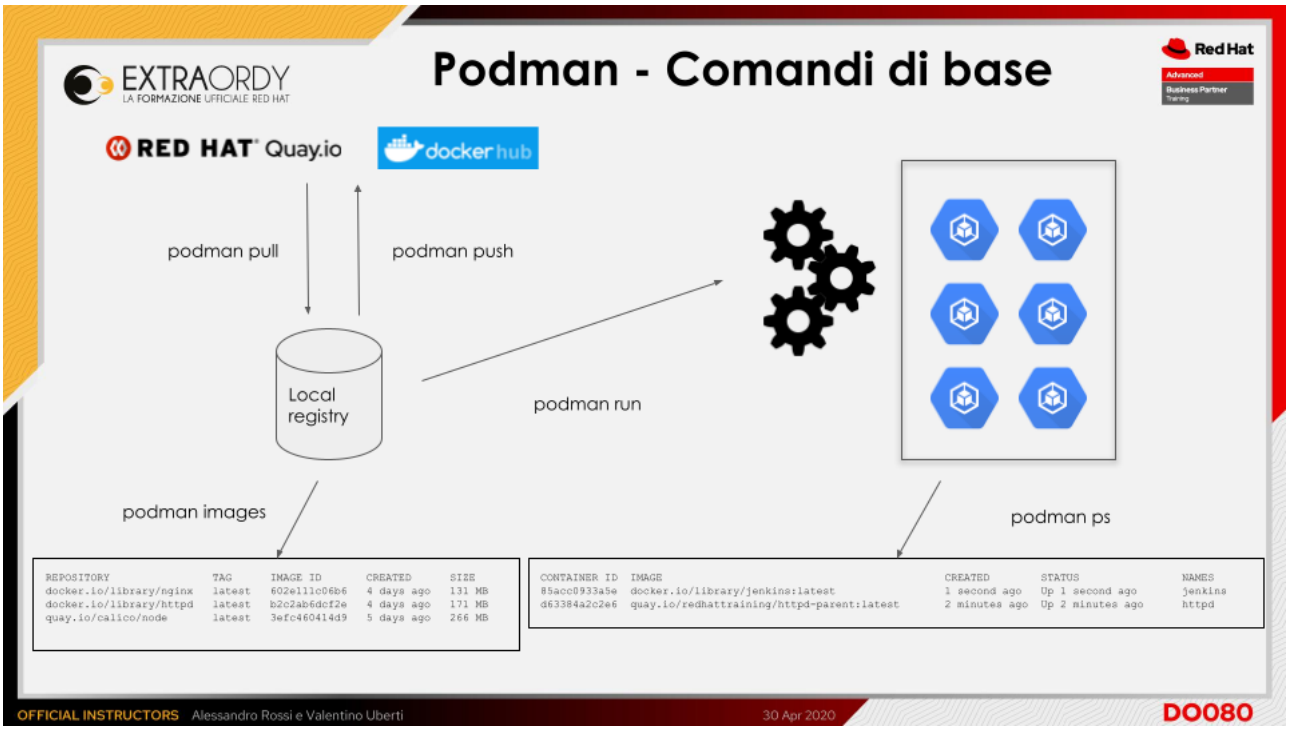
Si andrà a vedere quali sono le sue caratteristiche, quali sono i problemi che risolve e come si può interagire con dei semplici comandi per andare a creare il container.

Podman è un Container engine OCI-compliant cioè è in grado di eseguire le linee guida che vengono stabilite per quanto riguarda la parte di creazioni di Container.



Podman è daemonless e quindi è un'applicazione stand alone che si interfaccia direttamente con image-registry, il container-runtime e permette anche l'esecuzione di container in modalità root e rootless.

Con Podman, dopo aver cercato le immagini che si desiderano, si può recuperare dal registry.



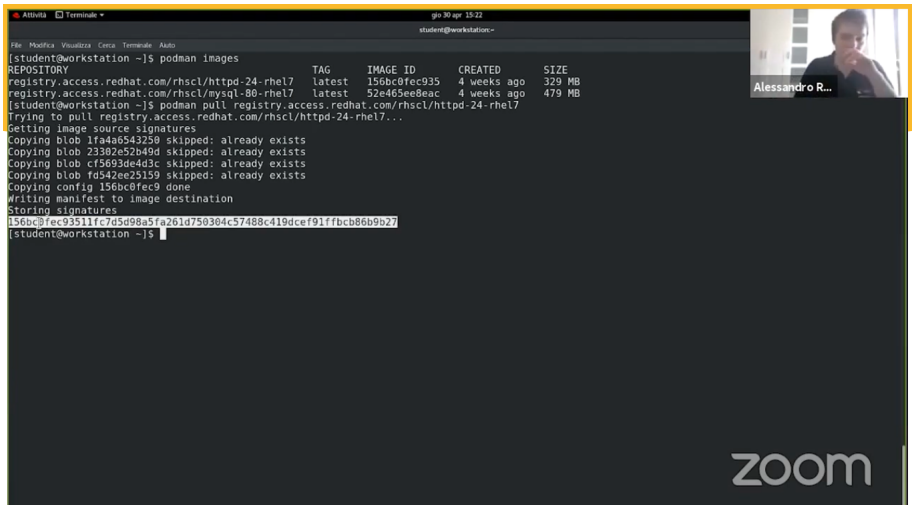
Una volta salvate le immagini, verranno salvate all'interno del quello che è chiamato il local registry, ovvero una directory dove vengono salvati quelli che sono tutti i layer delle immagini scaricate.

L'immagine non è altro che una compressione e un'unione di più layer che permettono anche di andare a velocizzare poi quelli che sono le operazioni di pull.

Si potranno quindi azionare i container che saranno poi visibili esattamente come dei processi e con il comando podman push si possono andare a vedere quali sono i container che stanno girando in un determinato momento.

Come creare un container a partire da un'immagine?

