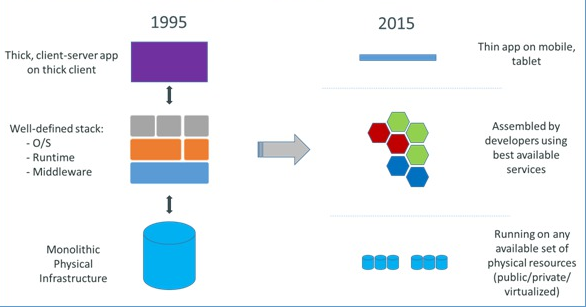


**Evolution of IT**

****

**What is Docker ?**

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

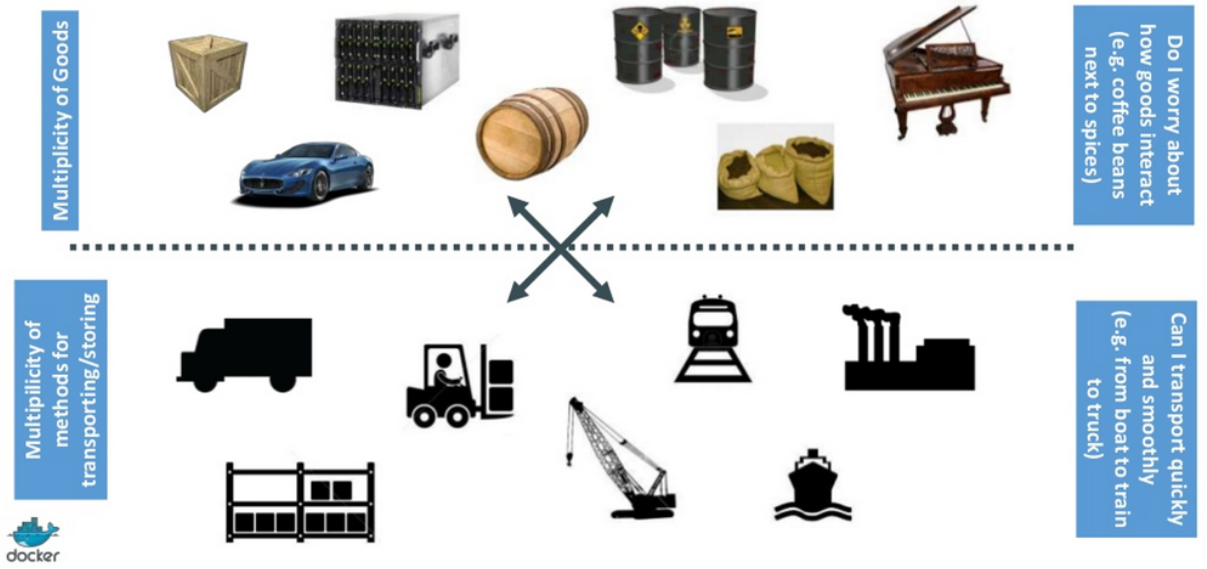
**Origin of Docker**

* Docker is a rewrite of similar code that currently powers the dotCloud PaaS
* Original version written in Python (like dotCloud PaaS), now written in Go
* It’s a young project , but with a huge community.

