



*I container sono i componenti alla base dei più popolari servizi di condivisione. Come funzionano e come utilizzarli direttamente.*

## Cloud, cloud, sempre più cloud. Come?

Nel numero scorso abbiamo visto come la domanda sempre crescente di servizi software dal “cloud”, cioè su computer virtuali accessibili via Internet, ha portato a un cambiamento profondo delle tecniche di virtualizzazione, su Linux e altrove. Se una volta si usavano quasi esclusivamente delle vere macchine virtuali, cioè modelli software di computer completi, su cui far girare sistemi operativi altrettanto completi, negli ultimi anni hanno preso piede i *container*: scatole software molto più leggere ed efficienti, ottimizzate e preconfigurate per far girare una o pochissime applicazioni. Completiamo il discorso, questo mese, presentando i problemi legati all’uso su larga scala dei container più popolari del momento, i Docker, e alcune loro soluzioni, principalmente ma non solo Open Source.

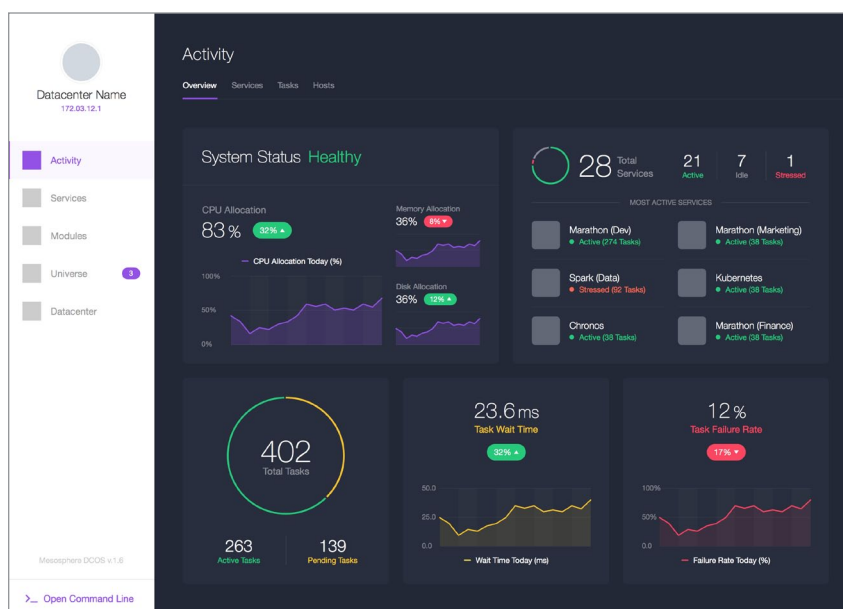
### SI FA PRESTO A DIRE CONTAINER

Il mese scorso abbiamo mostrato come e perché i container sono un sistema di virtualizzazione molto più compatto ed efficiente delle macchine virtuali, quindi ideale anche per i cosiddetti “microservizi” descritti nell’apposito box di questo numero. L’arrivo di Docker ha ulteriormente semplificato non solo, e non tanto, la creazione di container Linux, ma soprattutto il loro riuso e combinazione tramite immagini.

In questo contesto, per “immagine” si intende un pacchetto di bit che è una sorta di fotografia istantanea di un container, con tutte le librerie e file di configurazione che servono alla sua applicazione. Un’immagine può essere utilizzabile così com’è, oppure servire solo come base per altri container. Le immagini Docker sono estremamente convenienti, ma dalla loro esistenza

all’uso in produzione passa la stessa differenza che c’è fra realizzare una automobile perfettamente funzionante, e avere sia una fabbrica, sia centinaia di autosaloni in grado di venderne migliaia di esemplari.

