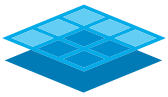
eBook

Container Management: Kubernetes vs Docker Swarm, Mesos + Marathon, Amazon ECS



PLATFORM9

PLATFORM9

What is Container Orchestration?

In the past, it was common for various components of an application to be tightly coupled. Consequently, developers could have spent hours rebuilding monolithic applications, even for minor changes. Recently, however, many technology professionals have begun to see the advantage of using a microservices architecture, wherein the application comprises of loosely coupled components, such as load balancers, caching proxies, message brokers, web servers, application services, and databases. The use of microservices allows developers to quickly create applications. In addition, this architecture saves a tremendous amount of resources in scaling applications, since each component can be scaled separately.

Containers make it easy to deploy and run applications using the microservices architecture. They are lighter-weight compared to VMs and make more efficient use of the underlying infrastructure. Containers are meant to make it easy to scale applications, meet fluctuating demands, and move apps seamlessly between different environments or clouds. While the container runtime APIs meet the needs of managing one container on one host, they are not suited to manage complex environments consisting of many containers and hosts. Container orchestration tools provide this management layer.

Container orchestration tools can treat an entire cluster as a single entity for deployment and management. These tools can provide placement, scheduling, deployment, updates, health monitoring, scaling and failover functionality.

What Can Container Orchestration Tools Do?

Here are some of the capabilities that a modern container orchestration platform will typically provide:

Provisioning

Container orchestration tools can provision or schedule containers within the cluster and launch them. These tools can determine the right placement for the containers by selecting an appropriate host based on the specified constraints such as resource requirements, location affinity etc. The underlying goal is to increase utilization of the available resources. Most tools will be agnostic to the underlying infrastructure provider and, in theory, should be able to move containers across environments and clouds.

Configuration-as-text

Container orchestration tools can load the application blueprint from a schema defined in YAML or JSON. Defining the blueprint in this manner makes it easy for DevOps teams to edit, share and version the configurations and provide repeatable deployments across development, testing and production.

Monitoring

Container orchestration tools will track and monitor the health of the containers and hosts in the cluster. If a container crashes, a new one can be spun up quickly. If a host fails, the tool will restart the failed containers on another host. It will also run specified health checks at the appropriate frequency and update the list of available nodes based on the results. In short, the tool will ensure that the current state of the cluster matches the configuration specified.

Service Discovery

Since containers encourage a microservices based architecture, service discovery becomes a critical function and is provided in different ways by container orchestration platforms e.g. DNS or proxy-based etc. For example, a web application front-end dynamically discovering another microservice or a database.

Rolling Upgrades and Rollback

Some orchestration tools can perform ‘rolling upgrades’ of the application where a new version is applied incrementally across the cluster. Traffic is routed appropriately as containers go down temporarily to receive the update. A rolling update guarantees a minimum number of “ready” containers at any point, so that all old containers are not replaced if there aren’t enough healthy new containers to replace them. If, however, the new version doesn’t perform as expected then the orchestration tool may also be able to rollback the applied change.

Policies for Placement, Scalability etc.

Container orchestration tools provide a way to define policies for host placement, security, performance and high availability. When configured correctly, container orchestration platforms can enable organizations to deploy and operate containerized application workloads in a secure, reliable and scalable way. For example, an application can be scaled up automatically based on CPU usage of the containers.