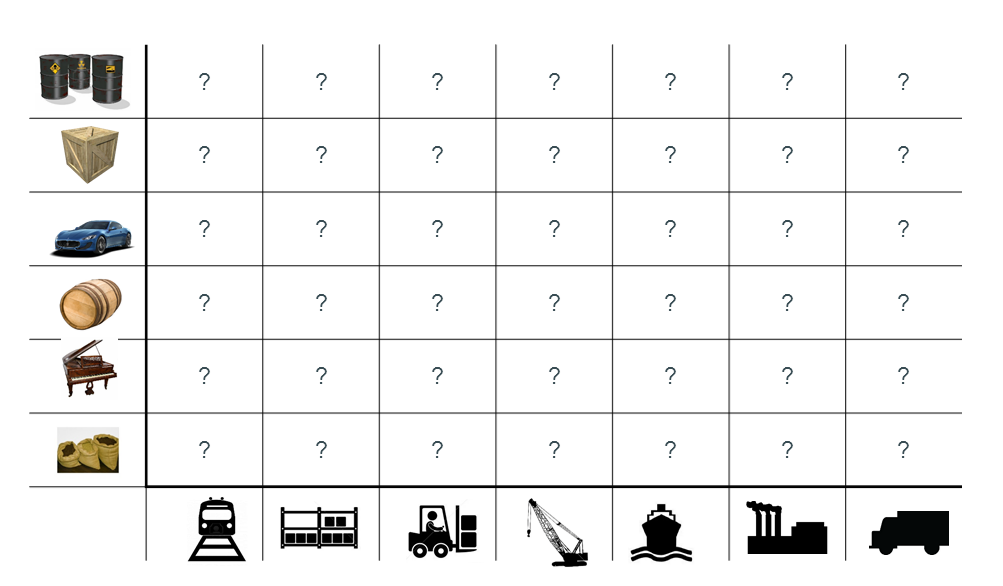
**Containers**

**Why Containers ?**

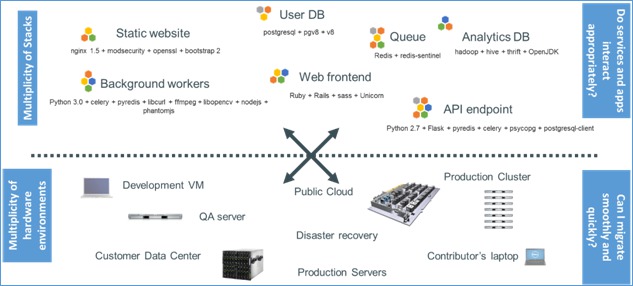
**Cargo Analogy**



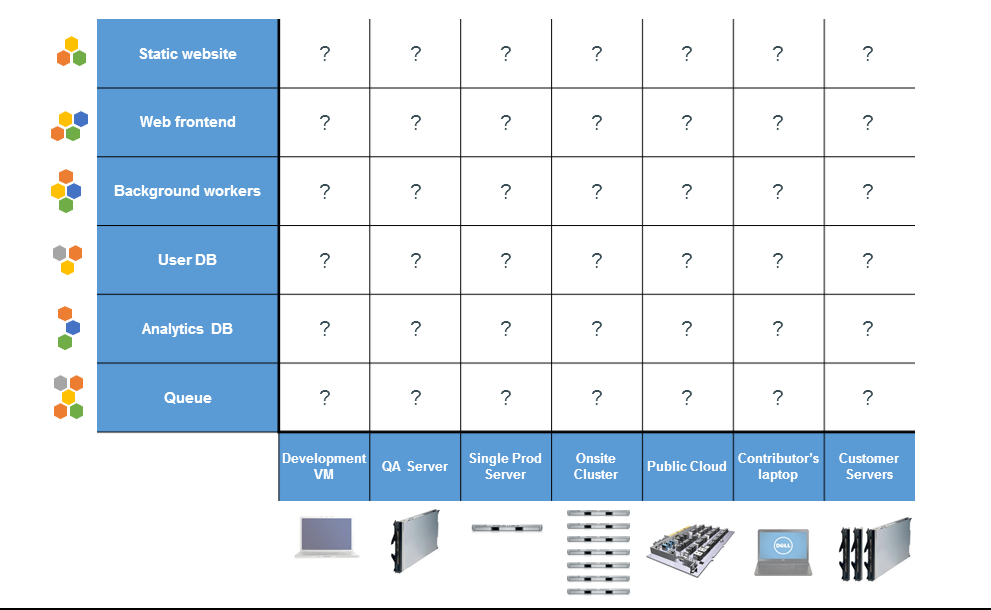
**The Matrix From Hell**

****

**Challenge of Multiple Environments**

****

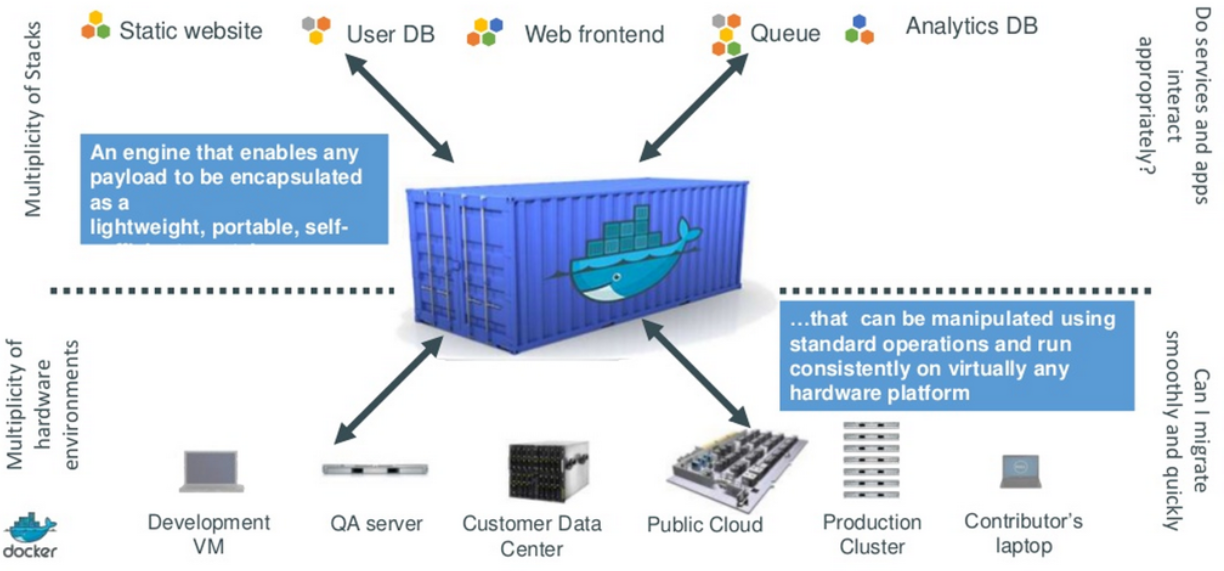
**The Matrix From Hell**

****

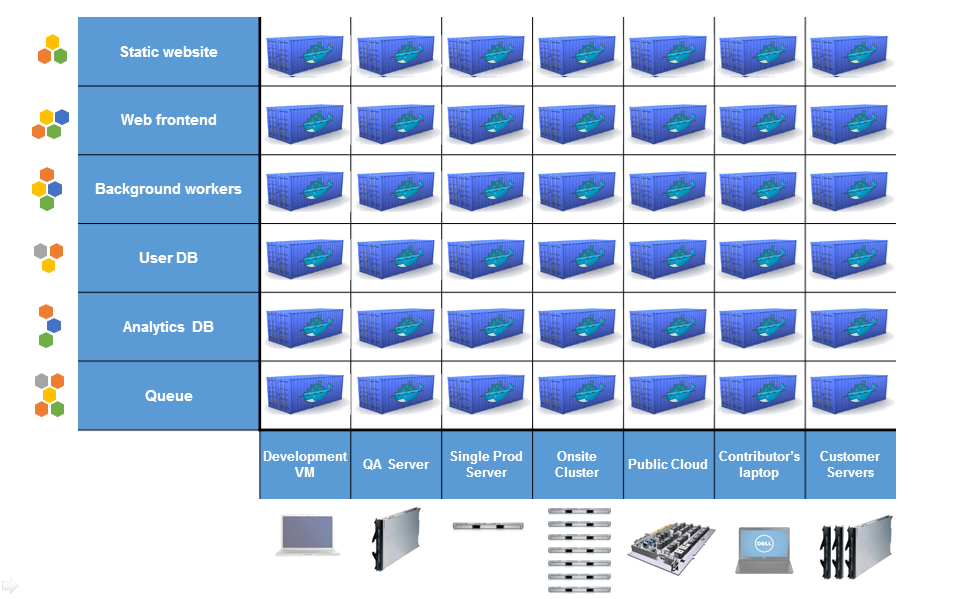
**Intermodal Shipping Container**



**Docker is a Code or Software Shipping Container**



**Docker eliminates the matrix from Hell**

****

* Build once…(finally) run anywhere\***(Why Developers care)**
  + A clean, safe, hygienic and portable runtime environment for your app.
  + No worries about missing dependencies, packages and other pain points during subsequent deployments.
  + Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
  + Automate testing, integration, packaging…anything you can script
  + Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
  + Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots? That’s the power of Docker
* Configure once…run anything **(Why DevOps care)**
  + Make the entire lifecycle more efficient, consistent, and repeatable
  + Increase the quality of code produced by developers.
  + Eliminate inconsistencies between development, test, production, and customer environments
  + Support segregation of duties
  + Significantly improves the speed and reliability of continuous deployment and continuous integration systems
  + Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs
* Speed: Boots in seconds
* Footprint: 100-1000 containers on one machine. Small disk requirements

**What is Container ?**

“Containers are to Virtual Machines as threads are to processes.”