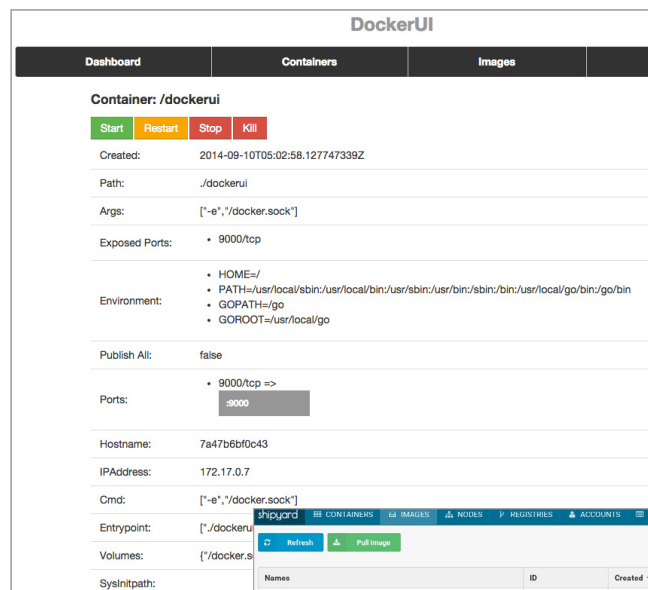
Per usare, oppure offrire, centinaia di container Docker occorre molto di più che il software Docker puro e semplice. Servono anche interfacce per gestire



I container sono software complessi, ma non è difficile avere un’idea di come stanno operando, per poterli utilizzare meglio. La dashboard di Dcos è un cruscotto che fa capire subito se e dove si stanno verificando malfunzionamenti. (fonte: dribbble.com)

# MICROSERVIZI CON DOCKER

Un buon indicatore della validità di Docker è la quantità e varietà di modi in cui viene usato, inclusi quelli non previsti dai suoi stessi creatori. Come esempio di quanto vogliamo dire citiamo i cosiddetti *microservizi*, spiegabili abbastanza facilmente con due software in grado di offrirli, anche se in maniera molto grezza. Dind ("Docker-in-Docker", <https://github.com/jpetazzo/dind>) è un sistema per far girare il gestore di Docker all'interno di un altro container Docker, sfruttando l'esistenza di una modalità di esecuzione "privilegiata" per questa tecnologia. Dind è obsoleto, a detta del suo stesso autore, ma se si conoscono i limiti è ancora utile per "microservizi" come collaudi di container, magari a scopo didattico. La versione di Dind ancora scaricabile, infatti, lancia all'interno di un container già esistente la versione di Docker più recente al momento. In altre parole, due esecuzioni di Dind a pochi minuti di distanza l'una dall'altra, con gli stessi parametri, potrebbero comunque dare risultati molto differenti. Oltre a questo, l'uso di Dind è ufficialmente sconsigliato senza almeno la protezione di un firewall. Il microservizio offribile installando il programma Dockersh (<https://github.com/Yelp/dockersh>) su un server Linux è l'accesso remoto al medesimo a più utenti, ma con molta più sicurezza che utilizzando normali account. Su Linux, infatti, a ogni account normale è associata una *shell*, cioè un interprete di comandi completo (e quindi un possibile punto di attacco, sia ad altri utenti sia al server in generale). Con Dockersh, invece, ogni connessione conduce il suo utente all'interno di un container separato, in cui non può interagire affatto con gli altri né, per esempio, utilizzare tutta la banda disponibile per collegarsi a Internet. A rigor di termini, questa segregazione automatica in container era, ed è ancora, possibile anche prima di Dockersh. Rispetto ad altri approcci quello di Dockersh è però più semplice ed efficiente, anche se non altrettanto collaudato sul campo. Con Dockersh, infatti, un unico daemon Ssh (Secure Shell) può gestire tutti gli utenti da segregare, con la stessa configurazione di default utilizzata con shell normali.



L'ambiente Shipyard mostra quante e quali immagini Docker sono disponibili per creare "swarm" (sciami) di container, classificandole con etichette per facilitarne scoperta e gestione.

