State of the Union: Containers – Part 1

An Educational Report and Market Analysis of Container Technologies

Mike Metral
Product Architect, Rackspace
mike.metral@rackspace.com
March 2015

Rev. 4

Table of Contents

1 Introduction

This report is intended to cover and educate on the various technologies emerging in and around the container space without deep diving too far into the intrinsic concepts and specifics of any one given technology. Specifically, this report should be seen as a means to get up-to-speed on what containers are as well as the architecture decisions that exist in adapting containers into both your infrastructure & toolbox. In addition, this report will highlight what container systems and utilities serve as the best solution depending on the use case, as well as which of these options compete and overlap with one another.

The container ecosystem is one of the fastest evolving trends in tech space that we've seen since the virtualized movement of the previous decade. As such, it should be made known that the findings and opinions expressed in this report are *extremely* time sensitive as a small time-span of merely 6 months can and has rapidly shifted what the community has come to think and consider as a viable option.

Lastly, a lot of the information detailed in this report has been a result of personal experience using the technologies, but this is not meant to take away from the influence of various community discussions, articles and blog posts, and accrediting the exact source has proven to not be fully possible because of the author's lack of not noting the source.

2 Overview

The hype and obsession of containers in the last two years becomes evident when you think of IT infrastructure from a traditional virtualization perspective and the implication that one can be utilizing their computing resources with an even further elasticity and capability.

In short, the virtualization we've all come to know and use today is possible because of the development and rise of the hypervisor.

> "So why does everyone love containers and Docker? James Bottomley, Parallels' CTO of server virtualization and a leading Linux kernel developer, explained to me that VM hypervisors, such as Hyper-V, KVM, and Xen, all are "based on emulating virtual hardware. That means they're fat in terms of system requirements."
>
> Containers, however, use shared operating systems. That means they are much more efficient than hypervisors in system resource terms. Instead of virtualizing hardware, containers rest on top of a single Linux instance [on the host]. This in turn means you can "leave behind

the useless 99.9% VM junk, leaving you with a small, neat capsule containing your application," said Bottomley.

Therefore, according to Bottomley, with a perfectly tuned container system, you can have as many as four-to-six times the number of server application instances as you can using Xen or KVM VMs on the same hardware" (Vaughn-Nichols 2014)[1].

3 Docker

3.1 Background

Container technology has been around for many years but its popularity really came to fruition through Docker: an open-source project primarily focused on automating the deployment of apps within containers.

Docker was released in early 2013 by Docker Inc., formerly known as dotCloud, and has seen a massive amount of community development & contribution and has amassed an immense following from folks in various roles ranging from developers, devops, CIO's and CTO's to name a few.

The success of the Docker project has even has gone as far to accumulate several millions of dollars in funding for Docker Inc. providing them with an estimated valuation of $200-400 million dollars at the time of this report[2].

Needless to say, Docker and the ecosystem being built around it, primarily the container orchestration & management tools, are providing a popular and open-source means for companies of all sizes to adapt to the scalable, dynamic and agile technologies that similarly power the infrastructure and applications at tech giants such as Facebook, Google and Amazon.

3.2 Overview

Docker is based on Linux Containers (LXC), which is a simplified virtualization environment that allows you to run multiple, isolated Linux systems on a single Linux host.

In a traditional VM, the instance is allocated its own set of resources that allows for true isolation on the host, but it comes with a heavy price in terms of the resources required to run it, not to mention the lack of portability of the VM across different virtualization platforms – think of the difficulty of trying to share a VM image across

[1] http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/
[2] http://www.forbes.com/sites/benkepes/2014/09/16/the-rumors-were-true-docker-funding-confirmed-and-40-million-enters-the-coffers/

Amazon AWS, Rackspace Cloud, Microsoft Azure and VirtualBox. Now think about the size of the image in addendum to the image format, and how transferring it from one platform to the other doesn't always prove to be the most time and cost effective – these two issues just begin to scratch the surface of what VM's impose today.

In a container, you'll get less isolation from other containers running on the same host, but they have a severely smaller resource footprint. This allows for far more usage of your resources – and because they're much lighter they can be instantiated much quicker.

Unfortunately, the one key functionality that a hypervisor provides *today* that a container does not is that you can co-locate different operating systems or kernels on the same platform. Therefore, if you're set on running instances of Windows alongside instances of Debian, Ubuntu, Red Hat etc., then containers aren't applicable to your use case. This is due to the fact that hypervisors abstract an entire machine; where as containers only abstract the physical host's operating system kernel. Nevertheless, as of October 2014, Microsoft and Docker announced a partnership to invest on enabling the Windows Server container in the Docker engine, in addition to other Docker support in the suite of Microsoft products at a future date.[3]

Circling back, in its simplest form, you can think of Docker as a wrapper for LXC. However, Docker provides much more functionality through a layer of abstraction and automation of LXC, as well as a high-level API and a booming ecosystem that is actively being built around it.

Docker's main features center on: providing ease of portability through its format & bundling capabilities, being optimized for the deployment of applications rather than machines, and its philosophy that components should be reusable including containers themselves, which could serve as a base image for other containers.