Se si utilizza un podman pull e si indica l'immagine, lui andrebbe a connettersi al registry e andare a ricercare i layer salvando le informazioni dentro un id



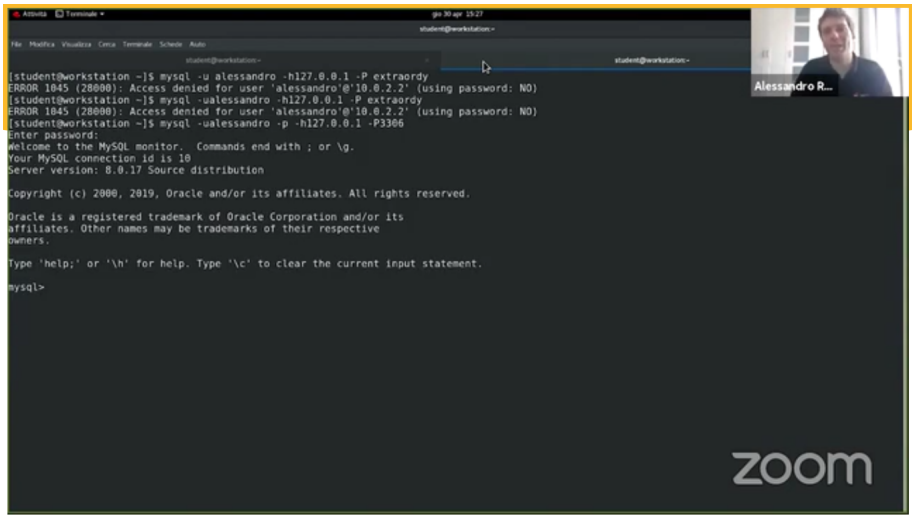
Se si volesse andar ad eseguire l'immagine, invece, si utilizzerà il comando podman run che permette di andare ad eseguire i container in due modalità:

- interattiva
- daemonized

Come creare un piccolo container e farlo girare in modalità background?

Se si andasse a eseguire il contenere in questa modalità esso terrebbe "appeso" l'input del container.

Il container ha necessità di essere "trattato" in un determinato modo perché per funzionare ha bisogno di alcune informazioni: un utente, una password e un nome del db che si deve andare ad eseguire.



Una volta all'interno del container si chiude la connessione se si è abbastanza sicuri che il container "giri".

Si può dare un nome diverso al container con l'opzione "--name".

Se si volesse interrompere il container, si può utilizzare il comando "podman stop" e farlo in diversi modi (prendendo l'id ad esempio)

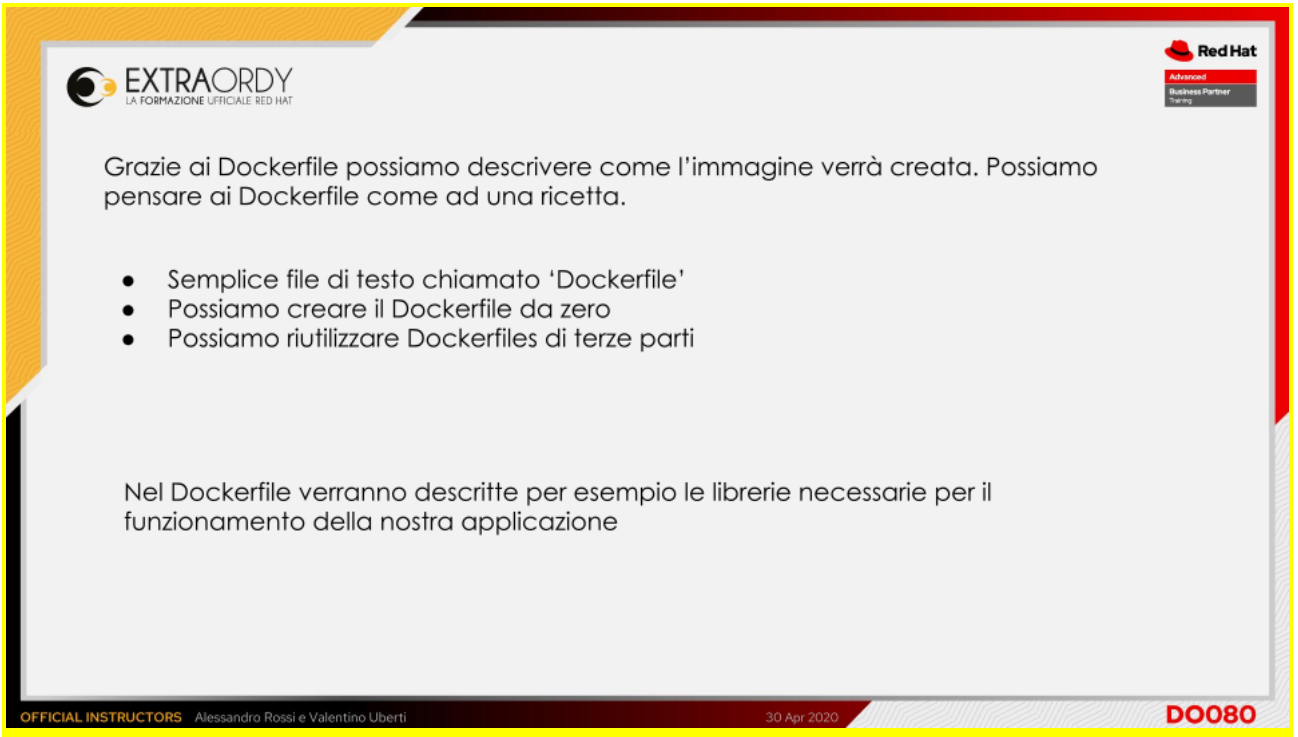
Con l'opzione podman -a si possono andare a vedere anche i pod che sono stati "uccisi" o stoppati nelle ultime esecuzioni. Esiste anche un podman kill.

CREAZIONE DELLE IMMAGINI

Ma come si possono andare a creare le immagini? Come possiamo inserire la nostra applicazione, il nostro sito web e il nostro codice sorgente in un'immagine per poi finalmente creare il container da quest'immagine? **Ci sono in realtà due metodi ma quello più utilizzato è grazie ai Dockerfile.**

Perché è più utilizzato? Perché è semplice. Si tratta banalmente di creare un file di testo con delle direttive.