Names	ID	Created	Node	Virtual Size
nginx:1.7.1	61e8f64e1d65	2014-07-21 05:22:18 -0400	qjx-1	476.01 MB
shipyard/etcd-kubernetes	3321c25160cd	2014-09-26 08:00:01 -0400	qjx-1	277.60 MB
shipyard/docker-swarm	b6e802f0f952	2014-10-24 12:00:21 -0400	qjx-1	91.84 MB
shipyard/etcd-kubernetes	49866fb1536	2014-12-31 17:23:56 -0500	qjx-1	2.32 MB
redis:latest	834620358f62	2014-12-31 17:39:27 -0500	qjx-1	230.15 MB
gonghi/etcd-kubernetes	0613e66b7c48	2015-01-28 17:35:49 -0500	qjx-1	595.42 MB
shipyard/etcd-kubernetes	dc49c042019b	2015-02-01 00:18:20 -0500	qjx-1	341.53 MB
localstack:2005-swarm	b786923851d	2015-02-25 19:51:12 -0500	qjx-1	6.86 MB
shipyard/shipyard:2.0.10	f1bc70df0e1	2015-03-02 19:50:19 -0500	qjx-1	34.58 MB
etcd-kubernetes	ec0937a388d	2015-03-03 09:19:29 -0500	qjx-1	259.81 MB
shipyard/shipyard	b494b7a20546	2015-03-08 03:00:13 -0400	qjx-1	6.21 MB
alpine:latest	b6b08730b93c	2015-03-17 19:15:51 -0400	qjx-1	4.80 MB

Maneggiare container, almeno per studio e collaudo, non è un'impresa riservata ai guru dello sviluppo. Pannelli di controllo come DockerUI presentano tutte le informazioni e operazioni principali in semplici schermate.

migliaia di account e siti utente, con le relative password, firme digitali e decine di altri parametri. Servono strumenti robusti e automatici per gli aggiornamenti software, o per far ripartire ogni container da solo dopo un guasto, minimizzando le perdite di dati. Oppure per raccogliere e analizzare migliaia di log file, magari comparandoli fra loro. Prima ancora di questo, però, occorrono assistenza e controlli per lanciare ogni container automaticamente, ma sempre nel momento e nel posto giusto della propria infrastruttura. Preferibilmente con la garanzia di poter lavorare nello stesso modo, anche quando la stessa infrastruttura di base è virtuale. Cioè quando si vogliono fare tutte queste cose... sopra una "nuvola" che in realtà è un mix, magari variabile, di data center di diversi fornitori indipendenti. In condizioni del genere, anche il solo decidere su quale server si dovrebbe lanciare un container è un'impresa troppo complessa per gestirla manualmente.

## LO SCIAME DEI CONTAINER

Il primo passo per risolvere quei problemi o se vogliamo, guardandola da un altro punto di vista, per *aggravarli*, si chiama Swarm (in Italiano

"sciame", <https://docs.docker.com/swarm>). Swarm ha infatti introdotto nel mondo Docker il supporto nativo per *cluster*, cioè raggruppamenti, di container. Usando Swarm, un insieme di container Docker altrimenti indipendenti appare dall'esterno come un'unica macchina virtuale, ovviamente con prestazioni e capacità molto superiori a quelle di ogni suo singolo componente. Allo stesso tempo, ogni programma già capace di comunicare con il daemon (il server di controllo) di Docker può lavorare con e nei container "raggruppati", come se fossero uno solo.

Già oggi l'unica operazione diversa dal solito che si deve fare per creare uno sciame è scaricare l'immagine Docker corrispondente, chiamata appunto Docker Swarm, e configurare tutti gli altri componenti del cluster che si sta costruendo a partire da quella. Il modo corretto di farlo a basso livello è usare la cosiddetta docker-machine ([www.docker.com/docker-machine](http://www.docker.com/docker-machine)): un programma da riga di comando fatto apposta per gestire tutte le procedure Docker, dalla creazione di host alla generazione dei relativi certificati per crittografia e firma digitale.

