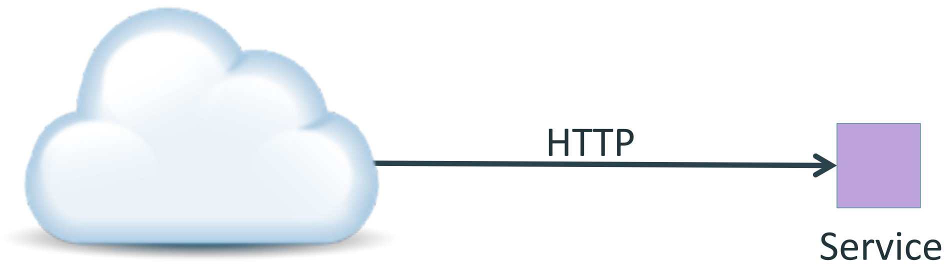
Why Does CA Platform Use OpenShift?

Mar 2015

technologies

The Problem

Let’s consider an application with a back-end web service.



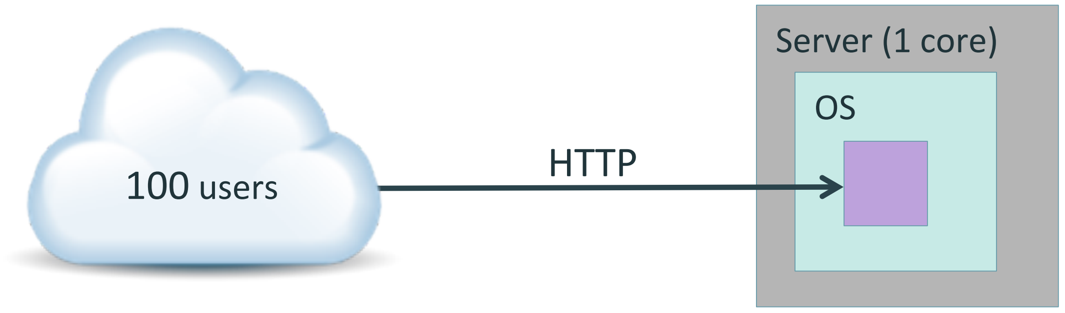
The service could be Tomcat serving HTML, Jetty serving OData, Node.js serving plain REST APIs, an instance of Ruby on Rails — doesn’t matter here.

How can we deploy this service?

The Problem

Let's say 1 instance of that service can handle **100 users** and requires **1 CPU core**.

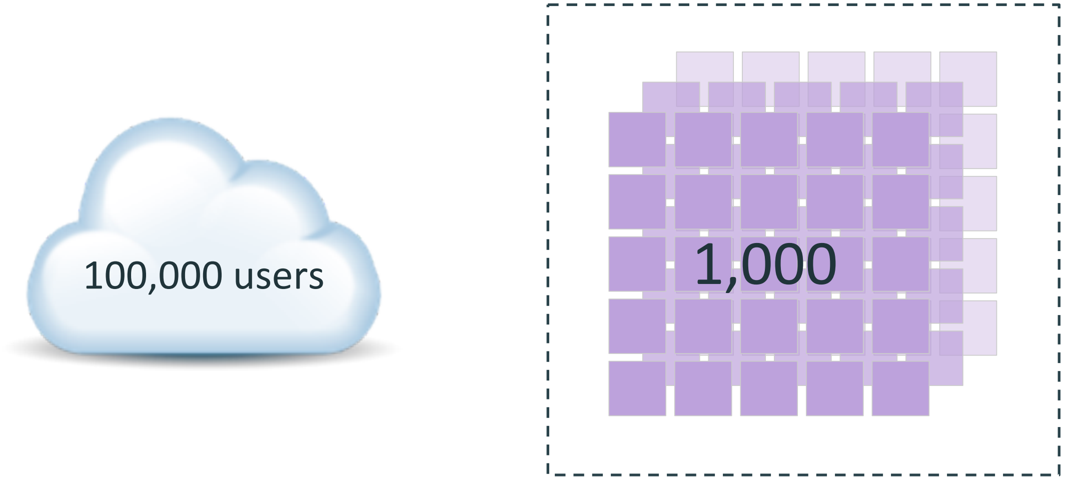
And let's say we have 100 users. We could simply run 1 service on 1 physical server (or VM) with 1 core. No problem.



