



*La virtualizzazione consente alta flessibilità e convenienza in molteplici attività. Quali sono le componenti a basso livello su cui intervenire?*

## Container: sempre più Cloud e aperti

La domanda di soluzioni Cloud è sempre più alta, per non dire isterica. Aziende di tutti i tipi e dimensioni, nazionali o estere, sono pronte ad assistere chi vuol fare di tutto e di più con computer e software, purché siano appunto “su Cloud”, cioè gestiti da qualcun altro, altrove. Lavorare e studiare *interamente* su Cloud è molto di più che usare posta elettronica o word processor come Gmail o Google Docs. Oggi, però, può capitare davvero a chiunque, volente o nolente, incluse scuole, Pubbliche Amministrazioni e Piccole o Medie Imprese. I motivi sono facilmente intuibili: costi a parte, lavorare su infrastrutture Cloud è molto, molto più semplice da gestire che farsi e tenere in piedi una rete informatica propria. Il Cloud è anche immensamente più flessibile di reti hardware vere e proprie, di proprietà dell'organizzazione. Un vantaggio non trascurabile, in quest'epoca di lavoro (anche per intere aziende) sempre meno stabile e prevedibile.

Serve tre volte più potenza di calcolo del solito, ma solo per una settimana, per sbrigare degli ordini inaspettati? Oltre a costare uno sproposito, fronteggiare imprevisibili del genere comprando hardware sarebbe un incubo organizzativo e burocratico. Grazie al Cloud, invece, basta ordinare server virtuali per poterli usare in pochi minuti e liberarsene con un clic appena

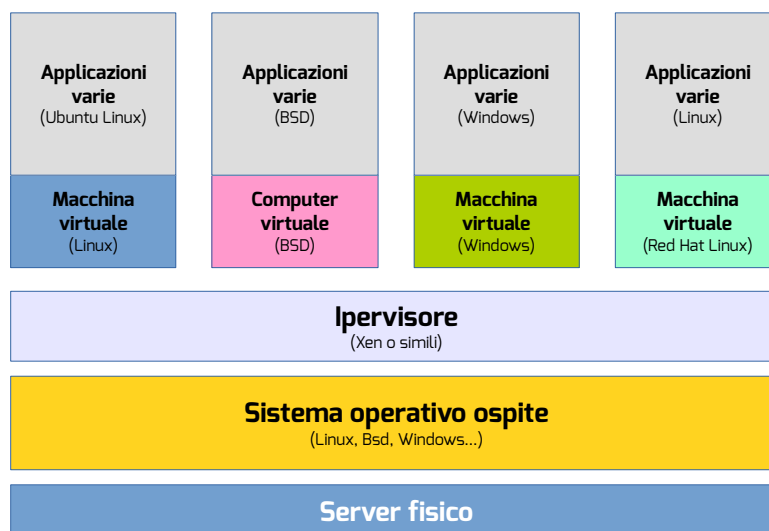
non servono più, pagando pochissimo. Come tutto il resto di Internet, anche il Cloud e tutti i servizi che esso rende possibile si sono basati, fin dall'inizio e in larghissima parte, su software e protocolli Open Source. Proprio per questo, negli ultimi uno o due anni quelle tecnologie hanno subito una trasformazione profonda. Dopo parecchio tempo, infatti, sembrano finalmente giunti a maturazione

nuovi strumenti, sia nel kernel Linux vero e proprio, sia nelle distribuzioni costruite intorno a esso.

Questo mese presentiamo proprio le parti più a basso livello di queste tecnologie, tutte basate sul concetto di *container*.

Il mese prossimo vedremo come gestire, su Linux, le grandi concentrazioni di container e i problemi che queste possono creare.

### VIRTUALIZZAZIONE CON VIRTUAL MACHINE



Le macchine virtuali sono modelli software di un computer completo. Grazie ad esse, diversi sistemi operativi possono girare simultaneamente su uno stesso server.

