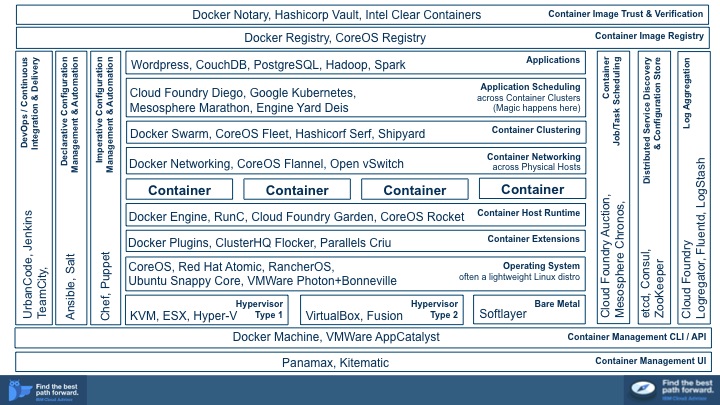
Evolving landscape of the (mostly) open source container ecosystem

Ever since the 2015 DockerCon in San Francisco, I am seeing an increasing number of questions from my cloud customers about various container technologies, how they fit together, and how they can be managed as part of the application development lifecycle. In the chart below I summarized my current understanding of the container ecosystem landscape. It is a pretty busy slide as the landscape is complex and rapidly evolving, so the rest of the post will provide an overview of the various parts of the stack and point to more information about the individual technologies and products. Although the chart is fairly comprehensive it doesn’t cover some experimental technologies like Docker on Windows.[](http://www.cloudswithcarl.com/wp-content/uploads/2015/07/Slide6.jpg)**Hypervisors Type 1, Type 2 and Bare Metal.** Most of the cloud providers today offer Docker running on a virtualized infrastructure (Type 1 hypervisors and/or low level paravirtualization) while developers often use Docker in a Type 2 Hypervisor (e.g. VirtualBox). Docker on bare metal Linux is a superior option as it ensures better I/O performance and has potential for greater container density per physical node.

**Operating System.** While Docker and other container technologies are available for a variety of OSes including AIX and Windows, Linux is the defacto standard container operating system. Due to this close affinity between the two technologies, many Linux companies began to offer lightweight distributions designed specifically for running containers: Ubuntu Snappy Core and Red Hat Atomic are just two examples.

**Container Extensions.** Docker codebase is evolving rapidly however as of version 1.7 it is still missing a number of capabilities needed for enterprise production scenarios. Docker Plugins exist to help support emerging features that weren’t accepted by the Docker developers in the Engine codebase. For example, technologies like Flocker and Criu provide operating system and Docker extensions that enable live migration and other high availability features for Docker installations.

**Container Host Runtime.** Until recently there existed a concern about fragmentation in the Linux container management space with Rocket (rkt) appearing as a direct competitor to Docker. Following an announcement of the Open Container Format (OCF) at DockerCon 2015 and support from CoreOS team for RunC, a standard container management in OCF, there are fewer reasons to worry. Nonetheless, container runtimes are evolving very rapidly and it is still possible that developers will change their preferred approach for managing containers. For example Cloud Foundry Garden is tightly integrated into the rest of the Cloud Foundry architecture and shields developers who simply want to run an application from having to execute low level container management tasks.

**Container Networking.** Support for inter-container networking had to be completely rewritten for Docker v1.7 and network communication across Docker containers hosted on different network nodes is still experimental in that release. As this particular component of Docker is evolving other technologies have been used in production to fill the gap. Some Docker users adopted CoreOS Flannel while others rely on Open vSwitch. Concerns about Docker Networking are also delaying adoption of container clustering and application scheduling in the Docker ecosystem and are in turn helping to drive users to the Cloud Foundry stack.