But what if we have **100,000** users?

The Problem

With 100,000 users, we need 1,000 service instances, which need at least 1,000 cores.



How can we deploy an application with 1,000 Tomcats or 1,000 RoR instances?

That's a complex problem, and good solutions must meet many criteria. But here are 4 key criteria for CA Platform.

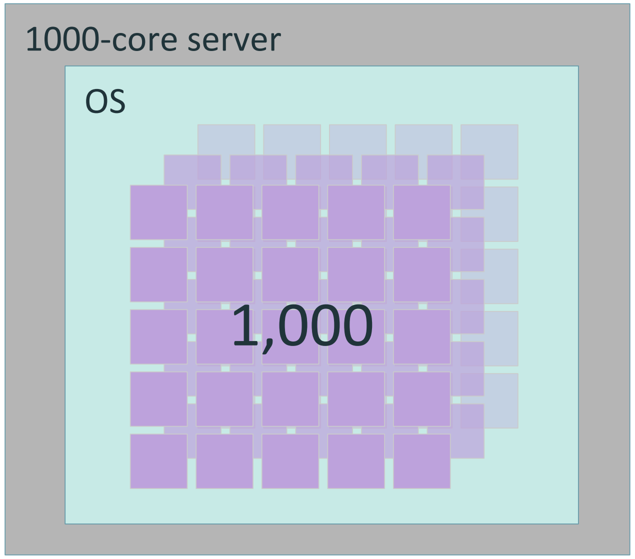
Four Key Criteria

Capex	We’re going large-scale because we are delivering SaaS. SaaS profitability calls for the most cost-efficient **hardware, software,** and **operations labor** that can do the job.
Opex	
Isolation	To achieve sufficient reliability, we need **resource isolation** between service instances — we can’t let one misbehaving instance take down hundreds of others.
Flexibility	We need to efficiently **add, remove, grow**, and **shrink** services. In real life, that hypothetical set of 1,000 service instances isn’t static, isn’t uniform, and isn’t the only set of services.

So what are some approaches to this problem?

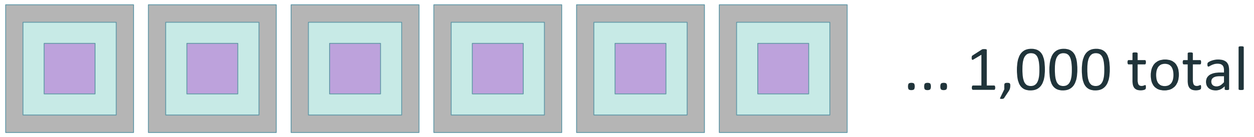
Bad Approach: Humongous Servers

Putting all 1,000 services on a small number of very large servers is a non-starter. Biggest problem in general: large servers are **dramatically less cost-effective** than small servers. Large servers only make sense for specialized workloads that can't run on small servers.



Capex	Awful
Opex	OK but doesn't matter
Isolation	Awful
Flexibility	Awful

Bad Approach: Horde of Tiny Servers

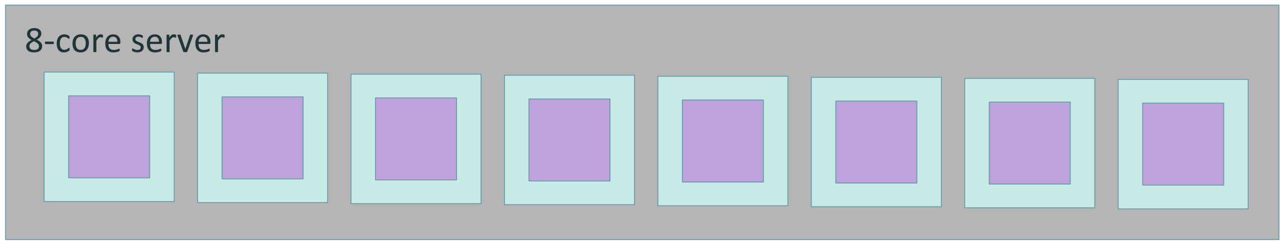


Putting each service on its own just-big-enough server doesn't work either. Small servers might be cost-efficient in cores-per-dollar, but opex and flexibility are unacceptable.