## IN PRINCIPIO, C'ERANO LE MACCHINE VIRTUALI

Il cloud è l'infrastruttura che ci consente di svolgere quasi qualsiasi attività informatica su hardware che non vedremo mai, e di solito senza installare di persona alcun software. Alla sua base, da sempre, c'è possibilità di creare e mettere a disposizione di tutti interi *computer virtuali*, o parti di essi. Questa tecnologia è entrata nella coscienza informatica di massa nel 1999, con il rilascio di VmWare Workstation. Da allora le *Virtual Machines*, o Vm per brevità, sono arrivate dappertutto. Oggi

chiunque, con pochi Euro al mese, può noleggiare un server virtuale privato tutto per sé in migliaia di data center. Almeno concettualmente, però, le Virtual Machine di oggi sono identiche a quelle degli inizi: modelli software di computer completi di tutto, dal Bios a processori, dischi, schede di rete e se necessario anche altre periferiche hardware. Grazie a questa loro natura le Vm sono tecnologia matura, sicura e ormai standardizzata tanto quanto l'hardware per Pc. Chi ne noleggia una non ha alcun bisogno di sapere su quale processore girerà: deve soltanto specificare quale *sistema operativo completo* vuole trovarci già installato. A quel punto tutto quanto, dalla procedura di boot all'amministrazione di sistema, funzionerà come su un computer reale con lo stesso sistema operativo. Il tutto con rallentamenti, grazie al supporto dei processori moderni per la virtualizzazione direttamente a livello hardware, di pochi punti percentuali. Questo progresso non ha impedito alla tecnologia Vm di causare sempre più insoddisfazione fra utenti e (soprattutto) fornitori di Cloud. I motivi sono gli stessi che abbiamo appena descritto: prima di tutto, "affittare" interi sistemi operativi a chi ha bisogno di far girare soltanto due o tre programmi è un grosso spreco di memoria, cicli Cpu e spazio su disco, sempre meno tollerabile

nell'economia moderna. Per non parlare del suo costo *ambientale*, se è vero che i data center sono ormai produttori non trascurabili di CO2 a livello mondiale. Idem per le prestazioni. "Pochi punti percentuali" potevano essere trascurabili quando le Vm le usavano pochissimi, non oggi. Ancora più grave è la lentezza con cui le macchine virtuali possono essere create e fornite *on demand*, soprattutto a chi ne chiede decine o centinaia alla volta. O la lentezza con cui le stesse macchine ripartono ogni volta che si fermano (sono veri e propri computer con tanto di boot, ricordate?). Tutte inefficienze tollerabili quando la domanda di cloud, oltre che molto minore in termini assoluti, era anche molto più stabile di oggi, ma non oggi.

## ARRIVANO I CONTAINER

I limiti dei sistemi menzionati nel box qui a sinistra, insieme ai problemi di efficienza delle Vm vere e proprie, erano già noti dieci o quindici anni fa. Le "prigioni" (Jails) di Bsd o le "zone" di Solaris cercavano proprio di fornire qualcosa a metà fra quei due estremi. Purtroppo sia quei sistemi, sia i primi dello stesso genere per Linux o non erano Open Source, o erano troppo complicati da utilizzare, o avevano limiti pratici di altro genere. Questa situazione è cambiata drasticamente con