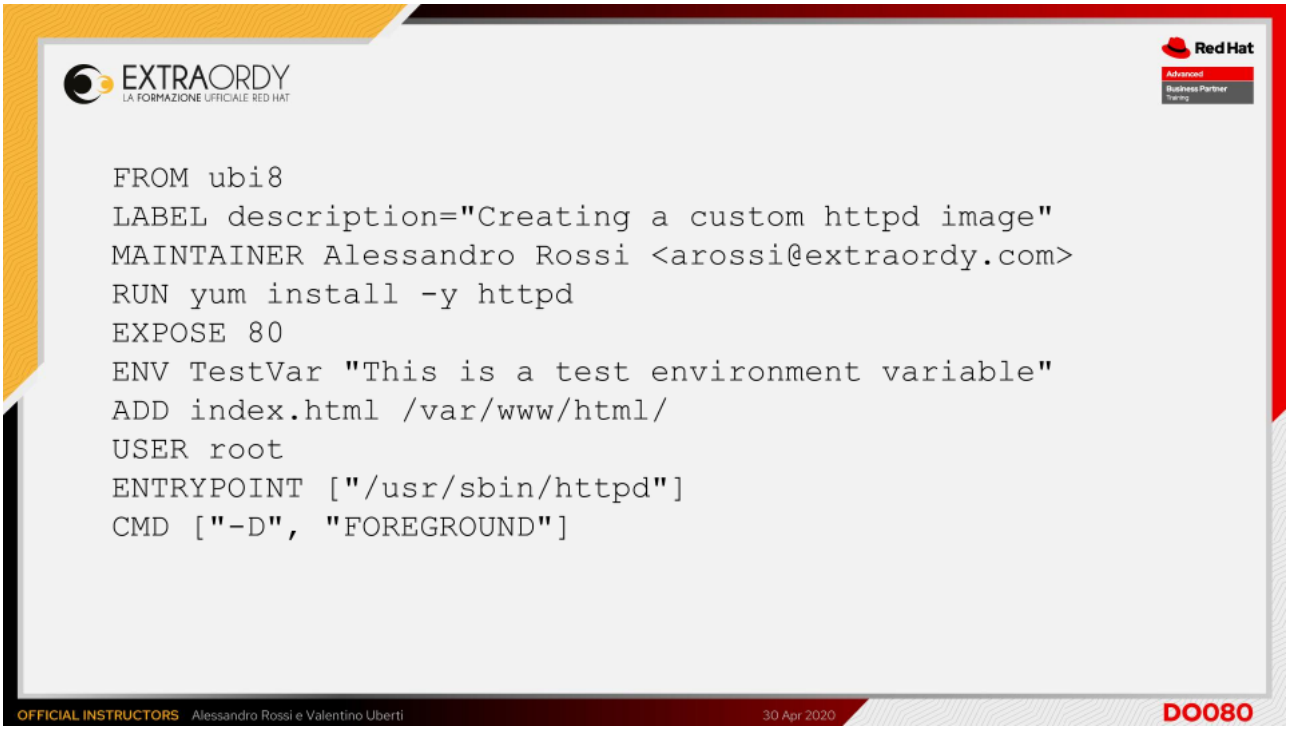
Essendo un semplice file di testo è possibile scambiarselo; ecco perché ha avuto un grosso successo ed è a tutti gli effetti attualmente uno standard de facto per andare a creare le immagini con una utility come Podman.



Per andare ad utilizzare un Dockerfile (e quindi andare a creare una nuova immagine) ci sono due strade; o lo si crea davvero o, come spesso avviene, si riutilizzano Dockerfile creati da terze parti, modificando solo le parti che ci interessano. Grazie ai Dockerfile si possono andare ad inserire le librerie necessarie all'applicazione.

Si vedono ora alcune delle direttive che si possono utilizzare con i Dockerfile; innanzitutto le direttive sono sempre in maiuscolo e la lettura di un Dockerfile avviene dall'alto verso il basso.

Si può pensare al Dockerfile come se fosse una ricetta.



La prima direttiva è sempre FROM che indica il nome dell'immagine di partenza a cui si vuole inserire l'applicazione o le librerie.

Ubi 8 fa parte di una famiglia di tre immagini create da Red Hat ed è un'immagine base che contiene il root file system, ossia tutte le runtime necessarie per creare un un processo. Ci sono tre diverse immagini ubi;

- ubi 8 che è l'immagine standard
- ubi minimal che è un'immagine ulteriormente ridotta
- ubi-init che permette all'applicazione di utilizzare anche i systemd

La direttiva LABEL che semplicemente aggiunge delle label all'immagine.

La direttiva MAINTAINER che non è obbligatoria ma è una best practice per sapere chi ha effettivamente ha creato questa immagine.

Con la direttiva RUN si incominciano a installare i pacchetti necessari per l'applicazione; grazie all'istruzione RUN si possono lanciare dei comandi Linux e il comando Linux per installare un pacchetto è yum. L'opzione -y è mandatoria perché questo questo processo di installazione deve essere automatico. Segue poi il pacchetto che vogliamo installare.

L'istruzione EXPOSE ci indica che il processo container che verrà creato partendo da questa immagine starà in ascolto alla porta 80.

L'istruzione ENV che permette di creare delle variabili di ambiente che possono essere utilizzati dalla nostra applicazione; in questo caso verrà creata una variabile d'ambiente chiamata testVar che sarà valorizzata con la stringa "this is a test environment variable".