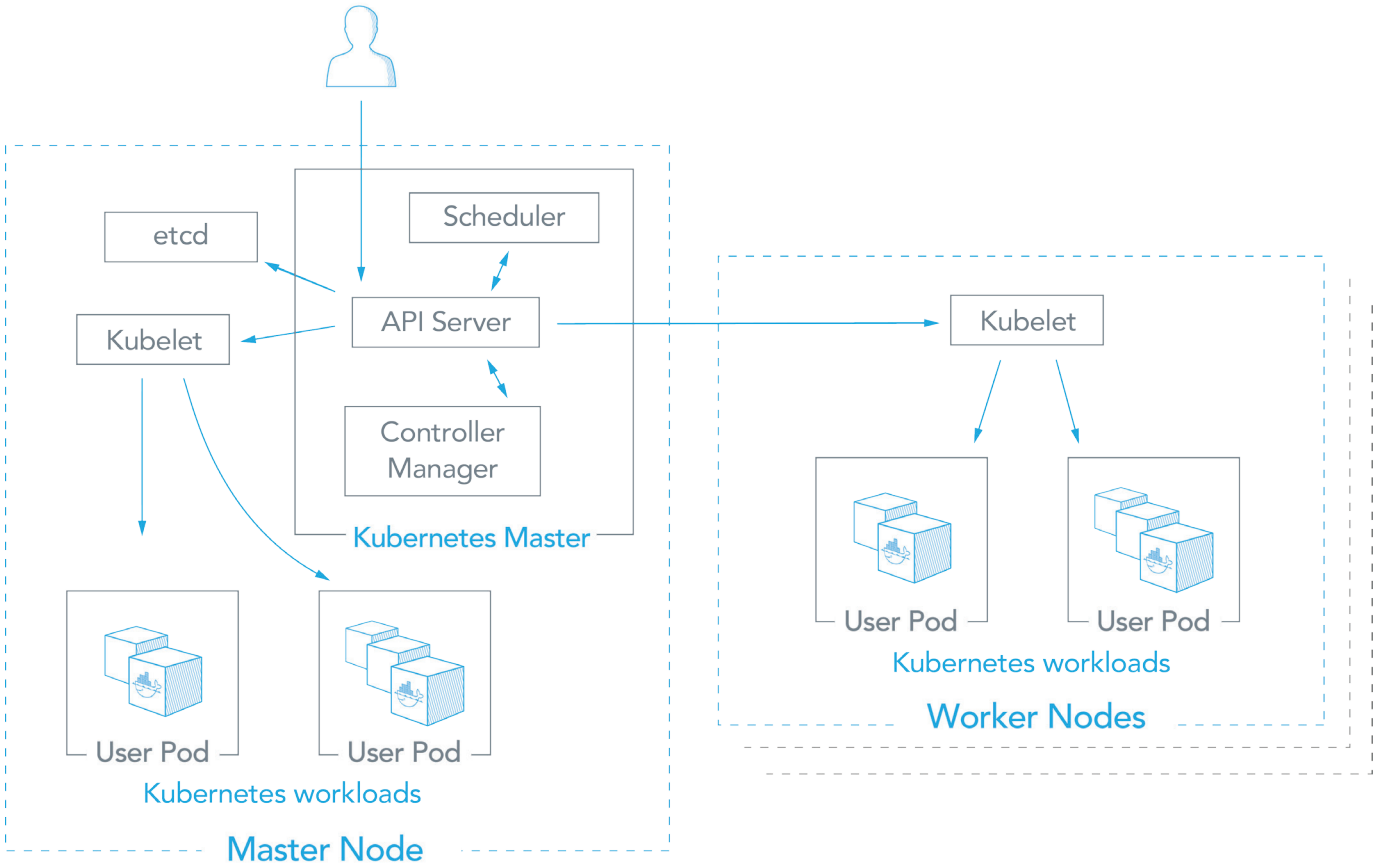
Administration

Container orchestration tools should provide mechanisms for administrators to deploy, configure and setup. An extensible architecture will connect to external systems such as local or cloud storage, networking systems etc. They should connect to existing IT tools for SSO, RBAC etc.

The following sections will introduce Kubernetes, Docker Swarm, Mesos + Marathon, Mesosphere DCOS, and Amazon EC2 Container Service including a comparison of each with Kubernetes.

Kubernetes

According to the Kubernetes website, "Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications." Kubernetes was built by Google based on their experience running containers in production using an internal cluster management system called Borg (sometimes referred to as Omega). The architecture for Kubernetes, which relies on this experience, is shown below:



API Server: management hub for Kubernetes
Scheduler: places a workload on the appropriate Node
Controller Manager: scales workloads up/down
etcd: stores configuration data which can be accessed by API Server

Kubelet: Receives pod specifications from API Server, updates Nodes
Master Node: places workloads on Nodes
Worker Nodes: receives requests from Master Nodes and dispatches them
User Pod: a group of containers with shared resources

As you can see from the figure above, there are a number of components associated with a Kubernetes cluster. The master node places container workloads in user pods on worker nodes or itself. The other components include:

- **etcd:** This component stores configuration data which can be accessed by the Kubernetes master's API Server by simple HTTP or JSON API.
- **API Server:** This component is the management hub for the Kubernetes master node. It facilitates communication between the various components, thereby maintaining cluster health.
- **Controller Manager:** This component ensures that the clusters' desired state matches the current state by scaling workloads up and down.
- **Scheduler:** This component places the workload on the appropriate node.
- **Kubelet:** This component receives pod specifications from the API Server and manages pods running in the host.