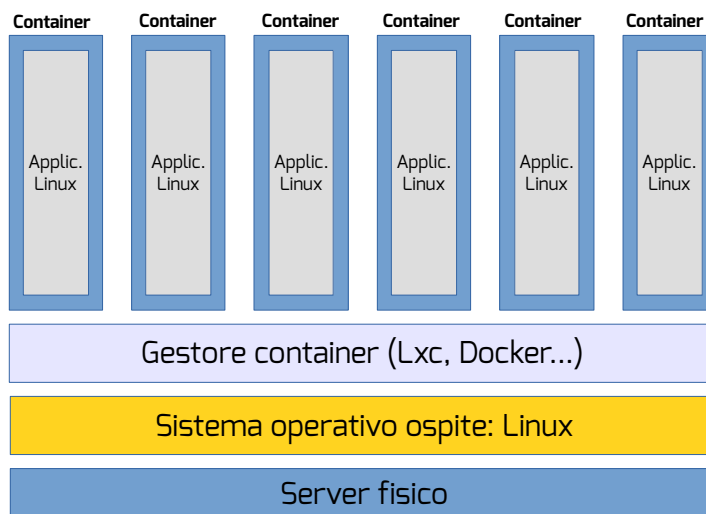
## CHROOT E GLI ALTRI

Macchine virtuali e container creano ambienti in cui un utente può, fino a un certo punto, fare quel che vuole senza disturbare o essere disturbato dagli altri utenti dello stesso computer fisico. In realtà, forme più rudimentali di "segregazione" sono state presenti nel mondo Unix, e quindi Linux, fin dai suoi inizi.

Far vedere, ai processi che un utente fa girare, solo una parte del file system è possibile su Linux con chroot. A differenza dei permessi di accesso a file e cartelle, che impediscono solamente di *aprire* certi file o cartelle, chroot cambia la directory "radice" (in Inglese root, appunto) di un processo, facendogli *credere* che vede tutto il filesystem mentre in realtà è rinchiuso in una sua parte.

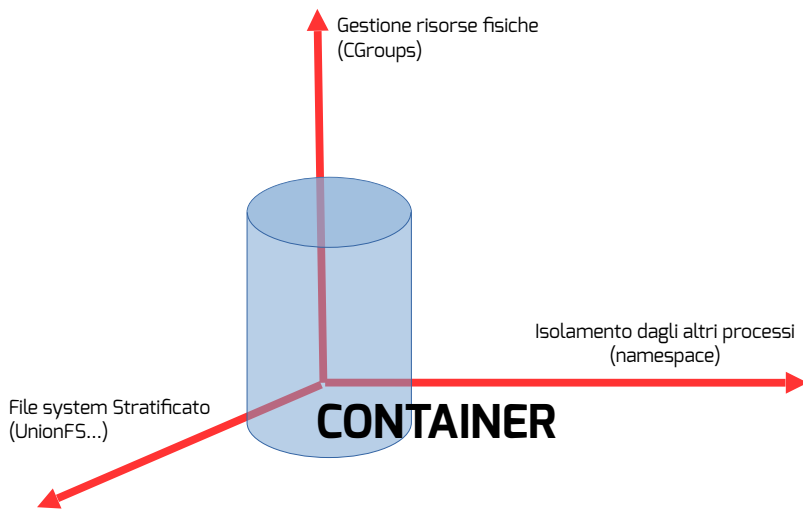
A un livello completamente diverso, comandi come "nice" ("beneducato") obbligano un processo a cedere il passo agli altri, o pongono limiti sulla quantità di memoria o file accessibili simultaneamente da un processo. Tutti questi meccanismi, ancora adeguati per workstation con pochissimi processi, sono però troppo rozzi e poco flessibili per accontentare le centinaia o migliaia di utenti, sempre diversi e in principio poco affidabili, che un server su cloud deve poter gestire ogni ora.

## CONTAINER PER VIRTUALIZZAZIONE SU LINUX



A differenza delle Vm, i container Linux condividono, fra le varie applicazioni che devono girare sullo stesso computer, quasi l'intero sistema operativo. Meno flessibilità, ma prestazioni molto maggiori.

## CONTAINER PER VIRTUALIZZAZIONE SU LINUX



Le tre dimensioni di un container Linux: Cgroups regola l'uso delle risorse, Namespace di ciò che è visibile dal container, un file system speciale per scrivere file al suo interno.

i progetti di cui parleremo fra poco, e soprattutto con il loro pieno supporto nel normale kernel Linux, senza cioè costringere chi ne ha bisogno a compilarne versioni speciali. La tecnologia complessivamente chiamata *container* evita i problemi delle Vm virtualizzando completamente non un intero sistema operativo, ma tutto e solo l'ambiente di cui ha bisogno un singolo *processo*. Per capire e apprezzare la differenza bisogna ovviamente aver chiara qual è, almeno su Linux, la differenza fra un *programma software* e il suo *processo* corrispondente.