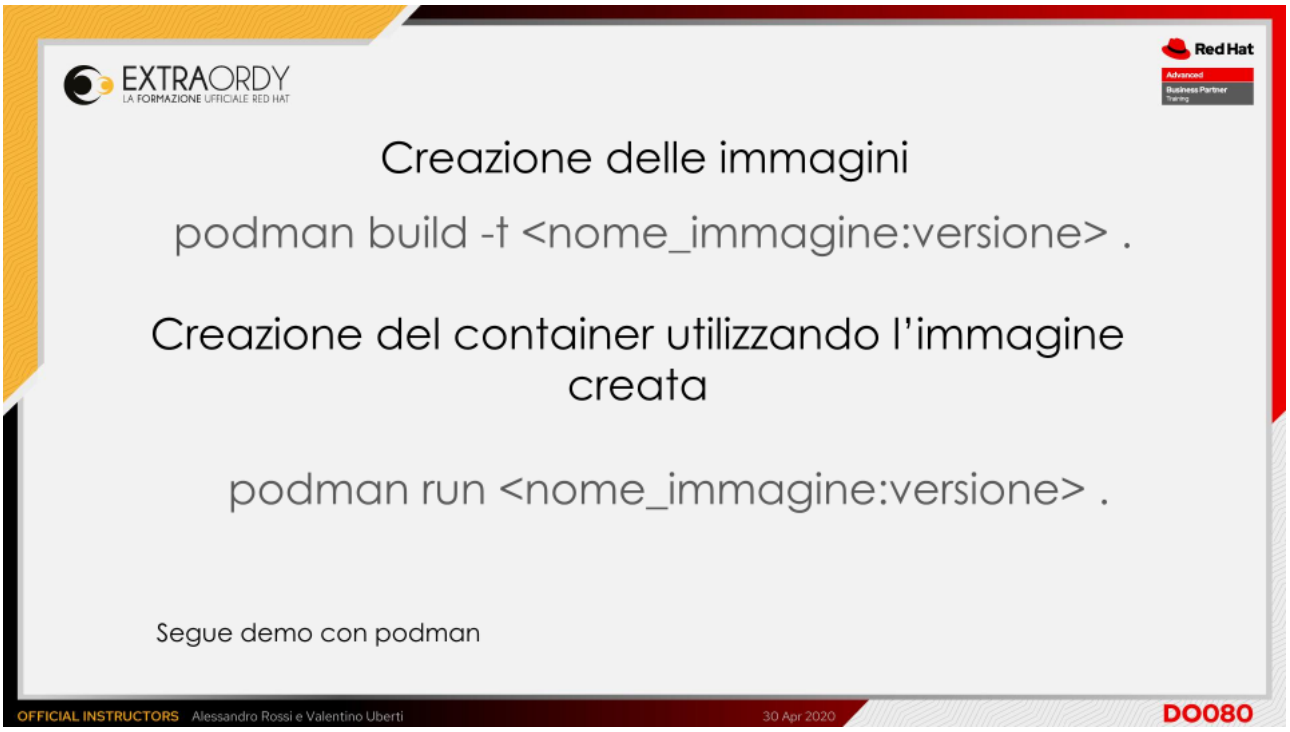
Per copiare i file dell'applicazione all'interno di questa immagine si hanno due istruzioni; add e copy. In entrambe le istruzioni il primo argomento index.html è un file che è presente sulla working directory e da dove poi si lancerà il comando per creare l'immagine. Il secondo argomento /var/www/html/ è la directory in cui si vuole andare a copiare il file index.html

Dopo aver dato l'utenza root alla porta 80 e finire con l' ENTRYPOINT e le opzioni che si vogliono passare all'ENTRYPOINT - ovvero quando si va a creare un processo container partendo da questa immagine, qual è il comando che deve essere lanciato? -

Con quali opzioni si deve lanciare quel comando, invece, ce lo dice la direttiva CMD.
Se l'ENTRYPOINT non viene esplicitato, di default viene lanciata la shell sh con l'opzione -C ovvero accetterà tutti i comandi presenti in CMD.

Quindi si crea questo Dockerfile sulla work station e si può andare a creare prima un' immagine da cui creare il processo container. Se si carica l'immagine su un registro e un'altra società scarica l'immagine da questo registro, può metterla in running e si troverà con un server web che risponderà e che metterà a disposizione il contenuto di questo file index.html.

Ma come si fa a creare un'immagine partendo dal Dockerfile?



Si ha il verbo build; podman build e con il flag -t dove si dà un nome all'immagine. Il nome deve essere composto da un nome alla nostra scelta e, se si desidera, anche dalla versione dall'immagine. Se non si va a mettere il nome della versione verrà assegnata una versione di default. L'ultimo punto (.) indica la directory corrente: questo per dire a podman dove andare a ricercare il Dockerfile. Quando l'immagine sarà pronta si potrà andare a creare un processo container partendo dall'immagine, quindi con tutti i file e con tutte le librerie che si sono voluti inserire.

Quindi ricapitolando alcuni dei vantaggi dell'utilizzo dei Dockerfile per creare immagini si è visto che:

- **è un semplice file di testo quindi è facilmente distribuibile;**
- **le istruzioni complete di un Dockerfile non sono poi tantissime;**
- **è un po uno standard de facto per la per la creazione di immagini - parlando di solo podman.**

RED HAT OPENSHIFT

OpenShift è il prodotto paas - platform as a service- creato da Red Hat ed è il prodotto principe degli orchestrator.

Ma cos'è OpenShift?

OpenShift è una distribuzione di Kubernetes; si basa su Kubernetes e aggiunge molte molte funzionalità. All'inizio molto spesso si diceva che OpenShift renda umano l'utilizzo di Kubernetes e non è poi così tanto falso; la cosa importante da tenere a mente è che se si ha già confidenza con Kubernetes, con OpenShift ci si ritrovano le stesse funzionalità e anche molte di più.

Di sotto sono illustrate alcune di queste di queste novità.

Partendo dal nuovo sistema operativo creato da Red Hat che è CoreOS e la base dei modi in cui verrà installato OpenShift; questo perché è un sistema operativo immutabile. Grazie a CoreOs, non è neanche più richiesto e attualmente è sconsigliato l'accesso ssh

ai nodi OpenShift perché gli update del sistema operativo vengono gestiti direttamente da funzionalità del nostro cluster.

OpenShift si può installare su quasi la totalità delle infrastrutture.