In addition to the core features of Docker, sharing Docker images through what is known as an image repository allows for the communal use of applications. These applications can be pre-fabricated & uploaded by others to facilitate their consumption. For example, one can easily download and run in a matter of seconds a Docker image of the latest MySQL Server, Redis, Apache, Golang Environment, OpenVPN Server etc. without having to go through the full setup & configuration process for each tool, let alone worry if it will function on your virtualization platform.

These features are possible because Docker provides the ability to easily and quickly snapshot your application and its OS components into a common image that can be

[3] http://azure.microsoft.com/blog/2014/10/15/new-windows-server-containers-and-azure-support-for-docker/

deployed on other hosts also running the Docker engine. This capability resonates in the technical community that is unfortunately familiar with a sea of different VM image formats and hypervisors that don't play well together without some heavy lifting, if at all.

In short, Docker is about being able to consistently deploy an application environment in an easy and reproducible manner. In doing so, Docker has started a forward progression of IT infrastructure and how we construct the applications that run on it to truly be very flexible in various aspects more than ever before.

3.3 Basic Concepts

- **Container**: The technology that allows the deployment and execution of an application and includes its dependencies, user files & settings and the operating system.
- **Docker Daemon/Engine/Server:** Responsible for managing and instantiating the containers.
- **Docker Command-line Client:** Allows a user to communicate and control the Docker server daemon.
- **Dockerfile**: A Dockerfile is a set of instructions to run over a base image. And Docker will build an image from this Dockerfile.
- **Docker Image**: Blueprint / template used to launch the Docker container.
- **Docker Index / Image Repository:** Registry of Docker images that can be browsed and downloaded. The public Docker image repository is located at https://registry.hub.docker.com, but a private, local version of it can be run if needed.

3.4 Operational Concepts

Recall that containers share a common operating system kernel. Therefore, to allow that each container have some form of isolation, proper resource allocation and that it be lightweight and fast, the following concepts allow for that:

- **cgroups (Container Groups)**: Kernel feature which accounts for and isolates the resource usage (CPU, memory, disk, I/O, network, etc.) of a collection of processes[4]
- **Namespaces**: Kernel feature that allows for a group of processes to be separated such that they cannot see resources in other groups[5]
- **Unionfs**: Filesystem service, which allows for actions to be done to a base image. By this method, layers are created and documented, such that each layer fully describes how to recreate an action. This strategy enables Docker's lightweight images, as only layer updates need to be propagated

[4] http://en.wikipedia.org/wiki/Cgroups
[5] http://en.wikipedia.org/wiki/Cgroups#NAMESPACE-ISOLATION

from one environment to another, much like the code repository system git operates.

3.5 Current Container Philosophy

Since Docker and the ecosystem is currently an emerging field, it is worth noting that containers are not intended to be a replacement for VM's or baremetal machines, let alone be applied to all use cases. Containers are certainly fulfilling a solution to the painful problem for developers, operations & devops folks with regards to the lifecycle of apps and managing their environments, but a greenfield stance must be accepted and held first and foremost.

Also, there is a lot of competition and overlap across several of the options and its hard to pinpoint which is better than the other. That being said, not all use cases can be adapted perfectly into containers today, particularly, stateful applications, but work in this space is being addressed by several technologies and should be in a much better stance at a future date.

This report intends to educate and outline recommendations of which technologies are currently the best option, but when deciding which to choose, the simplest answer is that there is no simple answer. All of these tools are not only young, but are also moving at a rapid pace and the scale & ease at which they clearly outline what they're meant to aid with, let alone which technologies they respectfully tend to interoperate with, is still to be determined. It is the author's opinion that throughout 2015, the community will make it known which technologies are meant to stick and which should be shelved.

4 Modern Container Operating Systems

With the success and popularity of containers made by Docker, modern operating systems have emerged embracing the container culture. These OS' tend to provide the minimal functionality required to deploy applications along with self-updating and healing properties which are different than standard OS's today. In doing so, these types of OS' have evolved the operating model users are accustomed to by moving away from deploying applications at the application layer to deploying applications inside containers operated by Docker. More simply put, applications can be thought of as self-contained binaries that can be moved around environments accordingly based on your requirements of QOS, policy, affinity, replication etc.

4.1 CoreOS

CoreOS is a new Linux distribution that has been architected to provide features needed to run modern infrastructure stacks via containers with a hands-off

approach to keeping the OS up-to-date much like the manner in which browsers receive updates. The strategies and architectures that influence CoreOS are similar to the mechanisms that allow companies like Google, Facebook and Twitter to run their services at scale with high resilience.

In addition to the self-updating nature of CoreOS, the real value lies in its flagship products:

- **etcd**: A highly-available key value store for shared configuration and service discovery.
- **fleet**: A distributed init system that uses etcd as its manifest and systemd as its mechanism for instantiating units. Units are configuration files that describe the properties of the process that you'd like to run. One could think of it as an extension of systemd that operates at the cluster level instead of the machine level, so it functions as a simple orchestration system for systemd units across your cluster.

4.2 Red Hat Project Atomic

Red Hat's Project Atomic facilitates application-centric IT architecture by providing an end-to-end solution for deploying containerized applications quickly and reliably, with atomic update and rollback for application and host alike.