Un programma software non è altro che una sequenza di istruzioni a basso livello, direttamente eseguibili da un processore. Per eseguire quelle istruzioni, cioè per far girare effettivamente quel programma, il sistema operativo deve creare un processo, cioè una copia temporanea ma *vivente* di quel programma. Ogni processo ha un suo identificatore numerico ed è composto dalla zona di memoria riservata a quell'esecuzione, più tutta una serie di dati e metadati dei tipi più disparati: file da leggere o scrivere, connessioni con altri processi (inclusi quelli che *esso stesso* ha lanciato), certificati digitali, librerie condivise, messaggi da o per il kernel, lingua di sistema o simili configurazioni generali, eccetera. In altre

parole, perché un programma funzioni, il processo corrispondente deve avere a disposizione tutto un ambiente che è *molto più piccolo* del sistema operativo completo, ma *molto più ricco* del mero file eseguibile di quel programma. Un container Linux è proprio una capsula che rinchioda perfettamente un ambiente così, facendo credere ai processi al suo interno di avere a disposizione il loro intero sistema operativo standard.

### DENTRO I CONTAINER PER LINUX

Le macchine virtuali garantiscono che un certo programma non consumi mai più di una certa percentuale della memoria fisica in maniera tanto brutale quanto imprecisa, cioè limitando la Ram utilizzabile *dall'intera macchina virtuale*. I container Linux moderni forniscono garanzie del genere con molta più eleganza e precisione, grazie funzioni kernel come Cgroups (*Control Groups*). I parametri Cgroups associati a ogni container permettono di definire con molta precisione quanto i suoi processi potranno sfruttare tutte le *risorse fisiche finite* del computer, a partire da: memoria, larghezza di banda per le comunicazioni e numero di accessi al secondo ai dispositivi di storage. Dal punto di vista dell'uso ottimale

delle risorse, Cgroups fa meraviglie rispetto ai comandi come "nice" (vedi box) o alle Vm complete, ma non basta per rendere i container utilizzabili. Se ci fosse solo Cgroups, infatti, i processi dentro un container potrebbero ancora ficcare il naso dove non devono, cioè riuscire a vedere file, librerie software, periferiche e altri programmi in esecuzione... di cui invece dovrebbero ignorare completamente l'esistenza. A fornire questo isolamento provvedono i sei tipi diversi di spazi di nomi riservati (*namespace*) supportati dai kernel Linux moderni.

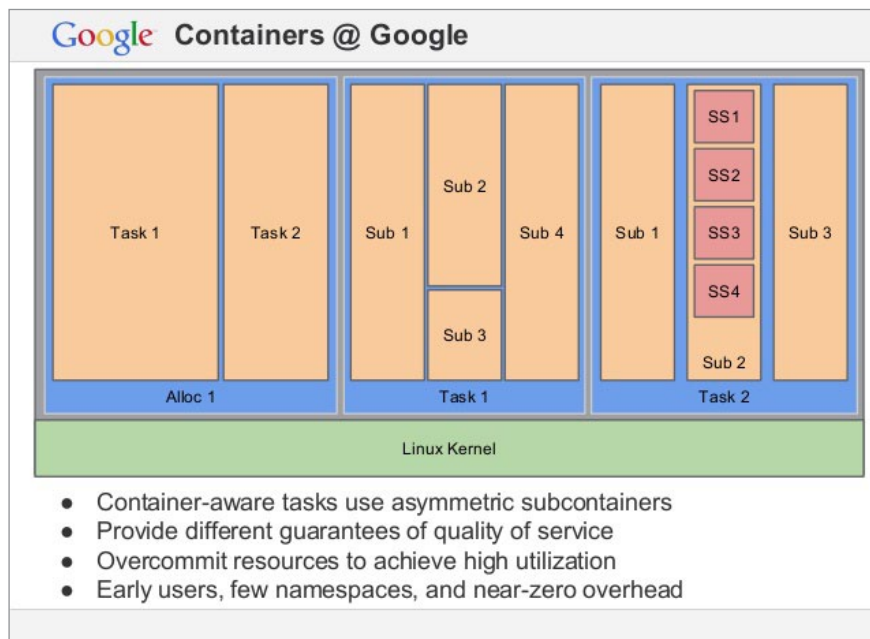
Ogni namespace fa vedere al container a cui è associato solo una parte di una certa risorsa del computer fisico su cui gira. I namespace del file system rinchiodano tutti i processi di un container all'interno di una certa directory. Altri namespace permettono di assegnare un identificatore numerico aggiuntivo, diverso da quello vero, visto dal kernel, a utenti, gruppi e processi. In questo modo il primo utente di un certo container può credere di essere il numero 1 (cioè il super-utente *root*) anche se in realtà non ha alcun potere del genere sul resto del computer. Allo stesso tempo, i processi interni a un container non possono nemmeno accorgersi di quelli al di fuori, perché non possono leggerne i numeri.