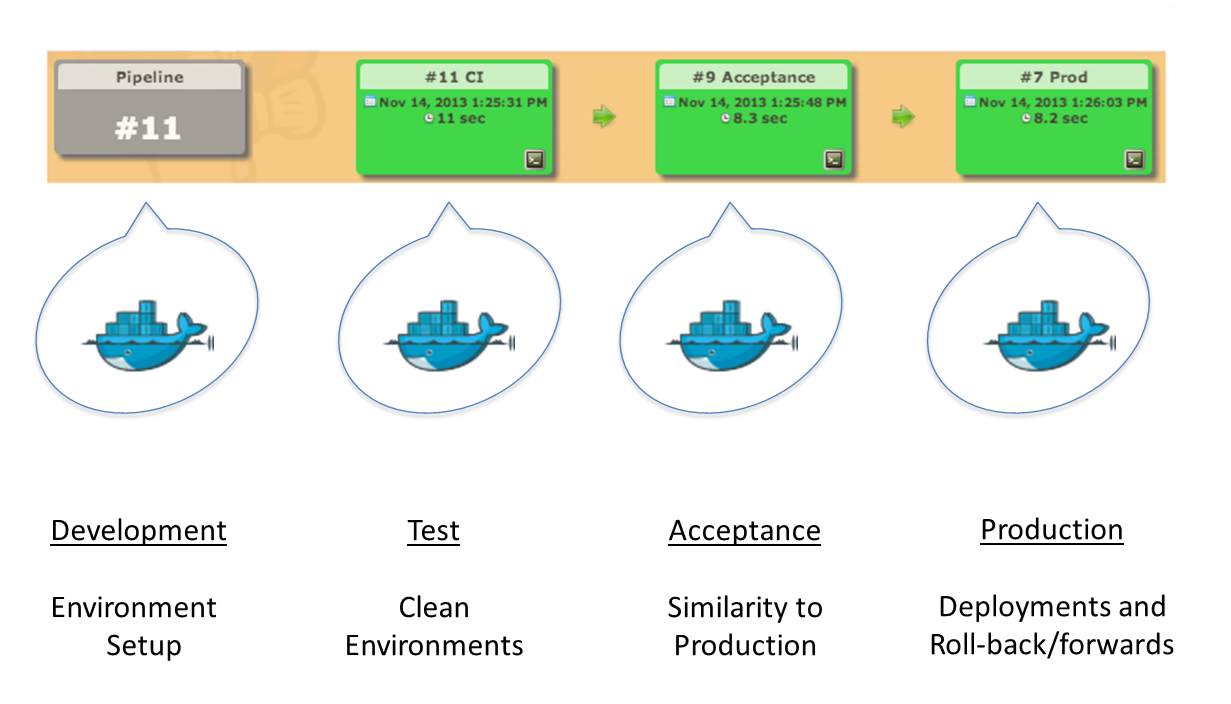
**What is container in docker?**

* LinuX Containers (LXC)
* Kernel namespaces (ipc, uts, mount, pid, network and user)
* Chroots (using pivot\_root)
* Apparmor and SELinux profiles
* Kernel capabilities
* Control groups (cgroups)
* AUFS or replacement in 0.7 version and later

**LinuX Containers (LCX)**

* Let’s your run a Linux system within another Linux system
* A container is a group of processes on a Linux box, put together is an isolated environment
* From the inside, it looks like a VM
* From the outside, it looks like normal processes

**Delivery Pipeline with Containers**

****

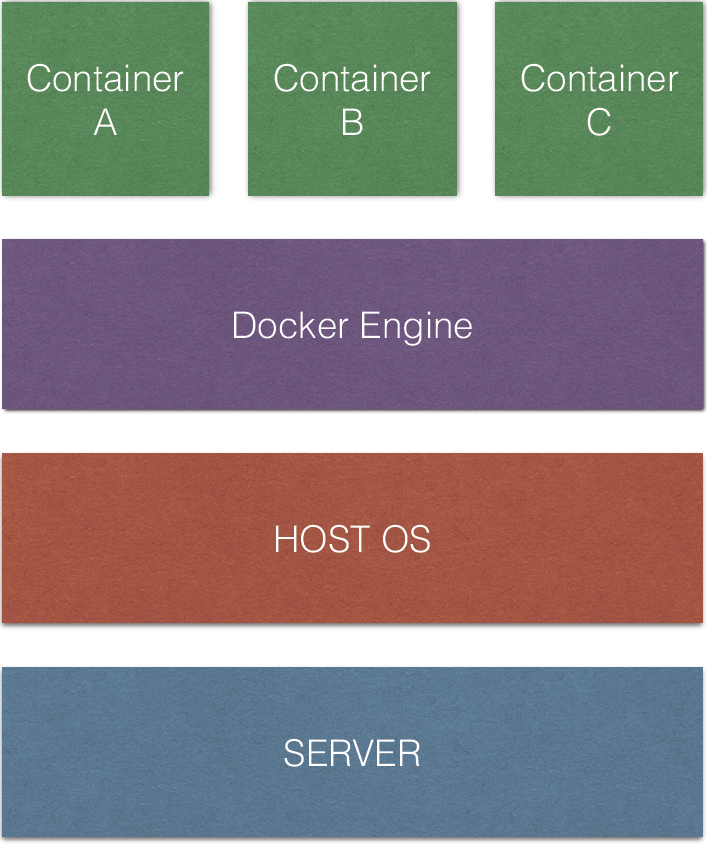
**Container Lifecycle**

* docker run - creates a container.
* docker stop - stops it.
* docker start - will start it again.
* docker restart - restarts a container.
* docker rm - deletes a container.
* docker attach - will connect to a running container.
* docker wait - blocks until container stops.

