Contenidors OCI: tecnologies i eines

A continuació descriurem els conceptes i tecnologies més importants relacionats amb els contenidors que són de tipus "aplicació", i més en concret, amb els contenidors que segueixen l'estàndard "OCI".

Imatges OCI, "engines" i "runtimes"

Per "contenidor" OCI es pot entendre dues coses que són diferents i que cal distingir:

a) Si no està en execució, un contenidor és un fitxer guardat en disc. Aquest fitxer s'anomena "imatge" i el lloc on estan ubicades les imatges disponibles se sol anomenar "repositori", ja sigui local o remot. Si el repositori és remot, les imatges han hagut d'haver-se descarregat prèviament per tal de poder-se fer servir a partir de llavors com a punt de muntatge en el moment que es posi en marxa el contenidor en sí.

> NOTA: Normalment ens serà suficient utilitzar alguna imatge de tercers que ja estigui publicada perquè n'hi ha moltes i molt diferents (hi ha que només contenen un shell bàsic, n'hi ha que contenen tota una plafaforma Java,.NET,PHP, n'hi ha que contenen un software de servidor concret com PosgreSQL, ELK, etc), així que serà relativament fàcil trobar la que més ens convingui en un moment donat. Exemples de repositoris públics de tercers compatibles amb OCI són el "Docker Hub" (de Docker), el "Quay.io" (de Red Hat), el "ECR" de AWS, el de Google, el d'Azure, etc. No obstant, si volguéssim crear una imatge pròpia, el que se sol fer més sovint és partir d'una imatge base que sigui semblant a allò que voldríem tenir i llavors, a partir d'ella, afegir-hi/treure-hi el que ens falti/sobri (mitjançant l'aplicació de les instruccions adients, indicades dins d'un arxiu anomenat "Containerfile"). De fet, aquesta és la manera estàndard de crear imatges per part de tothom. Ho estudiarem.
> NOTA: Tècnicament, una "imatge" està composta de varis fitxers ".tar" que representen una "capa" cadascun i un fitxer "manifest.json" que incorpora les metadades necessàries per poder interpretar aquests fitxers ".tar" correctament. Cada capa diferent és afegida a la imatge com a resultat de l'execució d'una instrucció concreta indicada al fitxer "Containerfile" utilitzat per generar la imatge final, a mode de "ceba". Aquesta estructura interna de les imatges fa que la seva publicació no sigui tan senzilla com simplement allotjar un simple fitxer en un simple servidor-repositori ja que aquest reconegui quines capes conté cada imatge per tal d'evitar duplicitats, permetre la descàrrega de capes soltes, etc (veure nota següent)
> NOTA: El document "Distribution Specification"(https://github.com/opencontainers/distribution-spec/blob/main/spec.md) descriu com distribuir imatges a través de servidors-repositori OCI propis o públics de tercers (els quals, al final, no són més que servidors HTTPS amb certes restriccions a l'API que ofereixen). En altres paraules: aquest document estableix el format de la comunicació que ha d'haver entre el "CE" (veure punt següent) i el repositori en qüestió per tal de "pujar-ne" o "baixar-ne" imatges senceres i/o capes individuals (entre altres accions). Aquesta estandardització permet poder disposar d'imatges allotjades en diferents repositoris (com els mencionats a la primera nota anterior) indistintament, sempre que aquests respectin aquest document.

b) Quan l'usuari executa la comanda que inicia el contenidor, el "Container Engine" (a partir d'ara, "CE") desempaqueta la imatge com si fos un "zip" i proporciona el seu interior (un conjunt de fitxers i metadades) al kernel de la forma adient per a què aquest l'executi com si fos un procés normal més (de fet, es fan servir les mateixes crides al sistema per executar un contenidor que per executar un procés "clàssic", típicament *clone()* en comptes de *fork()* o *exec()*).