## MESOS

Le interfacce base come docker-machine sono sufficienti per uso professionale quando si ha a che fare con pochi container alla volta. Per configurazioni più complesse e fino ad ambienti con qualche centinaio di container, ci si può affidare a manager Docker relativamente semplici, come quelli descritti nell'altro articolo del mese. Per fare le cose ancora più in grande, invece, ma sempre utilizzando quanto più possibile software Open Source, le soluzioni migliori del momento sembrano essere il progetto Mesos (<http://mesos.apache.org>) oppure il suo spin-off commerciale chiamato Mesosphere Datacenter Operating System (Dcos, <https://mesosphere.com>). Entrambi i prodotti condividono una visione del data center inteso come un unico server, che oltretutto deve essere estremamente *elastico*. Con quest'ultimo termine si indica la capacità di adattarsi con grande rapidità, ma riducendo al minimo interventi manuali degli amministratori, a richieste continuamente variabili. La domanda è quella di poter far girare in un unico data center, magari virtuale, sia microservizi sia applicazioni "Big Data", e se necessario ogni giorno in combinazioni diverse.

Apache Mesos risponde a queste esigenze fornendo appunto una intera piattaforma per "sviluppare e gestire software in un data center, come se quest'ultimo fosse un unico computer". Le varie funzioni e interfacce di Mesos creano, al di sopra dei server fisici, processori, memoria, storage e altre risorse virtuali *su misura* per sciami di container. L'ambiente risultante è capace da un lato di fornire via cloud, da tutti i computer fisici su cui viene spalmato, sia microservizi sia servizi di calcolo molto pesanti; e dall'altro di adattarsi da solo, o quasi, a variazioni del carico o dell'infrastruttura. Se una macchina si libera, gli algoritmi di Mesos possono spostarvi da soli parte dei container che hanno bisogno di più risorse. Se al contrario si guasta o va offline, gli stessi algoritmi possono trovare ospiti per i container che ospitava.

Dcos è una sorta di "meta-sistema" operativo, che gira direttamente su Linux. La sua versione base è utilizzabile gratuitamente nei servizi cloud Aws di Amazon. Quella commerciale già gestisce milioni di container, anche

in compagnie come Groupon, Netflix e Twitter, e può girare direttamente su hardware o in qualsiasi cloud, da Vmware ad Amazon e Azure. Oltre a

Docker, Dcos permette di costruire e gestire cluster di contenitori anche con la piattaforma Kubernetes di Google, di cui parleremo fra poco.

In generale, piattaforme come Mesos e Dcos sono particolarmente appropriate in due casi estremi. Uno è la gestione centralizzata di ambienti pesanti e integrati, ma di fatto sparsi su

parecchie migliaia di container, magari ospitati da centinaia di server. All'estremo opposto, anziché pochi carichi enormi, si trovano una miriade di piccoli carichi indipendenti, molto diversi fra loro e in combinazioni sempre variabili. In entrambi i casi la già citata *elasticità* di Mesos è indispensabile per reggere il carico *organizzativo*, più che quello sull'hardware, minimizzando sia le spese sia lo spreco di risorse hardware.