**Container Info**

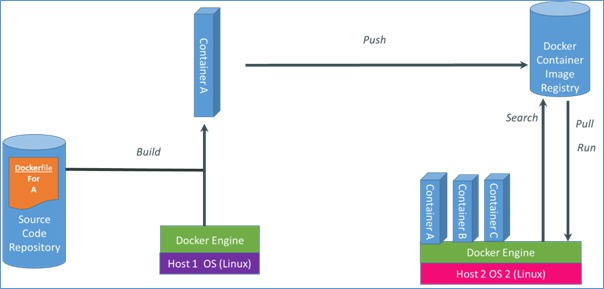
* docker ps - shows running containers.
* docker ps -a - shows running and stopped containers.
* docker inspect - looks at all the info on a container (including IP address).
* docker logs - gets logs from container.
* docker events - gets events from container.
* docker port - shows public facing port of container.
* docker top - shows running processes in container.

**Docker Engine**

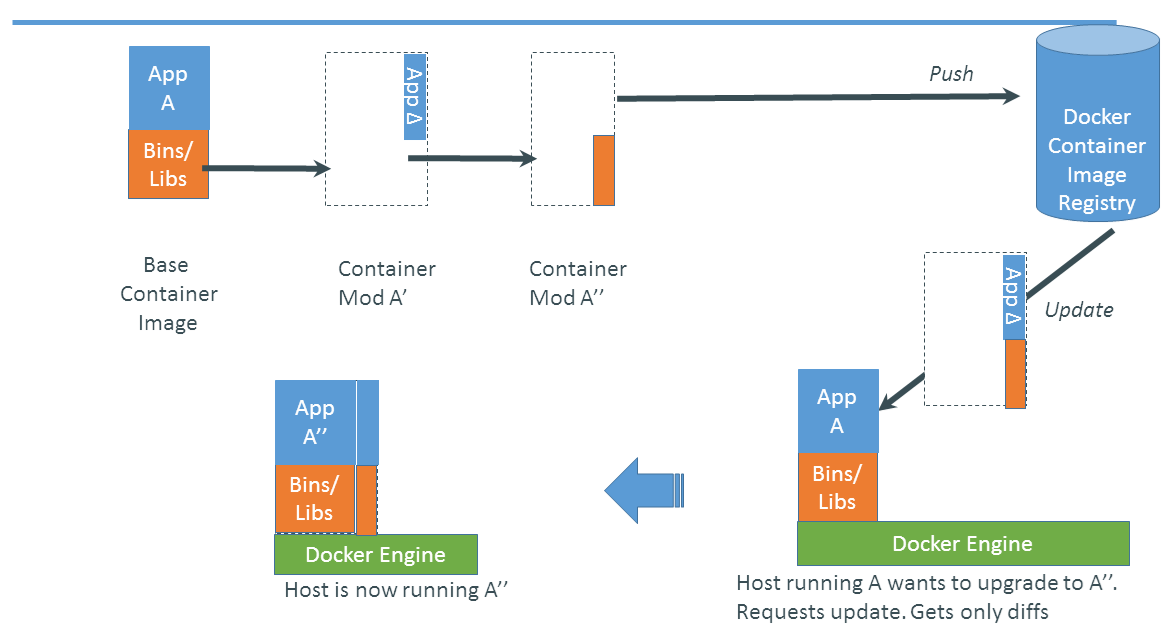


**Docker Functions**

**What are the basics of the Docker system?**

****

**Changes and Updates**

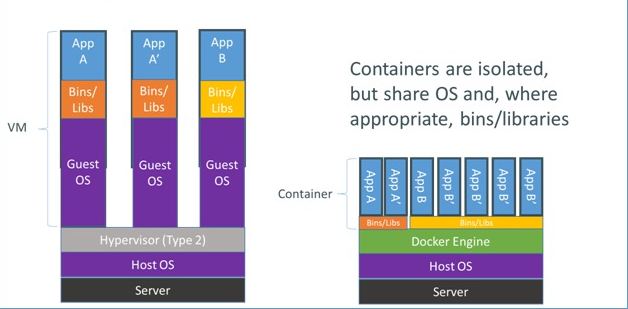
****

In a very simplistic way we can say following about docker functionality.