The core of Project Atomic is the Project Atomic Host. This is a lightweight operating system that has been assembled out of upstream RPM content. It is designed to run applications in Docker containers. Hosts based on Red Hat Enterprise Linux and Fedora are available now. Hosts based on CentOS will be available soon.

Project Atomic hosts inherit the full features and advantages of their base distributions. This includes systemd, which provides container-dependency management and fault recovery. It also includes journald, which provides secure aggregation and attribution of container logs.[6]

5 Service Registration & Discovery

Service registration & discovery is the centerpiece in systems that are both distributed and service oriented. Pinpointing how nodes in a cluster can register, discover and determine the necessary services available, and the means by which they can communicate with said service is the heart of the problem that service discovery technologies aim to solve.

In an active environment, containers are constantly being commissioned and decommissioned based on needs, standards, maintenance and failures. As you scale out your container architecture, keeping track of all the services in a static manner

[6] http://www.projectatomic.io/docs/introduction/

simply won't cut it, and a dynamic means of avoiding disturbance or disruption to your services is required.

The technologies described below are the current front-runners in the industry with regards to service registration, and in turn, service discovery.

5.1 Apache's "Zookeeper"

Zookeeper is a distributed configuration service, synchronization service, and naming registry for large distributed systems. ZooKeeper was a sub-project of Hadoop, but is now a top-level project in its own right.

It holds a Consistency+Partition Tolerance (CP) architecture, in the CAP theorem context, and uses the Zab protocol to coordinate changes across the cluster.

Many projects use Zookeeper, including: Hadoop's HBase, Yahoo and Rackspace's Email & Apps team.

5.2 CoreOS' "etcd"

Etcd is a distributed key/value store used for service discovery and shared configuration. It is aimed to be a simple implementation of the Raft consensus algorithm, particularly, with regards to agreements on election cycle & leader nomination, as well as manipulation to data.

It holds a Consistency+Partition Tolerance (CP) architecture, in the CAP theorem context, and chooses consistency over availability, specifically sequential consistency based on a quorum of nodes.

Many projects use etcd, including: Google's Kubernetes, Pivotal's Cloud Foundry, Rackspace's Mailgun, Apache Mesos & Mesosphere DCOS.[7]

5.3 Hashicorp's "Consul"

Consul is a tool for service discovery and configuration. It is distributed, highly available, and extremely scalable. Key features include:

- **Service Discovery** - Consul makes it simple for services to register themselves and to discover other services via a DNS or HTTP interface. External services such as SaaS providers can be registered as well.
- **Health Checking** - Health Checking enables Consul to quickly alert operators about any issues in a cluster. The integration with service discovery prevents routing traffic to unhealthy hosts and enables service level circuit breakers.

[7] https://coreos.com/blog/etcd-2.0-release-first-major-stable-release/

- **Key/Value Storage** - A flexible key/value store enables storing dynamic configuration, feature flagging, coordination, leader election and more. The simple HTTP API makes it easy to use anywhere.
- **Multi-Datacenter** - Consul is built to be datacenter aware, and can support any number of regions without complex configuration.[8]

It holds a Consistency+Partition Tolerance (CP) architecture, in the CAP theorem context, and implements the Raft protocol.

Public projects or companies using Consul were not found in a limited search performed.

5.4 Comparison

Org	Tool	Client / Server Arch	Primitive Key/Value Store	Basic Service Discovery	Adv. Service Discovery	Consistency	Language
Apache	Zookeeper	✓	✓	✓		✓	Java
Hashicorp	Consul	✓	✓		✓	✓	Go
CoreOS	Etcd	✓	✓	✓		✓	Go

Table 1 - Service Registration & Discovery Comparison

* Note: Basic vs. Advanced Service Discovery revolves around the notion that in an advanced setting, the technology has more service monitoring & health-checking capabilities.

In terms of which technology to use:

- Zookeeper has been around longer and thus is considered to be mature, but its dependency and usage of Java tends to deter many users. It is also worth noting many think that it may be starting to show some age and its adaptability into cloud infrastructures isn't the easiest process as its proven to be complicated, hard to work with and troubleshoot.
- Etcd & Consul are both new to the scene, but etcd is slightly older than Consul and the community seems to be favoring as well as using etcd far more.

Current Recommendation: etcd

6 Service/Resource Scheduling & Management

Service/Resource Schedulers and Managers on a cluster are tools that are aware of the underlying resources available, are capable of placing tasks across the cluster in

[8] https://github.com/hashicorp/consul

a specified and expected manner, abide by rules and constraints, and can offer the ability to execute tasks and services.

In the container ecosystem, the necessity of these tools becomes evident in situations such as the lifecycle of installing and maintaining the Docker engine as well as its dependencies, not to mention setting up the requirements needed by your applications, and most importantly servicing the container cluster orchestrator/management you decide to utilize.

To be clear, service/resource schedulers & managers do just that: they allocate the resources needed to execute a job, such as the execution of Docker containers. However, by themselves, these technologies should not be seen as options to create a PaaS offering or to solely orchestrate a set of containers. These tools serve a far more basic functionality in retrospect to the requirements actual Docker services require such as load balancing, failure recovery, deployment and scaling that are taken care of by an actual orchestrator sitting on top of your stack. Therefore, just because they can run any service or task from a simple hello world application or a much more complex stack across a cluster, to even instantiating a Docker container on said cluster, this does not mean that they should be in charge of full orchestration of containers.