> NOTA: En el procés d'execució del contenidor també s'inclouen, això sí, crides al sistema addicionals que proporcionen un aïllament addicional fent ús de "cgroups", "namespaces" i SELinux/AppArmor+capabilities +seccomp, etc

Per tant, podem definir un contenidor, ara sí, com la instanciació en RAM d'una determinada imatge guardada en disc. Es poden executar múltiples contenidors a la vegada a partir d'una sola imatge però, en tot cas, cal tenir en compte que per defecte tots els canvis interns que pateixi cada contenidor durant la seva execució no es veuen reflectits pas a la imatge original degut a què un cop iniciat, cada contenidor passa a ser una entitat separada i independent de la imatge de la qual sorgeix.

Tant el format en disc de la imatge (punt a)) com el procediment d'iniciar un contenidor (punt b)) actualment estan definits per sengles estàndards promoguts per l'OCI ("Open Container Initiative", https://www.opencontainers.org, un conglomerat d'empreses i organitzacions diverses que vetllen per la compatibilitat entre diferents eines tecnològiques dins de l'ecosistema dels contenidors d'aplicacions). Concretament, la "Image Specification" (https://github.com/opencontainers/image-spec) detalla el format que ha de tenir en disc una "imatge" compatible amb OCI (tant de les capes que la composen com de les metadades

que la defineixen) i la "Runtime Specification" (https://github.com/opencontainers/runtime-spec) detalla el procediment de com s'ha d'executar el contingut d'una imatge un cop desempaquetada. La gràcia de respectar l'estàndar OCI és que diferents projectes independents entre sí, poden generar imatges compatibles i desenvolupar eines relacionades (per editar-les, signar-les, executar-les, moure-les, inspeccionar-les, etc) d'una forma interoperable entre plataformes, proveïdors i arquitectures.

Existeixen diferents "CE" compatibles amb OCI; tots ells fan la mateixa funció bàsica ("agafar" una imatge -ja sigui descarregant-la en aquell moment o obtenint-la d'un disc local- i convertir-la en un contenidor -és a dir, en un procés- seguint l'especificació OCI) però cadascun té les seves particularitats. Podem destacar, a més dels ja mencionats **Docker** i **Podman**, **CRI-O** (https://cri-o.io), que és un "CE" -en forma de dimoni- utilitzat per defecte en els clústers Kubernetes.

> NOTA: Altres "CE" interessants són **Kata** (https://katacontainers.io) -el qual també permet executar màquines virtuals a més de contenidors- i **Containerd** (https://containerd.io), entre d'altres. D'altra banda, un projecte que pot funcionar per sobre d'aquests dos és el "metaCE" **Firecracker** (https://firecracker-microvm.github.io)
> NOTA: Kubernetes implementa l'anomenada "Container Runtime Interface" (**CRI**), la qual és una API estàndard que permet a Kubernetes que diferents "CEs" compatibles puguin intercanviar-se i així poder treballar amb qualsevol d'ells indistintament. A més del propi CRI-O, l'altre "CE" compatible amb CRI més important actualment és Docker. Podman, en canvi, no és compatible amb CRI perquè el seu "target" és diferent
> NOTA: Tant Podman com Kubernetes utilitzen l'anomenada "Container Network Interface" (**CNI**), la qual és tant una API estàndar (https://github.com/containernetworking/cni/blob/master/SPEC.md) com un conjunt de llibreries que les implementen, les quals permeten definir "plugins" especialment dissenyats per configurar tarjes de xarxa de diferents formes (en mode "pont", en mode "nat", etc). D'aquesta manera, un contenidor pot tenir la seva configuració de xarxa definida a partir del "plugin" que en aquest moment tingui carregat i canviar-la per una altra si canvia de "plugin".

Tots els "CE" anteriors es basen en **runc** (https://github.com/opencontainers/runc), que és el model de "Container Runtime" (a partir d'ara, "CR") proporcionat per la mateixa OCI com a implementació de referència de la seva pròpia especificació. Això fa que l'execució final dels contenidors es faci d'una forma homogènea, sigui quin sigui el "CE" emprat.

> NOTA: Altres "runtimes" importants compatibles amb la "Runtime Specification" (però alternatius a "runc") són **crun** (https://github.com/containers/crun) o **gVisor** (https://github.com/google/gvisor).