* It is based on existing technologies LXC containers, cgroups and AUFS
* There are
  + dockerfiles which are similar to source code and used to build the images.
  + Build process. Inherits an image, creates container, runs commands from Dockerfile inside and creates a new image.
  + New image is pushed into central repo - Docker Index. Central or Local
  + When container is started it will pull the relevant image, cache it locally and create container out of it. First time includes downloading, second time is typically around 0.100 Sec

Container will run on basically any Linux with Kernel 3.8+ or in any VM with such Linux. As well as natively on some cloud systems like OpenStack and at some service providers like dotCloud and DigitalOcean

**Docker Vs Virtual Machine**

****

Basically VM can do everything Docker does and more except:

* It is less portable. Most of the hypervisor and clouds have different VM formats despite the attempts to standardize them.
* More resources required to run VMs.
* Building VM will take anywhere between 5-30 minutes
* Startup time is typically around few minutes.

This makes creation on new VMs difficult and cumbersome which in turn will create the situation where developers try to avoid recreation of VMs as much as possible

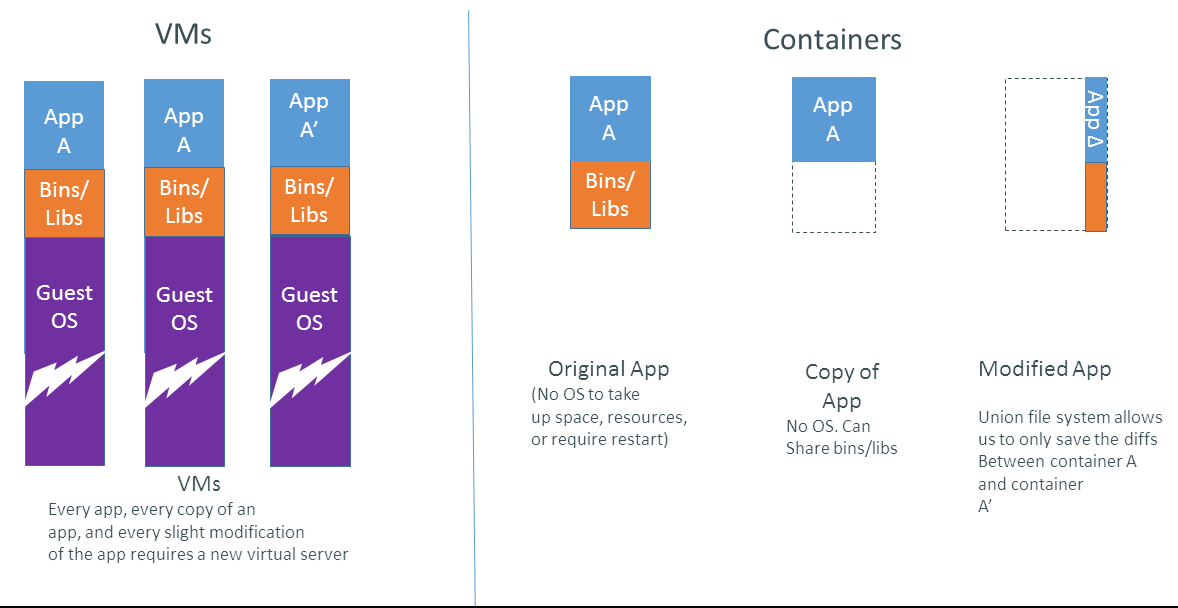
**Docker Vs Vagrant**

Vagrant is a virtual machine manager, it allows you to script the virtual machine configuration as well as the provisioning. However, it is still a virtual machine depending on Virtual Box (or others) with a huge overhead. It requires you to have a hard drive file that can be huge, it takes a lot of ram, and performance can be not very good.

Docker on the other hand uses kernel cgroup and namespacing via lxc. It means that you are using the same kernel as the host and the same file system. You can use Dockerfile with the docker build command in order to handle the provisioning and configuration of your container. You have example at docs.docker.io on how to make your Dockerfile, it is very intuitive.

| **Feature** | **Docker** | **Vagrant** |
| --- | --- | --- |
| Virtualization Type | VE | VM |
| Guaranteed Resources at hardware level? | No | Yes |
| Supported OS Platforms | Linux only | Linux, Unix, Windows |
| Startup time for created machine | A few seconds | A few minutes |
| Isolation level for created virtual systems | Partial | Full |
| Weight of the created virtual systems1 | Very lightweight | Heavy, but still better than full VM |
| Other Advantages | Quick, easy to learn | Integration with CM tools |