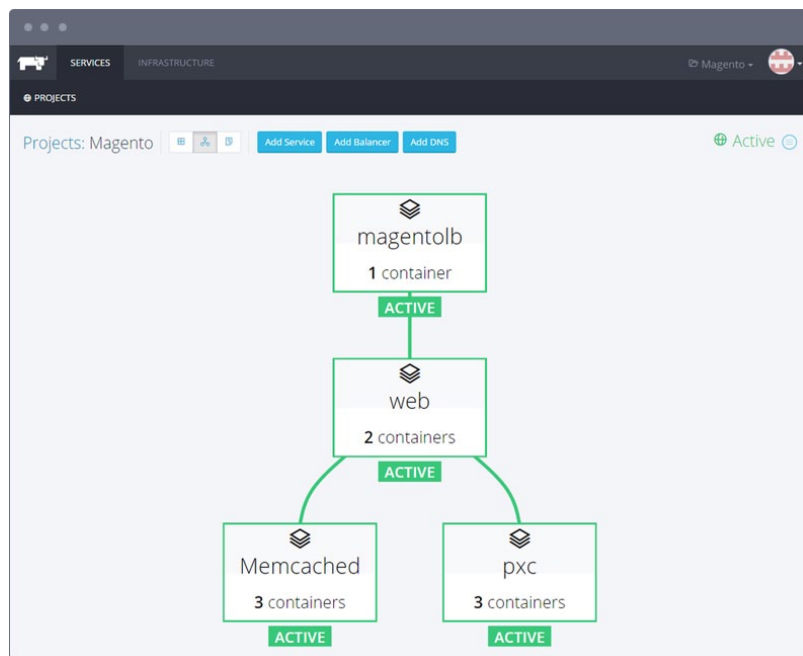
### Dcos: lo spinoff

La versione commerciale di Mesos è utilizzata da società come Netflix, Groupon e Twitter

## COREOS

Facilitare la gestione di grandi cluster di container è anche la missione di CoreOS (<http://coreos.com>), un sistema operativo minimo, ma con supporto completo e nativo per i container più popolari e i loro cluster. CoreOS è stato progettato per essere facile da avviare sulla maggior parte dei fornitori di servizi cloud. Pur mantenendo la compatibilità con altri sistemi, i suoi sviluppatori hanno anche creato da zero strumenti per CoreOS come "fleet", per la gestione di cluster ed "etcd" per il rilevamento dei servizi e aggiornamento delle configurazioni nel cluster.

Soprattutto, a dicembre 2014, CoreOS ha rilasciato un concorrente Open Source di Docker chiamato rct o per semplicità Rocket (razzo, <https://coreos.com/blog/rocket>). Il motivo ufficiale per questo apparente "doppione" è che Docker sarebbe troppo complesso, poco maneggevole, ma soprattutto troppo controllato da una sola società per essere davvero portabile. Al momento è ancora presto per capire se e come Rocket riuscirà davvero a scalfire la posizione dominante di Docker nel campo dei container Linux, oggi talmente forte che nemmeno CoreOS ha alcuna intenzione di abbandonare la piena compatibilità con quello standard.



Rancher dà a ogni suo utente una visione d'insieme molto chiara di quanti e quali container sta usando. Offre anche diverse funzioni per distribuirne dinamicamente il carico sui server disponibili.

## KUBERNETES

Nel numero scorso abbiamo visto come Google, nel corso degli anni, ha sviluppato tutta una serie di software per gestire container di cui ha reso Open Source la parte più a basso livello, chiamata *lmctfy* ("Let Me Contain That For You"). Da allora, per avere a disposizione sistemi di *sua* gradimento con cui coordinare e distribuire su più container carichi di lavoro imprevedibili e completamente indipendenti (anche quando una sola di quelle applicazioni potrebbe aver bisogno di *centinaia* di container per funzionare correttamente) Google ha rilasciato altro software per container, sempre con licenza Open Source. Stiamo parlando di Kubernetes (<http://kubernetes.io>), una parola greca che significa nocchiero, timoniere.

Kubernetes è appunto un sistema di gestione ad alto livello, anzi: "orchestrazione" di container. Quelli di default sono sempre Docker, ma Kubernetes è compatibile con ambienti multi-cloud, cioè con sciami o cluster di container eterogenei, e sparpagliati su più fornitori di cloud.