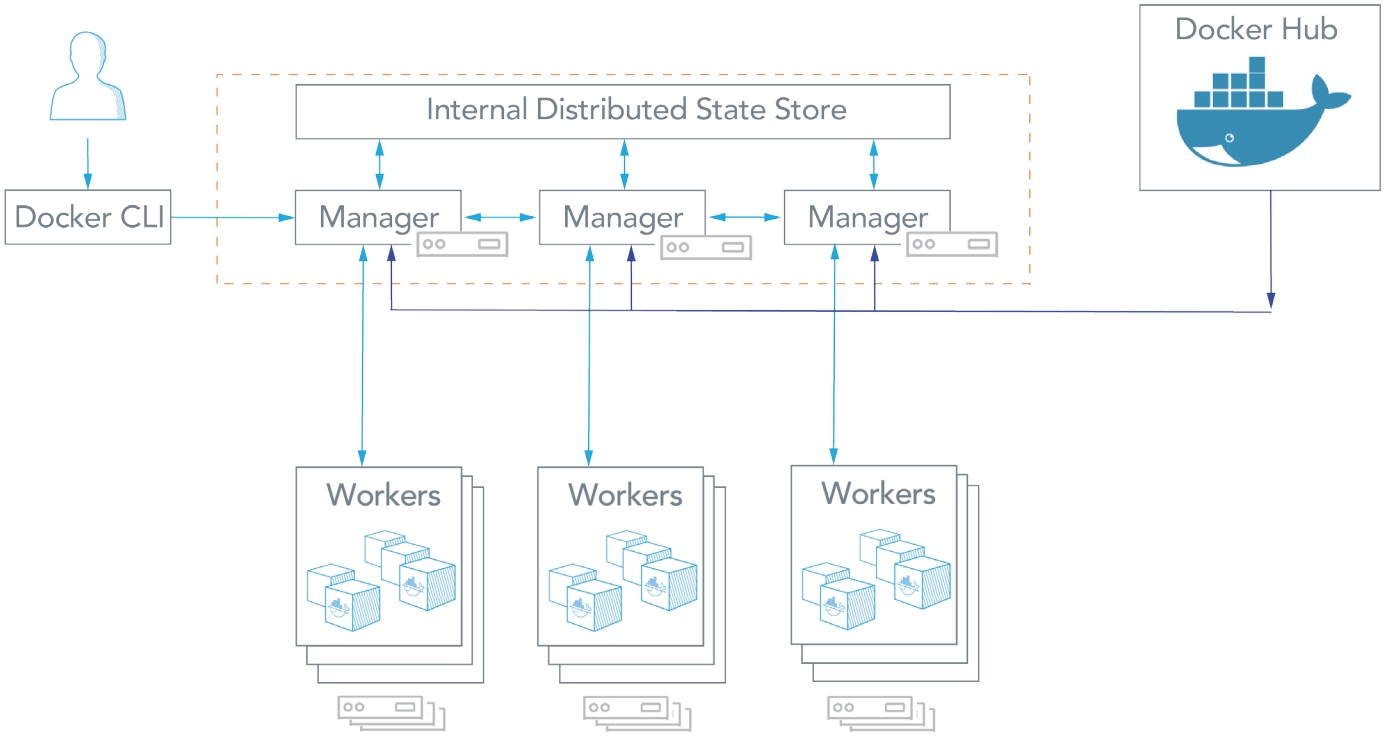
The following list provides some other common terms associated with Kubernetes:

- **Pods:** Kubernetes deploys and schedules containers in groups called pods. Containers in a pod run on the same node aand share resources such as filesystems, kernel namespaces, and an IP address.
- **Deployments:** These building blocks can be used to create and manage a group of pods. Deployments can be used with a service tier for scaling horizontally or ensuring availability.
- **Services:** These are endpoints that can be addressed by name and can be connected to pods using label selectors. The service will automatically round-robin requests between pods. Kubernetes will set up a DNS server for the cluster that watches for new services and allows them to be addressed by name. Services are the "external face" of your container workloads.
- **Labels:** These are key-value pairs attached to objects. They can be used to search and update multiple objects as a single set.

Docker Swarm

Docker Engine v1.12.0 and later allow developers to deploy containers in Swarm mode. A Swarm cluster consists of Docker Engine deployed on multiple nodes. Manager nodes perform orchestration and cluster management. Worker nodes receive and execute tasks from the manager nodes.