Capex	Bad. Cost-optimal server size is usually bigger than 1 service. Some costs (e.g. OS licensing) scale with the number of servers.
Opex	Awful. We have 1,000 independent OS instances to manage.
Isolation	Good (actually, total overkill for web apps).
Flexibility	Awful especially if our service outgrows our server!

Common Approach: Virtualization

Let's say the cost-optimal server has 8 cores — enough raw capacity for 8 services (ignoring overhead). To isolate those 8 services from each other, we need some way to subdivide our physical servers. **Enter virtualization.**



× 125 ...?

Capex	Just OK. 1,000 OS instances impose significant additional overhead — we actually need **way more** than 125 servers.
Opex	Still awful. We still have **1,000 independent OS instances** to manage (plus a ton of virtualization software to buy).
Isolation	Good (still overkill).
Flexibility	OK. We can change deployment sizing virtually (not dynamically)

Introduction to OS Container Technology

OS Container Technology

What if we could keep the flexibility of virtualization without the OS overhead? Especially for mostly-uniform workloads that don't need total isolation?

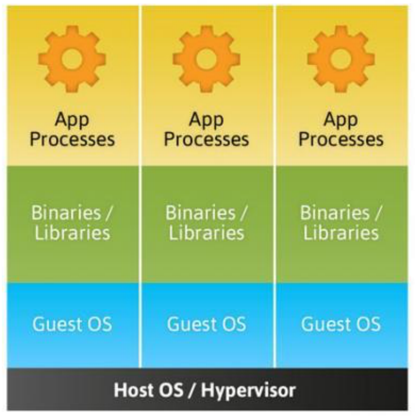
This observation led to the development of "lightweight virtualization" where the "VMs" are **partially isolated** partitions of the **same underlying OS**.

In full virtualization, one partition could be Windows, another Linux. With OS containers, both partitions expose the same OS instance.

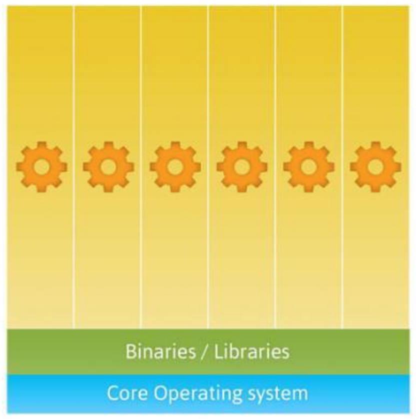
Virtualization is **external** to the guest OS. Containers are features provided **by** an OS.

From Hypervisors to Containers

- Hypervisors
  - Entire computing stack virtualized
  - Full OS image for each VM (high overhead)



- Containers
  - Shared OS kernel (more efficient)
  - All HW containers on top of single Linux instance
  - Most PaaS systems have a container foundation



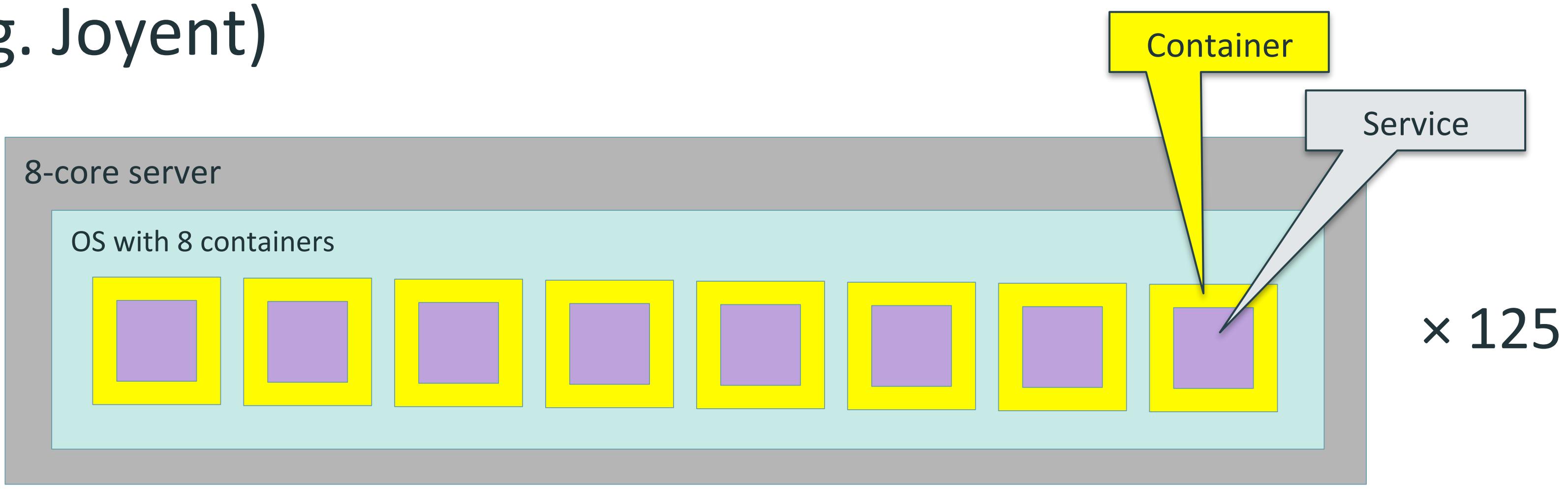
➡ Small, neat capsule containing your app