Sia con Kubernetes, sia con i prodotti descritti nell'altro articolo del mese, si parla di orchestrazione per sottolineare un fatto ben preciso: oltre a gestire i singoli container bisogna anche, anzi soprattutto, farli lavorare *di concerto, sempre in sincronia e in buon accordo fra loro*. Kubernetes fa proprio questo, spostando carichi di lavoro continuamente variabili fra tutti i computer fisici che ospitano i suoi container. In aggiunta, Kubernetes garantisce che la comunicazione fra tutti i container di uno stesso cluster funzioni al meglio, e in piena sicurezza. L'aspetto più interessante di Kubernetes è però la stessa

impostazione che, come abbiamo visto nel numero scorso, è condivisa a livello più basso da *lmctfy*. L'amministratore o utente finale di Kubernetes non deve impostare direttamente quantità, requisiti e vincoli dei container che gli servono, ma solo dichiarare le *intenzioni* dei servizi che dovranno girare al loro interno, e quali prestazioni si aspetta. Pensa poi Kubernetes a decidere quanti container servono, e come distribuirli fra tutti i computer fisici che ospitano il cloud che controlla, in base al loro effettivo carico in quel particolare momento.

## CONTAINER ENGINE: TUTTI I CONTAINER CHE VUOI, DENTRO GOOGLE

Sviluppare software Open Source così complesso e critico come Kubernetes non è stato certo un'iniziativa filantropica per Google, ma non è nemmeno soltanto una risposta a sue esigenze *interne*. Chiunque, infatti, volesse gestire parecchi container Docker con l'ap-proccio di Kubernetes, può farlo con il minimo sforzo, proprio affidandosi al servizio Container Engine di Google (<https://cloud.google.com/container-engine>). Iscrivendosi si hanno subito a disposizione, senza doverli installare o configurare, Kubernetes e tutto il resto del software che permette di istanziare e usare container Docker direttamente nei cloud di Google.

A garantire che tutto sia sempre in linea nonostante eventuali guasti o sovraccarichi, così come a curare gli aggiornamenti di Kubernetes e degli altri componenti, pensa Google. È disponibile anche un servizio di archivio di immagini Docker (Google Container Registry). Basta quindi definire le proprie intenzioni, come descritto nel

paragrafo precedente, per iniziare a lavorare. Penserà il Container Engine a piazzare e avviare nella nuvola i container che servono, spostandoli se necessario da server a server al variare del carico di lavoro. È possibile anche gestire cluster ibridi, cioè residenti in parte sul cloud di Google, e in parte altrove.

I maligni potrebbero dire che Container Engine è solo l'ultimo sistema di Google, dopo Gmail, Drive e tanti altri, di tenere d'occhio tutto quel succede nel mondo e di creare sempre più dipendenze dalla sua gigantesca infrastruttura hardware. D'altro canto, è difficile negare che proposte come Container Engine possono far scoprire subito i benefici dei container a molte più persone e aziende di quanto potrebbero fare altri sistemi.

Inoltre, almeno in teoria, quegli stessi utenti potrebbero facilmente passare in ogni momento ad altri fornitori di container, o a gestirsi da soli, visto che gran parte di Container Engine è composta da standard aperti e software Open Source. Se questo avverrà, probabilmente sarà anche grazie a distribuzioni commerciali come Tectonics (<https://tectonic.com>).

Questo sistema operativo, che combina Kubernetes con CoreOS ed è sviluppato dagli stessi autori di quest'ultimo, si presenta come "la soluzione universale per mettere in campo e gestire container, per aziende di qualsiasi dimensione". Oltre al sistema operativo di base e al gestore di cluster, infatti, Tectonic integra tutte le tecnologie Open Source necessarie, dal sign-on integrato a un archivio centralizzato di immagini, per lavorare in modo molto simile a quello di Container Engine, ma fuori dal cloud di Google.



## RISORSE

**U**n buon sistema, nonché semplicissimo, per farsi un'idea della portata del "fenomeno Docker" è scorrere il "Docker Public Registry" ufficiale (<https://index.docker.io>), ovvero l'elenco di tutte le applicazioni ufficialmente disponibili come container Docker. Le tante piattaforme su cui può girare CoreOS, dall'hardware nudo e crudo ("bare metal") ai maggiori cloud del pianeta, sono elencate su <https://coreos.com/os/docs/latest>. I documenti migliori sul Container Engine sono quelli nella sua home page (<https://cloud.google.com/container-engine/docs>). Per saperne di più sulle tecnologie di gestione Docker consigliamo invece gli articoli "Sette ragioni per usare CoreOS con Docker" ([www.airpair.com/coreos/posts/coreos-with-docker](http://www.airpair.com/coreos/posts/coreos-with-docker)) e l'annuncio ufficiale originario di Rocket (<https://coreos.com/blog/rocket>). Quest'ultimo, oltre a una buona descrizione di quel formato di container, fornisce una buona visione d'insieme dei problemi principali che sviluppatori e gestori di container devono affrontare.