**Container Clustering.** Developers want to be able to create groups of containers for load balancers, web servers, map reduce nodes, and other parallelizable workloads. Technologies like Docker Swarm, CoreOS Fleet, and Hashicorp Serf exist to fill this requirement but offer inconsistent feature sets. Further, it is unclear whether developers prefer to operate on low level container clustering primitives such as the ones available from Swarm, or to write declarative application deployment manifests as enabled by application scheduling technologies like Kubernetes.

**Application Scheduling.** Cloud Foundry is the most broadly adopted open source platform as a service (PaaS) and relies on manifests, which are declarative specifications for deploying an application, provisioning its dependencies and binding them together. More recently, Kubernetes emerged as a competing manifest format and a technology for scheduling an application across container clusters according to the manifest. This layer of the stack is also highly competitive with other vendors like Mesosphere and Engine Yard trying to establish their own application manifest format for developers.

**Imperative and Declarative Configuration Management & Automation.** As with traditional (non-container based) environments, developers need to be able to automate the steps related to configuration of complex landscapes. Chef and Puppet, both Ruby based domain specific languages (DSLs), offer a more programmatic, imperative style for definition of deployment configurations. On the other end of the spectrum, tools like Ansible and Salt support declarative formats for configuration management and provide a mechanism for specifying extensions to the format. Dockerfiles and Kubernetes deployment specifications overlap with some of the capabilities of the configuration management tools, so this space is evolving as vendors are clarifying the use cases for container-based application deployment.

**DevOps, Continuous Integration & Delivery.** Vendors and independent open source projects are extending traditional build tools with plugins to support Docker and other technologies in the container ecosystem. In contrast to traditional deployment processing, use of containers reduces the dependency of build pipelines on configuration management tools and instead relies more on interfaces to Cloud Foundry, Docker, or Kubernetes for deployment.

**Log Aggregation.** Docker does not offer a standard approach for aggregation of log messages across a network of containers and there is no clear marketplace leader for addressing this issue. However this is an important component for developers and is broadly supported in Cloud Foundry and other PaaSes.

**Distributed Service Discovery and Configuration Store.** Clustered application deployments face the problem of coordination across cluster members and various tools have been adopted by the ecosystem to provide a solution. For example Docker Swarm deployments ofter rely on Etcd or Zookeeper while Consul is common in the Hashcorp ecosystem.

**Container Job/Task Scheduling.** Operating at a lower level than fully featured application schedulers like Cloud Foundry Diego, Mesosphere Marathon, or Kubernetes, job schedulers like Mesosphere Chronos exist to execute transactional operations across networks. For example, a job might be to execute a shell script in guest containers distributed across a network.

**Container Host Management API / CLI.** Docker Machine and similar technologies like boot2docker have appeared to help with the problem of installing and configuring the Docker host across various operating system (including non-Linux ones) and different hosting providers. VMWare AppCatalyst serves a similar function for organizations using the VMWare hypervisor.

**Container Management UI.** Traditional CLI based approach for managing containers can be easily automated but is arguably error prone in the hands of novice administrators. Tools like Kitematic and Panamax offer a lower learning curve.

**Container Image Registry.** Sharing and collaborative development of images is one of the top benefits that can be realized by adopting container technologies. Docker and CoreOS registries are the underlying server side technologies that enable developers to store, share and describe container images with metadata like 5 star ratings, provenance, and more. Companies like IBM are active players in this space offering curated image registries, focusing on image maintenance, patching, and upgrades.

**Container Image Trust & Verification.** Public image registries like DockerHub have proven popular with developers but carry security risks. Studies have shown that over 30% of containers instantiated from DockerHub images have security vulnerabilities. Organizations concerned about trusting container images turned to various solutions that provide cryptographic guarantees about the image provenance implemented as cryptographic signing of images and changes to them by a trusted vendor.

This entry was posted in [Uncategorized](http://www.cloudswithcarl.com/?cat=1) on [July 24, 2015](http://www.cloudswithcarl.com/?p=202).