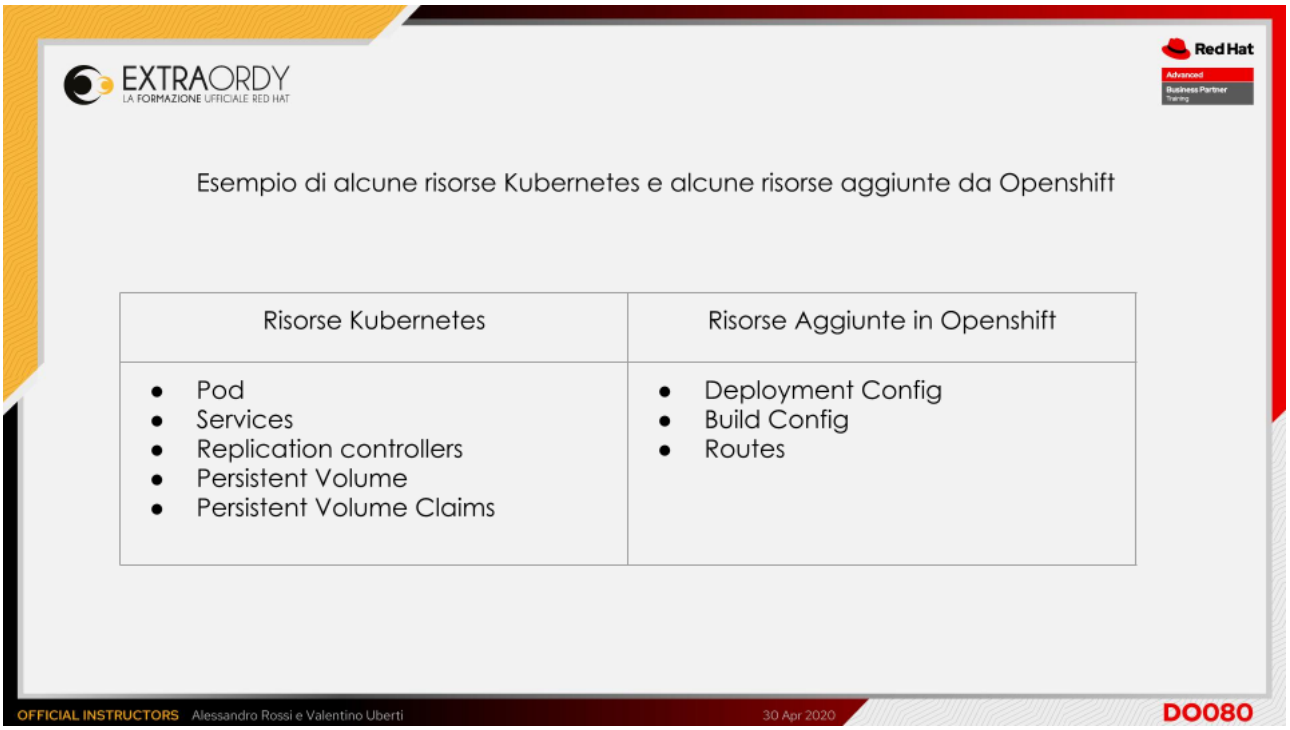
Tra le funzionalità aggiunte da Red Hat OpenShift all'orchestratore di Kubernetes si hanno anche gli automatismi; per esempio si hanno gli operators, che sono una grossa novità di Red Hat OpenShift 4 e che sono dei programmi che girano nei pod e che permettono un monitoring automatico e soprattutto rispondono in maniera automatica ad eventuali problemi delle applicazioni.

Si hanno i Service Mash che non vanno dimenticati tra le funzionalità aggiuntive. Si è ancora compatibili con Jenkins ma è stato introdotto il nuovo formato di pipeline che è Tekton.
In fase di installazione si ha una funzionalità molto comoda perché le metriche risultano già installate.
Service Mash utilizza l'up streaming project Istio che è fortemente integrato con Red Hat OpenShift e si ha tutta una serie di template per creare le applicazioni con pochi click.

E' molto orientato anche per tutta la parte Dev.

Insomma grazie a queste funzionalità aggiuntive **OpenShift** si colloca come strumento principe nella filiera del **DevOps** tanto che nel 2019 era dato al 44 per cento di market share come applicazione paas utilizzata.

Si vedranno ora alcune risorse di Kubernetes e alcune risorse che vengono aggiunte in OpenShift.



In Kubernetes un pod è l'insieme di uno o più container a cui vengono aggiunti dei metadata, ossia delle informazioni. Questi container all'interno del pod, per esempio, utilizzano lo stesso network namespace, che significa che i container presenti nel singolo pod possono comunicare tra di loro riferendosi al local host e soprattutto, siccome il pod è la risorsa che viene schedulata nei nostri nodi, se Kubernetes decide di schedulare un pod in un nodo, siccome il pod viene schedulato su quel nodo, i container al suo interno saranno entrambi schedulati in quel nodo. Questa soluzione è molto utile: per esempio Istio, con il suo proxy Envoy che viene inserito nello stesso pod,a tutti gli effetti aggiunge un container al pod applicativo.

Poi si ha la risorsa services che di norma viene chiamato cluster ip visibile solo all'interno dei nostri cluster e in load balancing va a comunicare con i nostri pod.

Si ha, per esempio, le risorse di replication controllers che effettivamente si occupano di garantire il numero di repliche richiesto perché lo stesso pod, se l'applicazione al suo interno lo permette, può essere scalato orizzontalmente ovvero possiamo avere n. repliche del nostro pod.

Quando si parla di OpenShift, si dice sempre che i processi container sono effimeri, ovvero che i dati che vengono scritti all'interno dell'applicazione vengono persi; questo è verissimo sia con OpenShift e anche Kubernetes, perché, per esempio, quando si ha un pod applicativo che viene riavviato,lì si andrà a perdere i dati. Allora per dare la persistenza dei dei dati alle applicazioni si ha il persistent volume che premette a OpenShift di offrire dello storage, utilizzando delle varie tecnologie. Le richieste di storage dalle applicazioni avvengono grazie a una risorsa che si chiama persistent volume claim.

Tra le risorse aggiunte in OpenShift, una molto importante è il Deployment Config che è la risorsa e dove, per esempio, si va a definire quante repliche si vogliono dell'applicazione, le variabili di environment che si vogliono inserire nell'applicazione e che permette di creare le richieste del persistent volume claim; quindi tutto quello che serve per creare un pod applicativo, partendo da un'immagine già fatta, lo si trova nella Deployment Config.