More apps on a single server

Source: linuxjournal.com

Back to the Main Story

Common Approach: OS Containers (E.g. Joyent)



Capex	Good. Cost-optimal hardware. Minimum OS overhead.
Opex	OK. Reasonable number of OS instances, **but we haven't addressed** how to manage thousands of services or how to route application traffic. This entails lots of custom automation.
Isolation	Good enough for web services.
Flexibility	OK. Containers are easier to manage than full VMs, but are still individually-configurable objects that must be externally automated or manually maintained.

Toward a Solution

- With OS containers, Capex and isolation are good.
- How can we extend that solution to fully-automated container management with sufficient flexibility for complex large-scale SaaS applications?

Introducing OpenShift

OpenShift is an open-source technology stack from RedHat with several major features. For this discussion, we care about two big aspects of OpenShift:

- **Ultra-lightweight containers** — OpenShift 2.x combines two standard Linux features (cgroups and SELinux) to define lightly-isolated containers, called **gears,** that impose essentially zero overhead versus running on the bare OS. Gears provide just enough isolation for web services.

- **Distributed application model** — OpenShift allocates gears to applications from a pool of gears (running on a pool of OS instances). The number of gears assigned to an application can change dynamically in response to manual commands or automatic load-driven scaling decisions. OpenShift automates a network of HTTP proxies so apps don't have to worry about which gears are running which services.

OpenShift-Based Approach to the Core Problem

Capex	Good. Cost-optimal hardware. Minimum OS (and virtualization) overhead.
Opex	Good. Reasonable number of uniform OS instances to manage. OpenShift fully automates the containers, service deployment & service traffic routing.
Isolation	Good enough for web services.
Flexibility	Good. All the dynamic large-scale complexity is software-defined and fully automated.

Why does CA Platform use OpenShift? **Because it is the only scalable application framework that fully delivers on all 4 key criteria.**