Oltre a questi namespace fondamentali, ci sono quelli Uts, Ipc e network che isolano nella stessa maniera i nomi di host, le aree di memoria per comunicazione diretta fra processi e le interfacce di rete.

Nel complesso, il bello dei cgroup e namespace Linux è che, mentre creano versioni separate di file, utenti e altre risorse, tutte viventi solo all'interno di un container, si tratta sempre di *risorse standard su Linux*. Un sistema operativo che ospiti macchine virtuali può vedere ben poco del loro interno, e quindi non accorgersi se, ad esempio, c'è della memoria inutilizzata che farebbe comodo ad altre Vm. Usando i container, invece, lo stesso sistema operativo potrà vedere e gestire direttamente, con gli stessi strumenti di sempre, normali utenti, processi e file.

In pratica, Namespace, Cgroup e altre tecnologie di base per i container Linux sono già utilizzabili su quasi tutte le distribuzioni grazie ai vari progetti coordinati dalla comunità di <http://linuxcontainers.org>. Quello base, chiamato Lxc, aspira a fornire un ambiente

## LMCTFY: CONTENITORI ASIMMETRICI



I container Linux di lmcftfy: contenitori asimmetrici, ognuno dimensionato in base al suo scopo (non ai programmi contenuti), pronti a cedere il passo a container con compiti più urgenti.

“più simile possibile a una installazione Linux standard, ma senza la necessità di un kernel separato”. Installandolo si hanno a disposizione librerie per vari linguaggi, nonché strumenti da riga di comando relativamente semplici, per creare e gestire container completi, subito utilizzabili per diverse attività. Il pacchetto include vari modelli di container per le distribuzioni più popolari, da usare come sono o adattare alle proprie esigenze. Un altro vantaggio non trascurabile di Lxc è che funziona anche su architetture diverse da quella x86, a cominciare dai processori Arm utilizzati nei microcomputer come Raspberry Pi. I due strumenti principali su cui si appoggia Lxc sono un file system ad hoc, chiamato appunto Lxcfs, e il daemon (server locale) CGManager. Il primo consente ai processi all'interno di un container di avere loro versioni private anche di file di sistema, normalmente usati per comunicare col kernel. CGManager permette di creare cgroups e container anche a utenti e processi già confinati, ovviamente a loro insaputa, all'interno di un container. L'ultimo “pezzo forte” fornito da linuxcontainer.org è il sistema di gestione Lxd, su cui torneremo nel prossimo numero.

### DOCKER, L'ALTRO LATO DEI CONTAINER

Lxc attacca direttamente, nel modo più semplice possibile, il problema originario per risolvere il quale sono nati i container: fornire ambienti Linux chiusi, in cui un programma Linux possa girare come vuole, credendo di avere un intero computer tutto per sé, ma in maniera molto più efficiente di una macchina virtuale. Iniziando a pensare a un container come a una capsula per uno e un solo programma con tutte le sue dipendenze fa però nascere, per così dire, una tentazione un po' diversa: quella di usare i container non per condividere fra più “clienti” possibile un singolo computer, ma come sistema di distribuzione software. Un container consentirebbe cioè di distribuire applicazioni che possono girare, sempre allo stesso modo, non solo in ambienti o sistemi operativi diversi ma anche, e soprattutto, in Cloud di fornitori diversi. Questo secondo scenario è una delle spinte principali dietro la libreria libcontainer (<https://github.com/>

docker/libcontainer) e la sua principale applicazione, Docker ([www.docker.com](http://www.docker.com)). Tramite Docker, libcontainer fornisce container sempre basati su Namespace e Cgroup, ma utilizzabili anche senza privilegi da amministratore di sistema, e soprattutto sempre nella stessa maniera, su qualsiasi versione di Linux e non solo.