The technologies described below are the current front-runners in the industry with regards to service/resource scheduling.

6.1 CoreOS' "Fleet"

Fleet is a distributed init system based on etcd for its manifest of tasks and systemd to do the task execution. It can be seen as an extension of systemd that operates at the cluster level and can be used to deploy a systemd unit file anywhere on the cluster.

Fleet can automatically reschedule units on machine failure, and can abide by properties such as ensuring that units are deployed together on the same machine, forbid colocation if necessary and deploy to specific machines based on metadata & attributes.

6.2 Apache's "Mesos"

Mesos is a distributed systems kernel. It is built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g., Hadoop, Spark, Kafka, Elastic Search) with API's for resource management and scheduling across entire datacenter and cloud environments.[9]

[9] http://mesos.apache.org/

Mesos is a cluster manager that provides efficient isolation of resources and is truly all about facilitating different types of workloads (a.k.a frameworks) to run top of it.

Some of the biggest technology companies such as HubSpot and Twitter are active users and advocates of Mesos.

6.3 Comparison

Org	Tool	Req. Supplied Membership	Basic Task Orchestration	Adv. Task Orchestration	10-100s of Hosts	1000s of Hosts	Language
CoreOS	Fleet	✓	✓		✓		Go
Apache	Mesos	✓		✓		✓	C++

Table 2 - Service/Resource Scheduling & Management Comparison

Org	Tool	Architecture	Resource Aware	Host Constraints	Host Balancing	Group Affinity	Anti-Affinity	Global Scheduling
CoreOS	Fleet	Monolithic		✓		✓	✓	✓
Apache	Mesos	Two-level	✓	✓	✓		✓	

Table 3 – Service/ Resource Scheduling & Management Functionality Comparison[10]

In terms of which technology to use:

- Fleet is new to the scene and has a decent community following, but it seems limited in its capabilities with regards to advanced scheduling and health metrics. It's also early in its development, and this could possibly accelerate with time.
- Mesos is the front-runner with some heavy names utilizing it today in their infrastructure. Also, Mesosphere, the company that is commercializing Mesos and is a separate entity from Apache (the developer of Mesos) has currently started work on a Mesos framework to support Kubernetes and has gotten a good amount of traction.

Current Recommendation: Mesos

7 Container Cluster Orchestration & Management

[10] http://gabrtv.github.io/deis-qconsf-2014/#/22

Orchestrating & managing a cluster of Docker containers is an emerging trend that is not only very competitive with many companies, but one that is evolving at a rapid pace. Many options currently exist with various feature sets, and the race to be the front-runner is in full swing.

Below is a list of the notable open source container orchestration engines & managers, along with a summary of what they each aim to achieve.

It is highly recommended that your respective teams perform the proper analysis of which option to choose given the use cases you intend to fulfill, as well as the scale you wish to operate.

7.1 Docker's "Compose"

Compose, previously known as "Fig" prior to its acquisition, is a simple orchestration framework really aimed to allow the definition of a fast, isolated development environment for Docker containers.

Its sweet spot really lies in apps that revolve around a single-purpose server that could easily scale out based on the notion that architectural complexity is not a requirement. Development environments, which tend to be an all-in-one baked in means of operation, obviously fits well into this requirement and thus makes Fig shine as a viable option.

Based on use cases, something as simple as Fig may be all one needs. However, because this is a space where a solution such as Fig has both limited capabilities and overhead, teams may and have decided to hand roll micro-solutions of this kind on their own for the sake of not taking on extra overhead in their stack.

Its reception in the community is notable, but the practicality of its usage and the lack of ability to create a long-term vision around it tend to minimize the actual legitimacy of adopting it as a container orchestration technology.

7.2 Prime Directive's "Flynn"

Prime Directive labels Flynn as "the product that ops provides to developers."[11] They believe that "ops should be a product team, not consultants" and that "Flynn is the single platform that ops can provide to developers to power production, testing, and development, freeing developers to focus."

Flynn is an open-source PaaS built from pluggable components that you can mix and match however you want. Out of the box, it looks a lot like a Heroku that you could self-host, but one that easily allows you to replace the pieces you want with whatever you need.

[11] https://flynn.io/

In regards to how Flynn differs from other PaaS' like Heroku, Cloud Foundry, Deis or Dokku, "the other PaaS technologies mainly focus on scaling a stateless app tier. They may run one or two persistent services for you, but for the most part you are on your own to figure that part out. Flynn is really trying to solve the state problems, which is pretty unique."[12]

It is worth mentioning that with regards to stateful management, particularly in databases, right now they support Postgres, but their goal is to have Flynn help you run the database services so you don't have to.

Sponsors and users of Flynn include but are not limited to: Coinbase, Shopify, and CentryLink.

7.3 OpDemand's "Deis"

Deis is an open-source PaaS that facilitates the deployment and management of apps. It is built on Docker and CoreOS (including etcd, fleet and the OS itself) to "provide a lightweight PaaS with a Heroku-inspired workflow."[13]

It can deploy any app or service that works in a Docker container and its structure mimics Heroku's 12-factor stateless methodology for how apps should be created and managed. Deis also leverages Heroku's Buildpacks and comes with out-of-the-box support for Ruby, Python, Node.js, Java, Clojure, Scala, Play, PHP, Perl, Dart and Go.