Un'altra risorsa che esiste solo in OpenShift è la Build Config che ha tutte le informazioni per creare un'immagine; per esempio nella Build Config troviamo la url git da cui si andrà a prendere il codice sorgente.
OpenShift può essere usato per creare delle immagini e mettere in running delle immagini direttamente partendo dal Dockerfile, per esempio, che è una cosa utilissima.

Poi si hanno le routes che servono per permettere la raggiungibilità dell'applicazione dall'esterno del cluster e funzionano solo con un protocollo http, https, ...

IL CLUSTER OPENSHIFT

Come si presenta un cluster OpenShift?

Un cluster OpenShift si presenta con con dei nodi che possono essere virtualizzati o fisici e ogni nodo ha un suo ruolo.

Tipicamente quando si va a installare OpenShift si hanno come minimo tre nodi chiamati master: la loro funzionalità è, per esempio, quella di gestire l'autenticazione. Si ha un servizio fondamentale che è il database chiave-valore chiamato etcd e lo scheduler che, in autonomia o meno, a seconda della configurazione, decide dove mettere in running i dei pod applicativi ed, infine, il management delle repliche. Mandatorio è che i nodi master debbano girare su Red Hat CoreOS, il sistema operativo immutabile.

E' consigliato utilizzare Red Hat CoreOS anche sui nodi chiamati "worker" che sono i nodi dove effettivamente vengono messi in running i nostri pod applicativi.
Si hanno anche i nodi chiamati "infra" dove si può decidere di far girare, per esempio, il router di ingresso, tutti i servizi di metriche e altro.

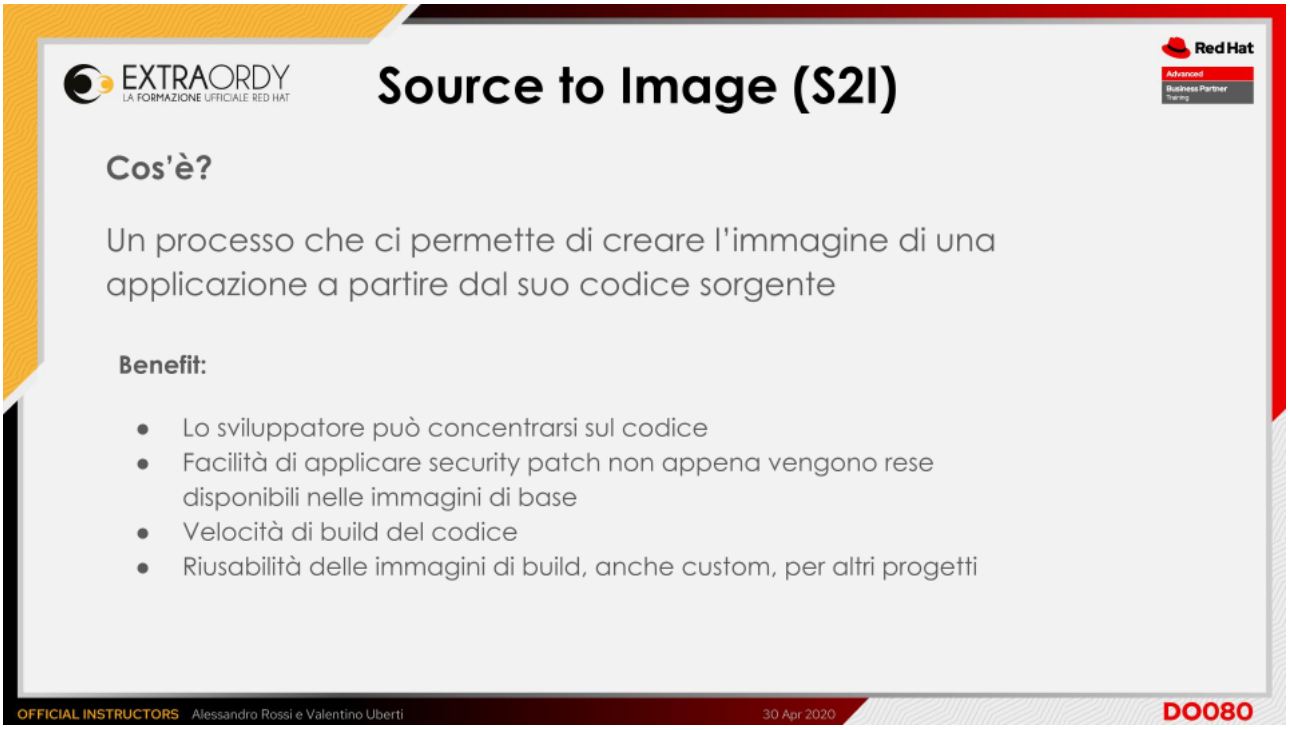
Gli utenti cosa fanno?

Si ha una utility che si chiama OC che permette dalla linea di comando di interfacciarsi col cluster o si ha la web console. Quindi, riassumendo, si ha la possibilità di utilizzare o la web console o la command line e, se si desidera è possibile andare a utilizzare API restfull.

CREARE APPLICAZIONI CON IL PROCESSO "SOURCE TO IMAGE"