È proprio per questo motivo che Docker, il cui slogan è appunto “crea, distribuisce ed esegui ovunque qualsiasi programma”, ha suscitato tanto interesse negli ultimi due anni da essere oggi promosso da partner tanto potenti quanto diversi fra loro. Citando solo alcuni dei più famosi, oggi lavorano su Docker Red Hat, Canonical, Amazon, Google e persino Microsoft, per portare Docker anche nei suoi servizi cloud Azure.

Tecnicamente, i container Docker sono costruiti su immagini dell'ambiente software di riferimento, scaricabili anche da archivi online come i normali pacchetti binari Linux. Con Docker si possono poi sovrapporre a quelle immagini altre, con file o parametri differenti, per creare altrettanti container ognuno con una configurazione unica. Qualsiasi sia il numero di strati, cioè di immagini sovrapposte, di un container Docker, nel momento in cui esso viene attivato viene aggiunto un ulteriore strato scrivibile, grazie al file system UnionFS (<http://unionfs.filesystems.org>). In questo modo l'applicazione dentro il container

### Per tutte le nuvole

I container permettono di distribuire applicazioni anche in soluzioni Cloud di fornitori diversi

può creare tranquillamente tutti i file che vuole, oppure “modificare” le sue copie di quelli di sistema. Volendo, è possibile salvare quello strato scrivibile, insieme a tutti quelli sottostanti, con sistemi simili a quelli con cui i programmatori mantengono, su archivi condivisi online, le varie versioni del codice che scrivono. È quindi facile salvare online, e volendo condividerla con chiunque, l'immagine di un container con stato o configurazione diversi dal suo antenato diretto, pronto a ripartire sullo stesso o su un altro computer.

### CONTAINER PER TUTTI? NON PROPRIO

Riassumendo quanto esposto finora, potremmo dire che le macchine virtuali hanno separato uso, proprietà e



gestione dei sistemi operativi da quelli dell'hardware. I container, invece, separano uso e gestione di singole applicazioni da quelli di tutto il resto del software, con notevoli semplificazioni per fornitori e utenti di Cloud. Le Vm sono scatole nere, mentre i container e i processi che girano al loro interno sono sempre controllabili, ottimizzabili e riconfigurabili quasi in tempo reale dall'amministratore di sistema. Oltre a questo, a parità di hardware i container possono far girare anche centinaia di applicazioni indipendenti in più rispetto alle Vm. L'era delle Vm è finita, dunque? No, affatto. In realtà, container e Vm sono complementari, e se nei prossimi anni è sicuro che useremo sempre di più i primi, le seconde non spariranno affatto. Un primo motivo è la standardizzazione, ancora maggiore nel secondo caso, un altro è la sicurezza. Le macchine virtuali danno ancora maggiori garanzie di isolare completamente fra loro i vari "coinquilini" di uno stesso server. Quando privacy e protezione da furto di dati sono i requisiti principali, dunque, conviene aspettare prima di passare ai container, o se è per questo ai cloud pubblici, anziché interni alla propria organizzazione. Le macchine virtuali sono ancora la soluzione migliore, se non l'unica, anche quando occorre far girare, sullo stesso server, programmi per sistemi operativi diversi. Alla fine, però, quel che fa pendere la bilancia da una parte o dall'altra è quante applicazioni diverse e indipendenti, o meglio quanti gruppi di tali applicazioni, si vogliono avere sullo stesso hardware. Il motivo per cui sono nati i container, in fondo, è far girare in un ambiente virtuale più leggero possibile, molto più di un sistema operativo, una **singola** applicazione.