Much like Flynn, it too looks a lot like Heroku clone that you could self-host. However, Deis lacks persistent storage and state aware support for use cases such as databases, and rather depends on some 3rd party cloud database solution. In this regard, Flynn seems to be ahead of Deis as the front-runner in Heroku-like projects.

Users of Deis include small to medium businesses & tech companies, but no major companies have announced their use of it.

7.4 ClusterHQ's "Flocker"

Flocker is an open-source data volume and multi-host container manager that supports and works with Docker's Compose (a.k.a Fig) file format syntax. Where Docker naturally shines with applications such as frontend or API servers, which utilize shared storage and are replicated or made highly available in some capacity, Flocker's intention is to offer the same portability but for applications with systems

---

[12] http://www.centurylinklabs.com/interviews/what-is-flynn-an-open-source-docker-paas/
[13] http://deis.io/overview/

such as databases & messaging/queuing systems as state management in containers is still an incomplete feature that is missing in the community.

The sweet spot with Flocker seems to be centered on its data management features and self-proclaiming themselves as the leader in this area. However, though appearing as the front-runner in datastore-centric models, full support of data in many use cases is still a work in progress and operations aren't met without undergoing downtime of some capacity.

Flocker alleviates the issue of managing data for containers by utilizing ZFS as the underlying technology for the attached datastore containers used, with volume behaviors and such operated by ZFS itself. In addition to the ZFS properties, Flocker imposes a network proxy across all of the Flocker nodes to handle container linking, storage mapping and user interaction throughout the cluster.

7.5 Cloudsoft's "Clocker"

Clocker is an open source project that enables users to spin up Docker containers, without generating excess containers, in a cloud-agnostic manner. The project is built on top of Apache Brooklyn, a multi-cloud application, and management software.

Some features of Clocker are:

- Automatically create and manage multiple Docker hosts in cloud infrastructure
- Intelligent container placement, providing fault tolerance, easy scaling and better utilization of resources
- Use of any public or private cloud as the underlying infrastructure for Docker Hosts
- Deployment of existing Brooklyn/CAMP blueprints to Docker locations, without modifications

Brooklyn uses Apache jclouds, a cloud API agnostic library, to provision and configure secure communications (SSH) with cloud virtual machines. The Docker architecture provides 'containers' on 'host' machines. Brooklyn provisions cloud machines using jclouds and uses them as Docker hosts.

Brooklyn uses a Dockerfile which makes an SSH server available in each Docker container, after which it can be treated like any virtual machine. Brooklyn receives sensor data from the app, every docker host, every docker container as well as every software making up the app and can effect changes in each of these; enabling Brooklyn to manage distribution of the app across the Docker Cloud. (Shenoy, 2014)[14]

[14] http://www.infoq.com/news/2014/06/clocker

In short, Brooklyn is a platform that monitors and manages Docker containers using a YAML configuration known as blueprints for its instructions. Clocker then, is essentially a blueprint for Brooklyn with extra intelligence geared at configuring and managing Docker hosts & containers.

7.6 Mesosphere's "Marathon"

Marathon is a cluster-wide init and control system for services in cgroups or Docker containers. It requires and is based on Apache Mesos and the Chronos job scheduler framework. Where Mesos operates as the kernel for your datacenter, Marathon is aimed to be the cluster's init or upstart daemon. It has a UI and a REST API for managing and scheduling Mesos frameworks, including Docker containers.

Marathon is a *meta framework*: you can start other Mesos frameworks with it. It can launch anything that can be launched in a standard shell. In fact, you can even start other Marathon instances via Marathon.[15] Because it is a framework built on Mesos, it can be seen as a comparable model to Clocker which is itself a blueprint (analogously a framework) for Apache's Brooklyn.

Because of its flexibility, Marathon can be seen as a cluster-wide process supervisor, including operating as a private PaaS through functionality that includes service discovery, failure handling, as well as deployment and scalability.

As of version 0.19 (current version is at 0.8) containers have become first class citizens in Marathon, utilizing Deimos as the Docker containerizer, a.k.a a Docker plugin for Mesos. Using Deimos and being based on Mesos, the combination of the frameworks allows Marathon to become an orchestration & management layer for Docker containers and provides the key services and dependencies one would come to expect in these particular toolsets.

Many major companies are using Marathon including: Airbnb, eBay, Groupon, OpenTable, Paypal and Yelp.

7.7 Google's "Kubernetes"

Kubernetes is a system for managing containerized clusters applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications.