**Why are Docker containers lightweight?**

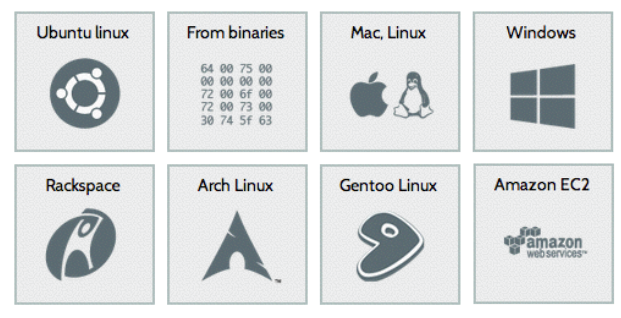
****

**In the first 8 months**

* >200,000 pulls
* >7,500 github stars
* >200 significant contributors
* >200 projects built on top of docker
  + UIs, mini-PaaS, Remote Desktop….
* 1000’s of Dockerized applications
  + Memcached, Redis, Node.js…and Hadoop
* Integration in Jenkins, Travis, Chef, Puppet, Vagrant and OpenStack
* Meetups arranged around the world…with organizations like Ebay, Cloudflare, Yandex, and Rackspace presenting on their use of Docker

**Installations of Docker**

“We can Install Docker on the listed platforms”



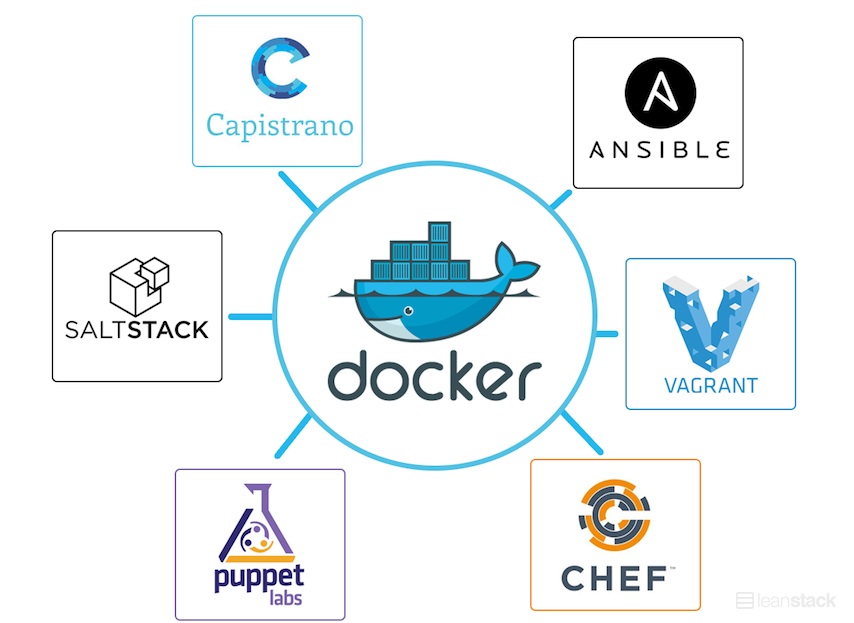
**Docker Integrations and Hosting(Ecosystem Support)**

****

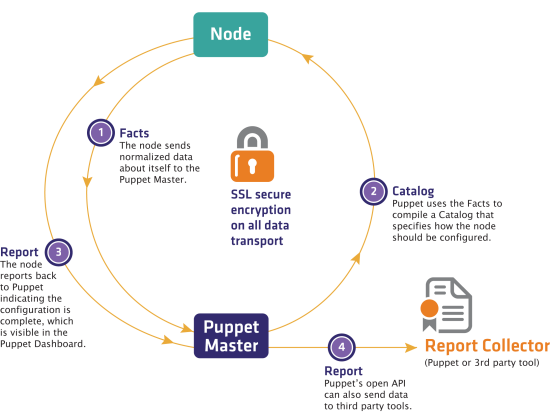
**Use Cases**

* Ted Dziuba on the Use of Docker for Continuous Integration at Ebay Now
* <https://speakerdeck.com/teddziuba/docker-at-ebay>
* <http://www.youtube.com/watch?feature=player_embedded&v=0Hi0W4gX--4>
* Sasha Klizhentas on use of Docker at Mailgun/Rackspace
* <http://www.youtube.com/watch?feature=player_embedded&v=CMC3xdAo9RI>
* Sebastien Pahl on use of Docker at CloudFlare
* <http://www.youtube.com/watch?feature=player_embedded&v=-Lj3jt_-3r0>
* Cambridge HealthCare
* <http://blog.howareyou.com/post/62157486858/continuous-delivery-with-docker-and-jenkins-part-i>
* Red Hat Openshift and Docker
* <https://www.openshift.com/blogs/technical-thoughts-on-openshift-and-docker>

**Docker in DevOps Ecosystem**

****

**Docker and Puppet/Chef/Ansible**

****

Docker is not going to replace puppet/chef/ansible, it will be added to the mix. Puppet/chef are good to manage underling infrastructure and VMs are very important for building clouds.