La diferència entre el que és un "CE" i el que és un "CR" es subtil però cal tenir-la en compte. Qui realment interpreta el contingut de la imatge i *executa* el contenidor (realitzant les crides al kernel necessàries, a més d'altres tecnologies com "cgroups", "namespaces" i SELinux+capabilities+seccomp, etc) és el "CR". El "CE" (el qual funciona "per sobre" del "CR", "embolcallant-lo") fa tasques addicionals però no menys importants com ara gestionar la interacció amb els usuaris (via paràmetres del terminal i/o via peticions REST des d'un orquestrador i/o via peticions gRPC -com ho fa CRI-O-, etc), connectar amb els repositoris configurats i descarregar les imatges sol.licitades, descomprimir-les i preparar el punt de muntatge d'entrada pel contenidor, cridar al "CR" quan calgui efectivament posar en marxa el contenidor (la imatge del qual haurà d'estar ja muntada), informant en aquest pas al "CR" de les totes metadades que aquest necessiti (normalment a través de la creació per part del "CE" d'un arxiu "config.json" que serà llegit pel "CR") per a què pugui reconèixer el punt de muntatge correctament, realitzar les crides adients, etc.

Orquestradors

És molt habitual voler connectar diferents contenidors entre sí per tal d'executar-los com una unitat (anomenada "pod"). Per exemple, podem tenir un contenidor executant Apache, un altre executant PHP-FPM i un altre MySQL. Per tal de connectar entre sí aquests tres contenidors i així fer que treballin en equip, compartint si cal espais d'emmagatzematge, variables d'entorn comunes, etc cal una eina que, de forma centralitzada, els pugui gestionar (iniciant-los i parant-los quan calgui, monitoritzant el seu estat, etc).

> NOTA: Un "pod" pot estar compost per un sol contenidor o per uns pocs íntimament relacionats entre sí

Si es vol tenir aquest conjunt (petit) de contenidors interrelacionats funcionant sobre una màquina local, les eines més habituals són **Docker-Compose** (per contenidors Docker) o **Podman-Compose** (https://github.com/containers/podman-compose, per contenidors Podman -tot i que aquest CE també

incorpora la comanda pròpia *podman pod* que també ens podria servir-). Aquestes eines disposen d'un únic arxiu de configuració (en format YAML) on s'especifiquen les característiques i interrelacions de tots els contenidors que formen el "pod" i fan molt senzilla la seva administració mitjançant comandes de l'estil *podman-compose up* (per iniciar tots els contenidors del "pod" de cop) o *podman-compose down*, etc

Si es vol anar més enllà, afegint capacitats de desplegament automàtic de "pods" en una màquina (o un clúster d'elles) i d'auto-escalada segons els recursos detectats (és a dir, amb la possibilitat de moure "pods" entre màquines del clúster o de matar-los o/i tornar-los a iniciar segons les circumstàncies, tot de forma desatesa i amb monitorització permanent -tant dels propis contenidors (a nivell de CPU, RAM disc i xarxa) com del funcionament del "CR" i "CE" emprat-) cal utilitzar un "orquestrador". L'orquestrador més utilitzat amb diferència actualment és **Kubernetes** (https://kubernetes.io)

> NOTA: Kubernetes està dissenyat per treballar amb milers de "pods" funcionant sobre un clúster format per vàries màquines. Existeixen "versions" de Kubernetes més "petites" especialitzades en gestionar desenes "pods" dins d'una única màquina física, fent-les ideals en entorns de proves i tests. Exemples d'aquestes "versions petites" de Kubernetes són **K3s** (https://k3s.io), **Minikube** (https://minikube.sigs.k8s.io/docs) o **Minishift** (https://www.okd.io/minishift)
> NOTA: Red Hat ofereix per la seva banda una "distribució" Kubernetes anomenada **OKD** (https://www.okd.io), la qual és la base del seu producte PaaS/SaaS Openshift (https://www.openshift.com) . Una "distribució" Kubernetes consisteix en un conjunt de software addicional que complementa l'orquestrador Kubernetes bàsic (gestor de paquets, panell de control web, etc)