Specifically, Kubernetes:

- Uses Docker to package, instantiate, and run containerized applications.
- Establishes robust declarative primitives for maintaining the desired state requested by the user. Self-healing mechanisms, such as auto-restarting, re-

[15] https://github.com/mesosphere/marathon

scheduling, and replicating containers require active controllers, not just imperative orchestration.
- Is primarily targeted at applications comprised of multiple containers, such as elastic, distributed micro-services.
- Enables users to ask a cluster to run a set of containers
    - The system automatically chooses hosts to run those containers on using a scheduler that is policy-rich, topology-aware & workload-specific

Kubernetes builds upon a decade and a half of experience at Google running production workloads at scale, combined with best-of-breed ideas and practices from the community. It is written in Golang and is lightweight, modular, portable and extensible.[16]

Some of the concepts behind Kubernetes include:

- **Pods**: a way to collocate a group containers together with shared volumes, [or even more explicitly: a collocation of 1 or more containers that share a single IP address, multiple volumes and a single set of ports].
- **Replication Controllers:** a way to handle the lifecycle of pods. They ensure that a specified number of pods are running at any given time, by creating or killing pods as required.
- **Labels:** a way to organize and select groups of objects based on key/value pair.
- **Services:** a set of containers performing a common function with a single, stable name and address for a set of pods – They act like a basic load balancer.[17]

Being one of the hottest, if not *the* hottest, technologies in the Docker ecosystem right now has forced many folks in the community to compare it to Mesos, the leader in cluster oriented development & management for the past couple of years.

However, the two have their differences:

> With Mesos, there is a fair amount of overlap in terms of the basic vision, but the products are at quite different points in their lifecycle and have different sweet spots. Mesos is a distributed systems kernel that stitches together a lot of different machines into a logical computer. It was born for a world where you own a lot of physical resources to create a big static computing cluster. The great thing about it is that lots of modern scalable data processing applications run well on Mesos (Hadoop, Kafka, Spark) and it is nice because you can run them all on the same basic resource pool, along with your new age container packaged apps. It is somewhat more heavy weight than the

[16] https://github.com/GoogleCloudPlatform/kubernetes
[17] http://stackoverflow.com/questions/26705201/whats-the-difference-between-apaches-mesos-and-googles-kubernetes

> Kubernetes project, but is getting easier and easier to manage thanks to the work of folks like Mesosphere.
>
> Now what gets really interesting is that Mesos is currently being adapted to add a lot of the Kubernetes concepts and to support the Kubernetes API. So it will be a gateway to getting more capabilities for your Kubernetes app (high availability master, more advanced scheduling semantics, ability to scale to a very large number of nodes) if you need them, and is well suited to run production workloads. (Google, 2014)

Lastly, [some say] Kubernetes and Mesos [can be] a match made in heaven. Kubernetes enables the Pod, along with Labels for service discovery, load-balancing, and replication control. Mesos provides the fine-grained resource allocations for pods across nodes in a cluster, and facilitates resource sharing among Kubernetes and other frameworks running on the same cluster.[18] However, Mesos can easily be replaced by OpenStack and if you've bought into Openstack, then the dependency and usage of Meso can be elimaterd. To get back to the comparison, Kubernetes is an opinionated declarative model on how to address microservices, and Mesos is the layer that provides an imperative framework by which developers can define a scheduling policy in a programmatic fashion – when leveraged together, they provide a data-center with the ability to do both.

The main take-away for Kubernetes is that right now it is best fit for typical webapps & stateless applications and that it's in pre-production beta. However, Kubernetes is one of the most active and tracked projects on Github, so expect many changes in not only in its functionality, stability and supported use cases, but also in the amount of technologies working on be highly interoperable with Kubernetes.

7.8 One-Off's

7.8.1 Docker's "Swarm"

Swarm is a tier aimed to provide a common interface onto the many orchestration and scheduling frameworks available. It serves as a clustering and scheduling tool that optimizes the infrastructure based on requirements of the app and performance. Solomon Hykes, CTO of Docker, stated, "Docker will give devs a standard interface to all [orchestration tools] and [Swarm] is an ingredient of that standard interface. [It can be thought of as] the glue between Docker and orchestration backends."

Swarm is designed to provide a smooth Docker deployment workflow, working with some existing container workflow frameworks such as Deis, but flexible enough to yield to "heavyweight" deployment and resource management such as Mesos. It is said to be a very simple add-on to Docker. It currently does not provide all the

[18] https://github.com/mesosphere/kubernetes-mesos/blob/master/README.md

features to say something of the likes of Kubernetes and its usage and place in the ecosystem is still to be determined.

7.9 Comparison

Org	Tool	1 Host (nano)	10s of Hosts (micro)	100s of Hosts (medium)	1000s of Hosts (large)
Docker	**Compose**	✓			
Prime Directive	**Flynn**		✓		
OpDemand	**Deis**		✓		
ClusterHQ	**Flocker**		✓		
Cloudsoft	**Clocker**			✓	
Mesosphere	**Marathon**				✓
Google	**Kubernetes**				✓

Table 4 - Size Comparison of Container Orchestrators & Managers

Org	Tool	Cluster State Management	Monitor & Healing	Deploy Spec	Allows Docker Dependency & Arch. Mapping	Deploy Method	Language
Docker	**Compose**			Dockerfile + YAML manifest	✓	CLI	Python
Prime Directive	**Flynn**			Procfile, Heroku Buildpack		git push	Go
OpDemand	**Deis**			Dockerfile, Heroku Buildpack		git push	Go
ClusterHQ	**Flocker**	✓		Dockerfile + YAML manifest	✓	CLI	Python
Cloudsoft	**Clocker**	✓	✓	Apache Brooklyn YAML Blueprint + Dockerfile	✓	API / Web	Java
Mesosphere	**Marathon**	✓	✓	JSON	✓	API / CLI	C++
Google	**Kubernetes**	✓	✓	YAML / JSON	✓	API / CLI	Go