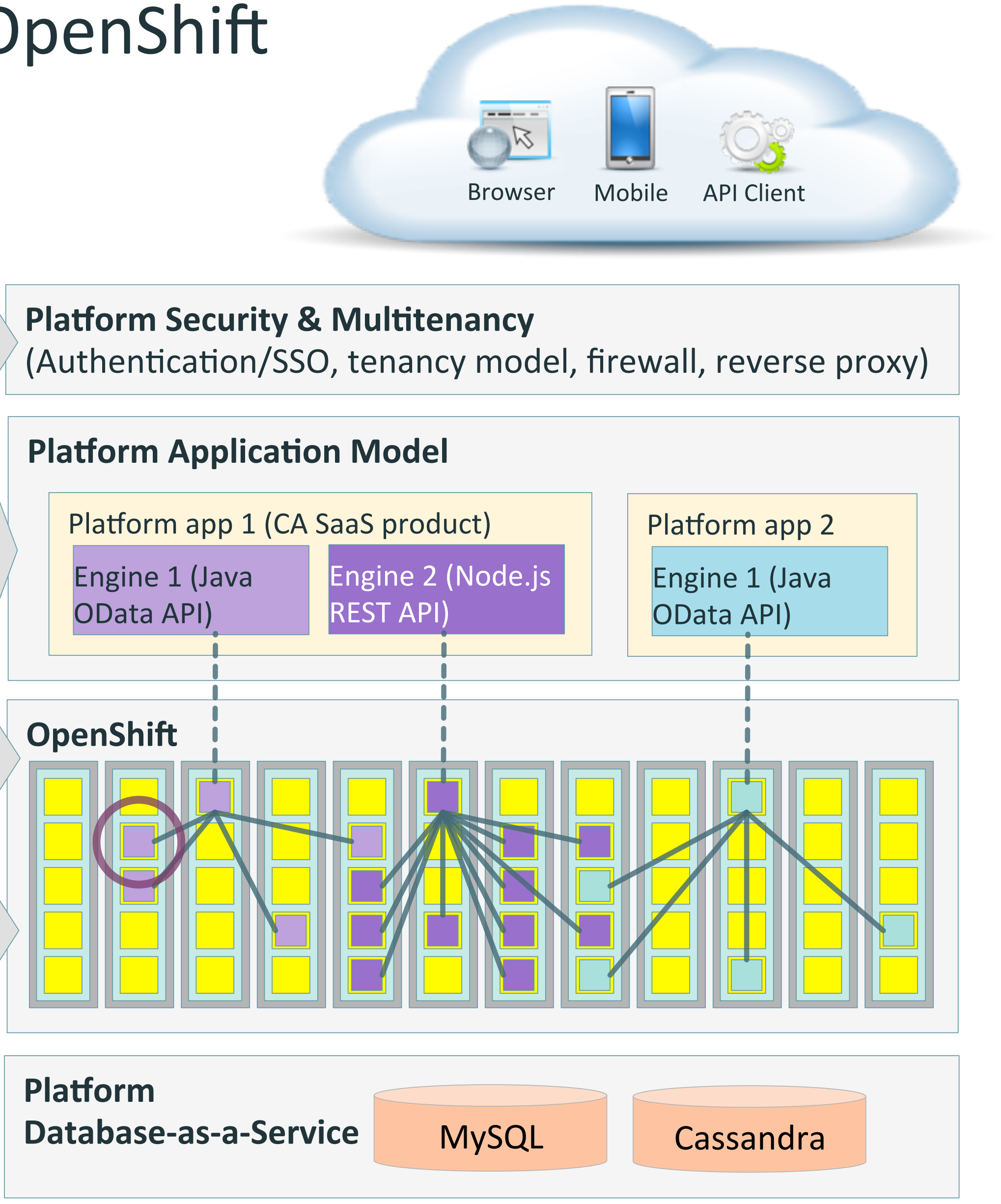
How CA Platform Uses OpenShift

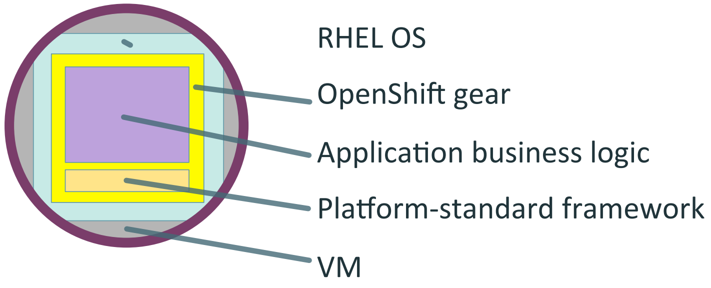
CA Platform leverages CA security products (e.g. CA API Management) for enterprise-grade authentication, security, and multitenancy.

CA Platform provides a standardized model. Apps model business logic as platform **engines** built on platform-provided app frameworks.
CA Platform is a PaaS that leverages OpenShift.

CA Platform uses OpenShift API to run an OpenShift app for each engine. OpenShift allocates gears and dynamically manages HTTP proxies.

OpenShift maintains a pool of gears on RHEL OS instances (running on IaaS or virtualized HW — making CA Platform highly portable).


Browser
Mobile
API Client
Platform Security & Multitenancy
(Authentication/SSO, tenancy model, firewall, reverse proxy)
Platform Application Model
Platform app 1 (CA SaaS product)
Engine 1 (Java OData API)
Engine 2 (Node.js REST API)
Platform app 2
Engine 1 (Java OData API)
OpenShift
Platform Database-as-a-Service
MySQL
Cassandra



CA Technologies

@CAinc

slideshare.net/CAinc

https://www.linkedin.com/company/ca-technologies

ca.com