**Docker Features**

Docker is not a replacement for lxc. "lxc" refers to capabilities of the linux kernel (specifically namespaces and control groups) which allow sandboxing processes from one another, and controlling their resource allocations.

On top of this low-level foundation of kernel features, Docker offers a high-level tool with several powerful functionalities:

* **Portable deployment across machines.** Docker defines a format for bundling an application and all its dependencies into a single object which can be transferred to any docker-enabled machine, and executed there with the guarantee that the execution environment exposed to the application will be the same. Lxc implements process sandboxing, which is an important pre-requisite for portable deployment, but that alone is not enough for portable deployment. If you sent me a copy of your application installed in a custom lxc configuration, it would almost certainly not run on my machine the way it does on yours, because it is tied to your machine's specific configuration: networking, storage, logging, distro, etc. Docker defines an abstraction for these machine-specific settings, so that the exact same docker container can run - unchanged - on many different machines, with many different configurations.
* ***Application-centric*.** Docker is optimized for the deployment of *applications*, as opposed to machines. This is reflected in its API, user interface, design philosophy and documentation. By contrast, the lxc helper scripts focus on containers as lightweight machines - basically servers that boot faster and need less ram. We think there's more to containers than just that.
* ***Automatic build*.** Docker includes a tool for developers to automatically assemble a container from their source code, with full control over application dependencies, build tools, packaging etc. They are free to use make, maven, chef, puppet, salt, debian packages, rpms, source tarballs, or any combination of the above,*regardless of the configuration of the machines*.
* ***Versioning*.** Docker includes git-like capabilities for tracking successive versions of a container, inspecting the diff between versions, committing new versions, rolling back etc. The history also includes *how* a container was assembled and by whom, so you get full traceability from the production server all the way back to the upstream developer. Docker also implements incremental uploads and downloads, similar to "git pull", so new versions of a container can be transferred by only sending diffs.
* ***Component re-use*.** Any container can be used as an "base image" to create more specialized components. This can be done manually or as part of an automated build. For example you can prepare the ideal python environment, and use it as a base for 10 different applications. Your ideal postgresql setup can be re-used for all your future projects. And so on.
* ***Sharing*.** Docker has access to a public registry ([http://index.docker.io](http://index.docker.io/)) where thousands of people have uploaded useful containers: anything from redis, couchdb, postgres to irc bouncers to rails app servers to hadoop to base images for various distros. The registry also includes an official "standard library" of useful containers maintained by the docker team. The registry itself is open-source, so anyone can deploy their own registry to store and transfer private containers, for internal server deployments for example.
* ***Tool ecosystem*.** Docker defines an API for automating and customizing the creation and deployment of containers. There are a huge number of tools integrating with docker to extend its capabilities. PaaS-like deployment (Dokku, Deis, Flynn), multi-node orchestration (maestro, salt, mesos, openstack nova), management dashboards (docker-ui, openstack horizon, shipyard), configuration management (chef, puppet), continuous integration (jenkins, strider, travis), etc. Docker is rapidly establishing itself as the standard for container-based tooling.

**Licensing**

Its open source

**Pros:**

Docker containers are very lightweight in natures as they don’t have their own OS.

* No need of Hypervisor.
* Docker can run many containers on one VM with very little overhead.
* Docker can start containers almost as quickly as starting a process
* Docker containers you simply package up all the re­quire­ments for a given ap­pli­ca­tion and nothing will interfere with anything else.
* Containers allow you to sandbox your ap­pli­ca­tions for providing security.
* Docker makes it easy to limit the CPU and memory available to an ap­pli­ca­tion.
* Upgrading or down­grad­ing of ap­pli­ca­tion versions is easy.
* Easily remove or add ap­pli­ca­tions by building or destroying containers.
* Docker is doing well with cross-container communications by using env variables.

**Cons:**

* Docker is compatible with linux only and do not support any other OS.
* need to consider how to manage docker processes with a system process manager
* Backing up the data stored in your ap­pli­ca­tion databases still requires a robust con­fig­u­ra­tion and backup strategy.
* Working out how to wire up containers with links and do service discovery requires some planning and careful container creation.
* On a larger scale, or in complicated toplogies, with lots of links and containers, it might make sense to use some orchestration tools on top of Docker to provide some high-level deployment management.
* In very dynamic and elastic architectures, the use of distributed services registry such as [Zookeeper](http://zookeeper.apache.org/), [etcd](https://github.com/coreos/etcd" \o "etcd) or [serf](http://www.serfdom.io/) must be considered.