Table 5 - Functionality Comparison of Container Orchestrators & Managers

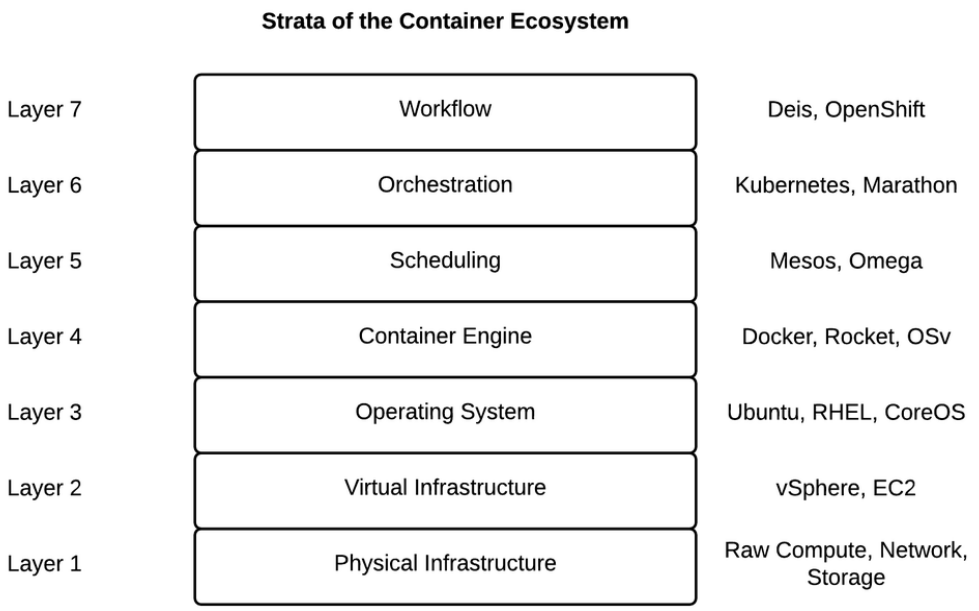

Figure 1 - Strata of Container Ecosystem[19] (Note: OpenStack can also be options at layers 2 & 5)

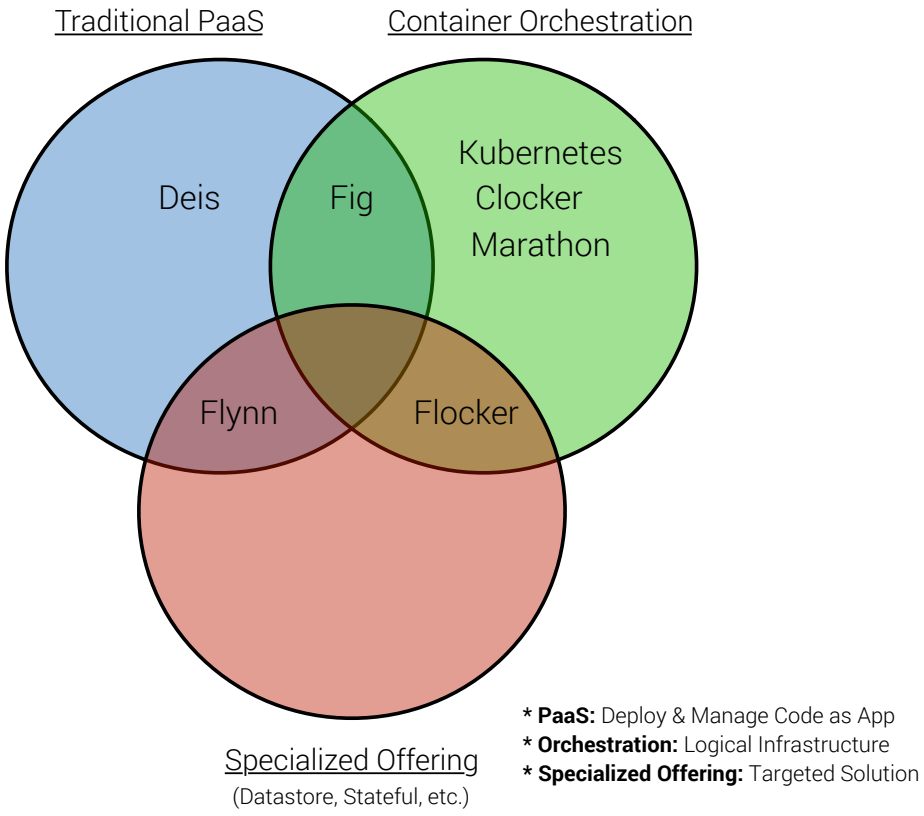

Figure 2 - Venn Diagram of Container Orchestrator & Managers

[19] Gabriel Monroy - https://pbs.twimg.com/media/B33GFtNCUAE-vEX.png:large

Current Recommendation: Kubernetes

8 Miscellaneous

8.1 Container Networking

8.1.1 Weaveworks "Weave"

Weave "makes the network fit the application, not the other way round," as the company CEO puts it. With Weave, Docker containers are all part of a virtual network switch no matter where they're running. Services can be selectively exposed across the network to the outside world through firewalls and using encryption for wide-area connections[20].