## NEWS

TUTTI PARLANO  
DI DOCKER. INTANTO,  
UBUNTU...

**S**e c'è una cosa che dovrebbe essere evidente, da questo numero della rubrica, è che in questo periodo il mondo dei container Open Source per Linux non vede quasi nient'altro che Docker, o i suoi diretti concorrenti come il Rocket di CoreOS, citato nell'articolo principale. Quasi al momento di andare in stampa è però arrivato un annuncio che vorrebbe contrastare proprio questa tendenza, o almeno completarla. Canonical, lo sponsor commerciale di Ubuntu, ha infatti comunicato il rilascio della versione 2.0 di Lxd ([www.ubuntu.com/cloud/tools/lxd](http://www.ubuntu.com/cloud/tools/lxd)). Questo prodotto è proprio il software di gestione dell'altra grande categoria di container Linux, quella basata sulla tecnologia Lxc descritta il mese scorso in queste stesse pagine. La differenza principale fra i due approcci è che Docker è fortemente orientato, o almeno viene usato quasi solo in quel modo, alla distribuzione di singole applicazioni su cloud generici. Lxc, invece, è più vicino a essere una macchina virtuale Linux generica, ma molto, molto più leggera di quelle basate su "ipervisor" (gestori di macchine virtuali) tradizionali come Xen o Kvm.

Lxd 2.0, in effetti, è stato presentato come il più compatto e veloce ipervisore del mondo, capace di offrire prestazioni e densità, cioè numero di applicazioni per singolo server fisico, finora non disponibili con nessun altro prodotto. Per questo, e per il fatto che offre anche servizi non disponibili con Docker, Canonical presenta Lxd 2.0 come una alternativa e un completamento ideale di Docker, più che come suo concorrente. Questo è visibile soprattutto nel fatto che Lxd è chiamato *machine container*, anziché *app container* come fanno tutti con Docker. La differenza di nomi sottolinea il fatto che si possono creare e utilizzare container Docker, con grande efficienza, proprio all'interno di un container Lxd.

## LIBRE OFFICE SU MISURA PER LA PUBBLICA AMMINISTRAZIONE INGLESE

**N**ei numeri scorsi abbiamo segnalato la prossima migrazione a Libre Office del nostro Ministero della Difesa. Questo mese un annuncio del genere, ma su scala molto maggiore, arriva da Oltremania. Il governo britannico ha infatti stretto un accordo con l'azienda software chiamata Collabora Productivity, per adottare in tutti i suoi uffici la suite chiamata GovOffice ([www.collaboraoffice.com/collabora-govoffice.php](http://www.collaboraoffice.com/collabora-govoffice.php)). Quest'ultima non sarebbe altro che una versione commerciale di Libre Office, arricchita di parecchi componenti aggiuntivi e in generale configurata su misura per le esigenze dell'amministrazione britannica. Gli extra inclusi nel pacchetto includono, ad esempio, interfacce più complete di quelle standard per effettuare la migrazione dei documenti a formati aperti, gestione centralizzata di installazioni e aggiornamenti e garanzie di supporto a lungo termine. GovOffice sarà disponibile ai suoi utenti anche da smartphone o da interfaccia Web. Oltre alle pubbliche amministrazioni centrali, l'accordo copre anche tutte le organizzazioni non a scopo di lucro, sia governative sia esterne, ma comunque al servizio delle stesse amministrazioni.