Di conseguenza, se è necessario dare a ogni "coinquilino" la possibilità di far girare quante applicazioni vuole, magari in maniera sempre diversa nel tempo, i container potrebbero creare più complicazioni, con pochi guadagni concreti, rispetto alle macchine virtuali. Se invece l'obiettivo è far girare su uno stesso computer più copie possibile di singole applicazioni, anche diverse ma tutte isolate dalle altre, la scelta è facile: i container saranno molto, ma molto più efficienti e facili da gestire. Ma di come gestire i container parleremo nel prossimo numero. •



## LA SOLUZIONE LMCTFY, CONTAINER STILE GOOGLE

I container Open Source sono sulla ribalta informatica mondiale solo dall'anno scorso. La loro storia però è iniziata almeno quindici anni fa e in essa Google ha svolto un ruolo interessante, da conoscere per apprezzare al meglio le possibilità dei container.

La funzione Open Source Cgroups ("Control Groups"), per esempio, è nata proprio in Google a metà degli anni 2000. In generale, Cgroups e altri strumenti del genere hanno beneficiato dell'uso intensivo che ne fa Google nella sua gigantesca infrastruttura. Se oggi i container Linux funzionano bene è anche per tutto lo stress a cui sono sottoposti in quei data center, equivalente a un controllo qualità gigantesco e continuo, ben difficilmente realizzabile da altri operatori. In una conferenza a maggio 2014 Google ha spiegato che, ormai da anni, *"tutto in Google gira all'interno di un contenitore"*. Inclusa ogni singola sessione Gmail, Google Plus, Drive e così via, per un ritmo di qualche miliardo di container creato *ogni settimana*.

Per far fronte a queste esigenze uniche, Google ha dovuto comunque seguire una differente dal resto del mondo Linux. Il kernel Linux usa Cgroups per limitare l'uso che ogni container può fare delle risorse fisiche del server ospite, e i namespace per limitare quanta e quale parte del server stesso riesce a vedere. Ma gli amministratori di sistema impostano namespace e Cgroups con gli strumenti da riga di comando di Lxc o Docker, fatti apposta per rendere il lavoro più semplice possibile. L'approccio di Google, invece, per ovvie ragioni ha dovuto sempre anteporre alla facilità d'uso l'ottimizzazione delle prestazioni e quella di utilizzo dell'hardware. È questo che lo ha portato, internamente, a sostituire Lxc con un software dall'impossibile nome di *lmctfy* (<http://lmctfy.io>) che letteralmente sarebbe l'acronimo della frase inglese *"permettiti di contenere [quel software] per te"*. *lmctfy* permette di configurare container in base al loro scopo e alle loro esigenze, ma senza dover conoscere i dettagli di Cgroups, con l'equivalente software dell'overbooking delle compagnie aeree. Le applicazioni da far girare con *lmctfy*, infatti, possono specificare le proprie esigenze di priorità e latenza, cioè ritardo massimo fra ricezione ed esecuzione di un comando. L'amministratore, dal canto suo, può assegnare a un server un carico complessivo maggiore di quello effettivamente sostenibile dal server stesso. Grazie a *lmctfy*, i container con applicazioni a bassa priorità verranno automaticamente messi in coda dopo quelli con lavori più urgenti. È possibile creare, dentro ogni container, altri container con le stesse caratteristiche. Grazie a questa architettura è possibile sfruttare fino all'ultimo ciclo di Cpu di un intero data center, e in effetti *lmctfy* è utilizzato per gestire tutti i container di Google fin dal 2007. Nei primi anni era completamente integrato con altre parti dell'infrastruttura, in seguito, l'architettura venne resa molto più modulare, portando a un *lmctfy* utile e riutilizzabile anche fuori da Google. Il punto d'arrivo di questo percorso fu il rilascio, a ottobre 2013, di *lmctfy* con licenza Open Source. Un "attacco" a Docker e simili prodotti Open Source, cioè un modo di imporre la visione e i requisiti Google per i container su Linux? Fortunatamente, pare proprio di no!