When using Weave, "applications use the network just as if the containers were all plugged into the same network switch, with no need to configure port mappings, links, etc. Services provided by application containers on the weave network can be made accessible to the outside world, regardless of where those containers are running. Similarly, existing internal systems can be exposed to application containers irrespective of their location.[21]"

Alexis Richardson, CEO, stated that "weave establishes per application Layer 2 networks for containers across hosts, even across cloud providers and other seemingly complex cases with minimum fuss[22]." Add to the fact that they just raised $5M in Series A, and it makes a compelling argument to considerably evaluate Weave as a viable option.

8.1.2 CoreOS's "Flannel"

Flannel is positioned as CoreOS' primary way to manage container networking via a private mesh network for the containers in a cluster, which happens to do away with issues such as port mapping.

At its core, Flannel is an overlay network that provides a subnet to each machine that initially was intended for Google's Kubernetes, as this is the main operating model that Kubernetes prescribes of all minions/nodes hosting containers. Flannel is backed and based on CoreOS' etcd to serve as the key/value store for the networking configuration and state management. Though it was originally intended for Kubernetes, it has evolved into a generic overlay.

---

[20] http://www.infoworld.com/article/2835222/application-virtualization/5-ways-docker-is-fixing-its-networking-woes.html
[21] https://github.com/zettio/weave
[22] http://www.eweek.com/cloud/weaveworks-raises-5-million-for-docker-container-networking.html

Flannel is still in its early stages and development is very much in flux and somewhat happens in spurts. It should be perceived as experimental but don't disregard Flannel's presence in the market, as their roadmap looks very optimistic given that CoreOS plans to be a big player in the space.

8.1.3 Metaswitch's "Calico"

Project Calico "integrates seamlessly with the cloud orchestration system (such as OpenStack) to enable secure IP communication between virtual machines. As VMs are created or destroyed, their IP addresses are advertised to the rest of the network and they are able to send/receive data over IP just as they would with the native networking implementation – but with higher security, scalability and performance."[23]

In late 2014, the team managed to create a prototype of the Calico stack that runs as Docker containers, in addition to a plugin, that informs it of all containers in the system. This prototype has established that the networking model Calico enables does work for containers as far as a proof of concept.

Though the team seems to have some ideas as to how to proceed with Calico and Docker, there are no short term plans to evolve the prototype and has put the drive and initiative in doing so, into the hands of the community.

8.1.4 SocketPlane's "SocketPlane"

Socketplane's concept is to bring Open vSwitch to the Docker host, so that one can "have a container that's going to be able to manage the data path and also manage either overlays or underlays.[24]"

However, if one where to look for an actual project to evaluate or even their webpage, you'll be met with neither as Socketplane is still very much in a semi-stealth mode. Its relevance and consideration as an option stems from the fact that its founders are three very well known networking gurus as well as contributors on the OpenDaylight Project that left RedHat to start Socketplane. The team currently consists of Madhu Venugopal, Brent Salisbury and Dave Tucker.

It is expected that a product is going to be made available in early 2015, so with both the concept and the team behind it, this could evolve into a sound & promising technology. It has recently been made public that SocketPlane was purchased by Docker Inc. and they plan to natively integrate with the Docker Inc. portfolio.[25]

---

[23] http://www.projectcalico.org/about-calico/
[24] https://www.sdxcentral.com/articles/news/madhu-venugopal-brent-salisbury-opendaylight-starts-open-shop-docker-startup/2014/10/
[25] http://thenewstack.io/docker-acquires-sdn-technology-startup-socketplane-io/

8.1.5 Comparison

It is very early in the Docker ecosystem to tell which container-networking solution will prevail, let alone which are being used at a production scale, as this space is quite new. Though intriguing and backed by some powerful teams, Flannel, Calico for Docker, and SocketPlane show signs that either not enough attention is being given to the project or there have not been any concrete products to seriously evaluate and test.

Current Recommendation: Weave (based on project attention, evolution & funding)

9 Conclusion

9.1 Market Analysis

The immense evolution that containers are bringing forth has created a space for a slew of technologies and tools to emerge, particularly at the orchestration level. Being somewhat of the Wild West right now in terms of competition does not mean that one tool is the be-all and end-all to enable a container offering.

Viewing the ecosystem as a complimentary vertical stack rather than a horizontal one where they are constantly bumping heads helps alleviate the confusion between choosing one tool over the other, as many tools can and will be interoperable with one another. In doing so, this ultimately enables and offers a new stack to power and operate your apps and services.

Now that's not to say that some tools aren't competing with each other directly, or stepping into areas that blur the lines. Because this is such a fast-paced movement, it is expected that the messaging, functionality and roadmap for all tools in this domain will constantly fluctuate.

It is strongly recommended that one should closely track, survey and implement the tools accordingly, based not only on the requirements you would need for your container offering, but more importantly, the popularity of the tool and its reception in the strong community that serves as its driving force.

10 Credits

The following name(s) have reviewed and contributed edits to this document:

- Hugh Blemings – Rackspace

- James Thorne – Rackspace