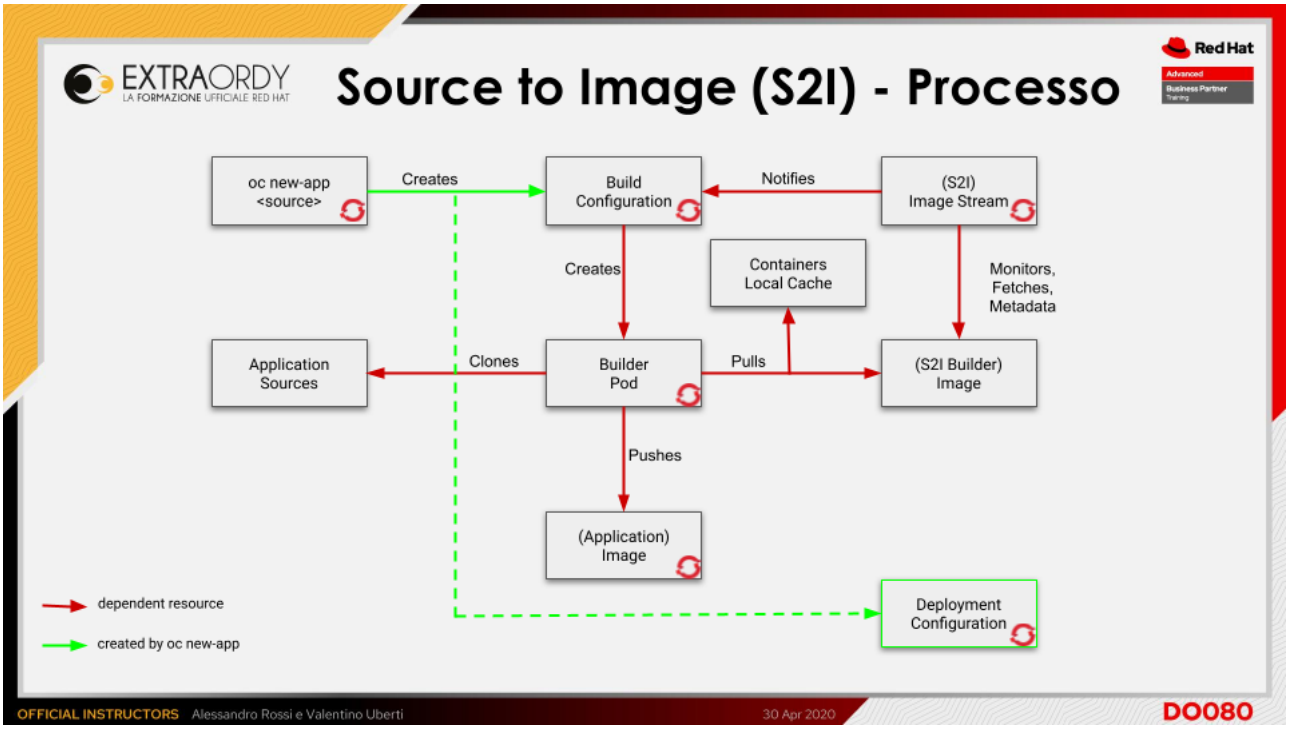
Il processo che si occupa di andare a gestire le applicazioni è processo che si chiama "source to image", ovvero una delle caratteristiche di Open Shift e permette, a partire da una base di codici o da un'immagine già esistente, di andare a compilare l'applicazione e deployarla all'interno di un pod come container. Ciò permette di andare ad uniformare quello che il processo di build e utilizzare delle immagini di build che sono sempre quelle decise dall'utente

Una volta conosciuta o creata quella che è l'immagine di build, lo sviluppatore potrà concentrarsi sul codice. Verranno introdotti degli strumenti, primi tra tutti gli imagine stream, che permettono di andare a gestire un processo di update in maniera molto rapida e in alcuni casi anche in maniera automatizzata.



Si potrebbe fare in modo di scaturire delle build al cambiamento dell'immagine. Nel momento in cui si verifica che c'è stato un cambiamento all'interno dell'immagine, si può andare a reperirla aggiornata. La velocità è diversa e l'utilizzo di risorse e la velocità di utilizzo di queste risorse è differente.

L'immagine di build può essere utilizzata anche in altri progetti all'interno della piattaforma è quindi una configurazione di build distribuita e utilizzabile da tutti.



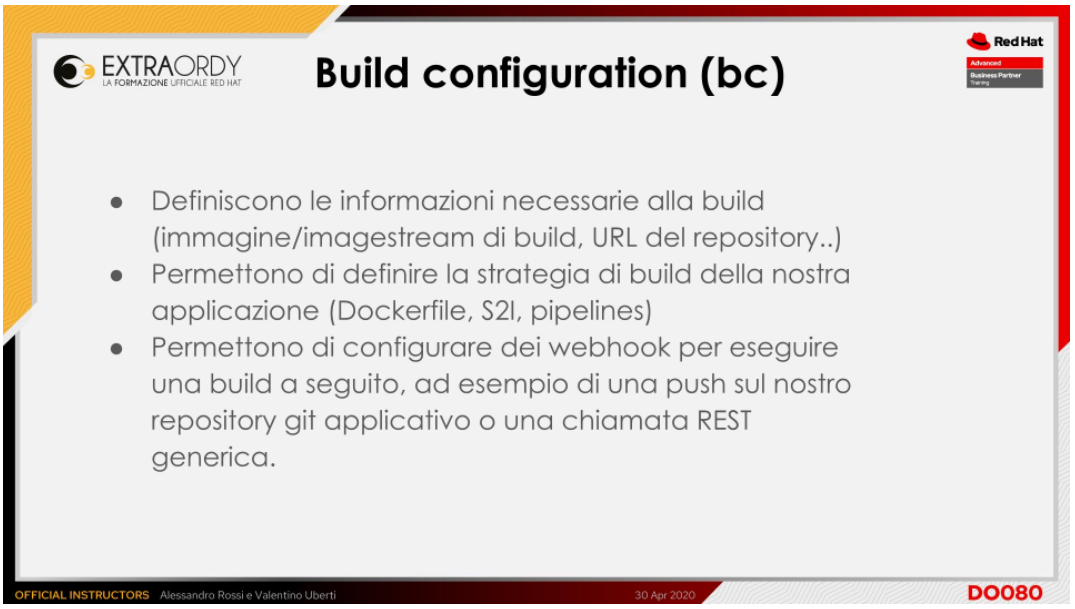
Oc new-app crea una risorsa, la build configuration va a creare alcune informazioni che occorrono alla piattaforma per andare a guidare l'immagine. Ad esempio, andrà ad impostare alcuni valori all'interno della build configuration cioè tutte le informazioni che possono servire durante il processo di build.

Viene poi creato un pod builder questo su cui viene fatta girare l'immagine.
Viene creato e clonato il repository, viene reperita l'informazione. Il codice prenderà l'informazione e andrà a reperire quelli che sono i requisiti delle immagini di build e va ad assemblare tutte le componenti per creare l'immagine.

Nel momento in cui viene creata l'applicazione viene creata anche un'altra piccola ma utilissima risorsa, l'image streams, che è una sorta di puntatore che mappa delle immagini.