Google, infatti, prima ha aggiunto Docker alle piattaforme container utilizzabili nel suo Cloud. Poi, a maggio 2015, ha annunciato di aver fermato il progetto per lasciare, in un certo senso, campo libero a Docker e alla sua libreria di base, *libcontainer*. La parte centrale di *lmctfy* verrà integrata in *libcontainer*, che in futuro dovrebbe sostituire *lmctfy* anche nel cloud di Google.

# NEWS

## LIBRE OFFICE? ABILE E ARRUOLATO!



L'estate 2015 ha visto, fra le altre cose, diverse polemiche sulla decisione del comune di Pesaro di abbandonare OpenOffice, a cui era passato nel 2011, per tornare a prodotti Microsoft. In quell'occasione, una causa importante dell'inversione di marcia era stata una prima migrazione a OpenOffice approssimativa, e mai realmente completata. Questo non sarà sicuramente il caso di una nuova migrazione all'Open Source, ben più rilevante sia come dimensioni, sia per il livello della Pubblica Amministrazione interessata. A settembre 2015, infatti, il Ministero della Difesa ha firmato un accordo con Document Foundation per adottare la suite libera per ufficio LibreOffice e, soprattutto, Open Document Format (Odf), lo standard libero internazionale per documenti da ufficio. Passando a Odf il Ministero della Difesa sarebbe il primo, tra i grandi enti pubblici italiani, ad allinearsi completamente con le scelte più avanzate in termini di interoperabilità. In base all'accordo, i docenti volontari di Libreltalia guideranno all'apprendimento delle funzioni di LibreOffice 25 formatori professionisti del Ministero della Difesa, che a loro volta istruiranno a cascata prima altri formatori interni, e poi gli utenti finali. Nel corso di queste attività, che dovrebbero permettere l'adozione di LibreOffice al minor costo possibile, Ministero e Associazione Libreltalia creeranno un percorso di e-learning su LibreOffice con licenza Creative Commons, quindi riutilizzabile da chiunque. A differenza di altre iniziative precedenti l'ingresso ufficiale di Libre Office nel Ministero della Difesa ha le carte in regola per diventare una buona pratica da imitare in tante altre Pubbliche Amministrazioni italiane.

## TAURINUS X200, IL PORTATILE PIÙ LIBERO CHE C'È

A rigori di termini, utilizzare solo ed esclusivamente software libero è ancora molto difficile, anche per i suoi sostenitori più convinti. Il motivo è che, anche quando non si è costretti a utilizzare applicazioni disponibili solo su Windows o altri sistemi proprietari, si devono per forza usare chipset, schede grafiche o Bios con driver o software embedded proprietari.

Questo problema è presente soprattutto nei laptop recenti, molti dei quali contengono schede madre Intel con microcontroller Intel Me (Management Engine), in cui gira appunto firmware proprietario. Secondo la Free Software Foundation (Fsf), avere codice segreto come quello direttamente integrato nelle schede madri è una violazione grave dei diritti degli utenti, in quanto potrebbe permettere a terze parti di accedere al computer da remoto. A settembre 2015 però la stessa Fsf ha annunciato che i sub-notebook Taurinus X200 di Libiquity, basati sui Lenovo ThinkPad X200, meritano a pieni voti la sua certificazione RyF (Respect your Freedom), riservata appunto ai computer liberi da software proprietario.

Il Taurinus X200, infatti, oltre alla distribuzione Trisquel Linux (<https://trisquel.info/it>) derivata da Ubuntu, è preconfigurato con driver grafici liberi e soprattutto con il sistema Libreboot ([http://](http://libreboot.org)

libreboot.org). Quest'ultimo, insieme ad altro software sviluppato da Libiquity sostituisce completamente il firmware proprietario dell'X200, incluso quello Intel Me. Essendo l'unico laptop con certificato RYF che è anche privo di quel firmware, il Taurinus X200 si aggiudica quindi il titolo di "laptop più libero del pianeta".