E' il componente che permette di prendere un'immagine e monitorarla permettendo di mantenere le immagini di build sempre aggiornate. Crea anche una deployment configuration che va a definire come dovrà essere deployata l'applicazione.

Le build configuration permettono di definire qual è la strategia di build che si vuole eseguire.



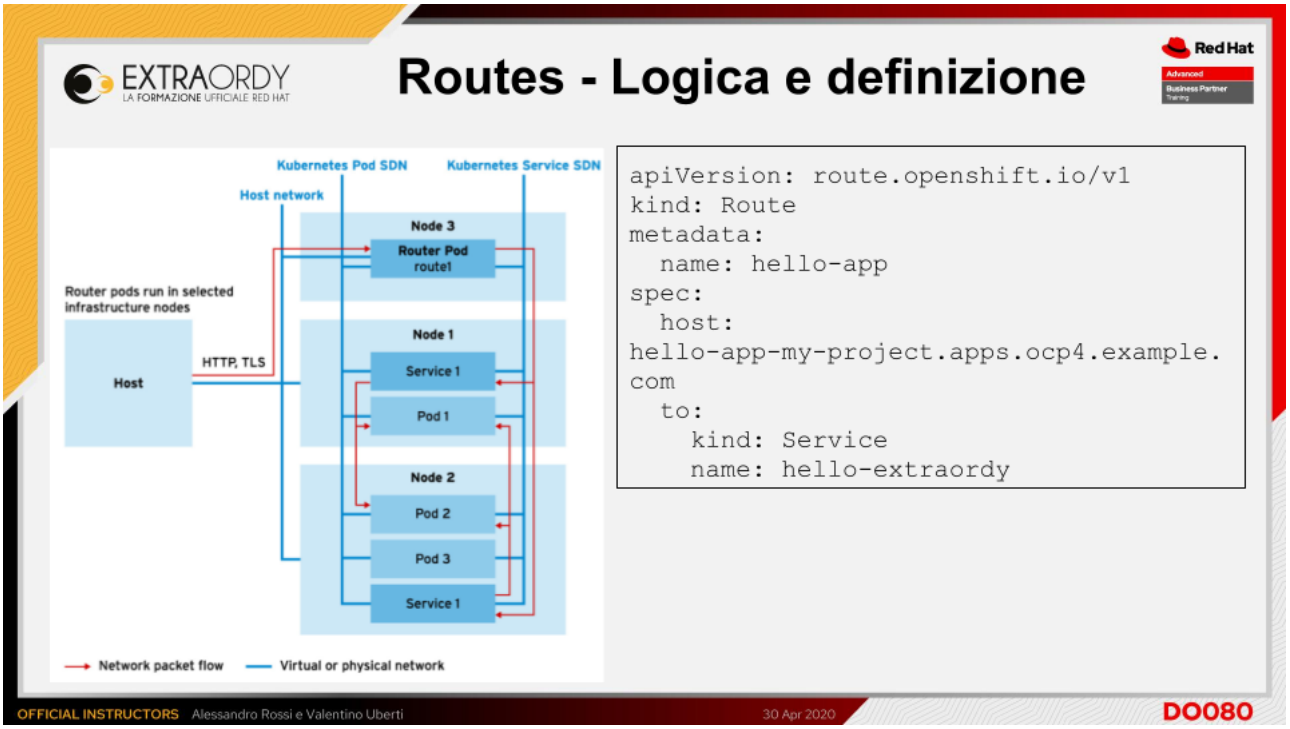
Le deployment configuration si occupano invece della parte di deploy e si può quindi specificare una strategia di deploy, rollout e roll back.

COME INSTRADARE TRAFFICO ESTERNO - ROUTES

La route permette di instradare traffico esterno al cluster verso i pod e permette di "pubblicare" il record dns relativo all'applicazione ed essere risolvibile grazie al pod del router.

Si può personalizzare l'hostname a cui sarà disponibile l'applicazione. Sono deputati a gestire il traffico layer 7.

I pod intercettano la richiesta, verificano attraverso il discovery services a chi è assegnata la risorsa.



Internamente, nella parte di orchestrazione, il router saprà esattamente che il service risponde all'hostname, e questo è l'unico modo che ha OpenShift per capire dove mandare la richiesta. Internamente c'è una rete una sdn per i service.

Il service è collegato a più pod e conosce tutti pod dell'applicazione; la richiesta viene poi smistata sul pod in base al tipo di bilanciamento impostato sul service.

La rotta quindi istruisce il router su dove mandare la richiesta e su quale pod inviarla.

WEB CONSOLE

La web console è stata completamente rinnovata in OpenShift 4 e ha una sezione dedicata più agli sviluppatori e una sezione amministrativa.

Cliccando sul pod si ha un elenco di risorse utilizzate per l'applicazione; il servizio; la rotta già creata e il pod stesso. Nel menu, "overview", è un'applicazione che permette lo scaling orizzontale e permette di scalare l'applicazione ovvero verrà creato un altro pod.

CONCLUSIONI

In piena ottica di condivisione, è stato ritenuto utile fornire una overview di quelle che sono le sostanziali differenze tra l'adozione di una piattaforma completa come Red Hat Openshift Container Platform, e l'utilizzo di un servizio di provisioning di un cluster Kubernetes all'interno dei maggiori cloud provider al momento disponibili sul mercato.

In fase di valutazione delle soluzioni disponibili, è sempre consigliato approfondire tutti gli aspetti implementativi, rivolgendosi a personale qualificato in grado di poter identificare, evidenziare ed indirizzare quelle che sono le necessità di business e come trovare la soluzione idonea a realizzarle.

Chiarezza, competenza e concretezza devono essere le fondamenta sul quale costruire la propria infrastruttura, evitando di incorrere in sorprese o adottare soluzioni che di fatto non siano coerenti con quelle che sono le nostre aspettative, seppur superficialmente più attraenti, magari a livello di pricing e di promozione.

EXTRAORDY - LA FORMAZIONE UFFICIALE RED HAT

Via Gaio 4,
20129 Milano

training@extraordy.com
www.extraordy.com