A service, which can be specified declaratively, consists of tasks that can be run on Swarm nodes. Services can be replicated to run on multiple nodes. In the replicated services model, ingress load balancing and internal DNS can be used to provide highly available service endpoints. (Source: Docker Docs: Swarm mode)



Manager: a node that dispatches tasks
Worker: a node that executes tasks provided by a Manager
Internal Distributed Store: used to maintain cluster state

Docker CLI: User interacts with the swarm using Docker CLI, for example "docker service"
Docker Hub: contains repositories for downloading and sharing container images

As can be seen from the figure above, the Docker Swarm architecture consists of managers and workers. The user can declaratively specify the desired state of various services to run in the Swarm cluster using YAML files. Here are some common terms associated with Docker Swarm:

- **Node:** A node is an instance of a Swarm. Nodes can be distributed on-premises or in public clouds.
- **Swarm:** A cluster of nodes (or Docker Engines). In Swarm mode, you orchestrate services, instead of running container commands.
- **Manager Nodes:** These nodes receive service definitions from the user, and dispatch work to worker nodes. Manager nodes can also perform the duties of worker nodes.
- **Worker Nodes:** These nodes collect and run tasks from manager nodes.
- **Service:** A service specifies the container image and the number of replicas. Here is an example of a service command which will be scheduled on 2 available nodes:

```
# docker service create --replicas 2 --name mynginx nginx
```

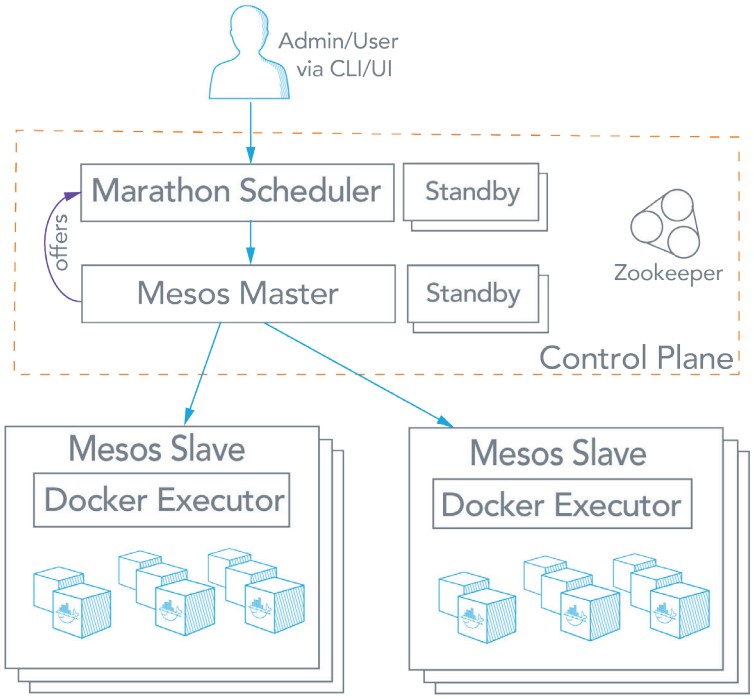
- **Task:** A task is an atomic unit of a Service scheduled on a worker node. In the example above, two tasks would be scheduled by a master node on two worker nodes (assuming they are not scheduled on the Master itself). The two tasks will run independently of each other.

PLATFORM9

Mesos + Marathon

Mesos is a distributed kernel that aims to provide dynamic allocation of resources in your datacenter. Imagine that you manage the IT department of a mid-size business. You need to have workloads running on 100 nodes during the day but on 25 after hours. Mesos can redistribute workloads so that the other 75 nodes can be powered-off when they are not used. Mesos can also provide resource sharing. In the event that one of your nodes fails, workloads can be distributed among other nodes.

Mesos comes with a number of frameworks, application stacks that use its resource sharing capabilities. Each framework consists of a scheduler and a executor. Marathon is a framework (or meta framework) that can launch applications and other frameworks. Marathon can also serve as a container orchestration platform which can provide scaling and self-healing for containerized workloads. The figure below shows the architecture of Mesos + Marathon.



Mesos Master: manages resources in cluster. Provides offers to Marathon
Mesos Slave: runs agents which report resources to master
Offer: a list of available CPU and memory resources for slave nodes
Standby: activated if current masters for Mesos/Marathon fail

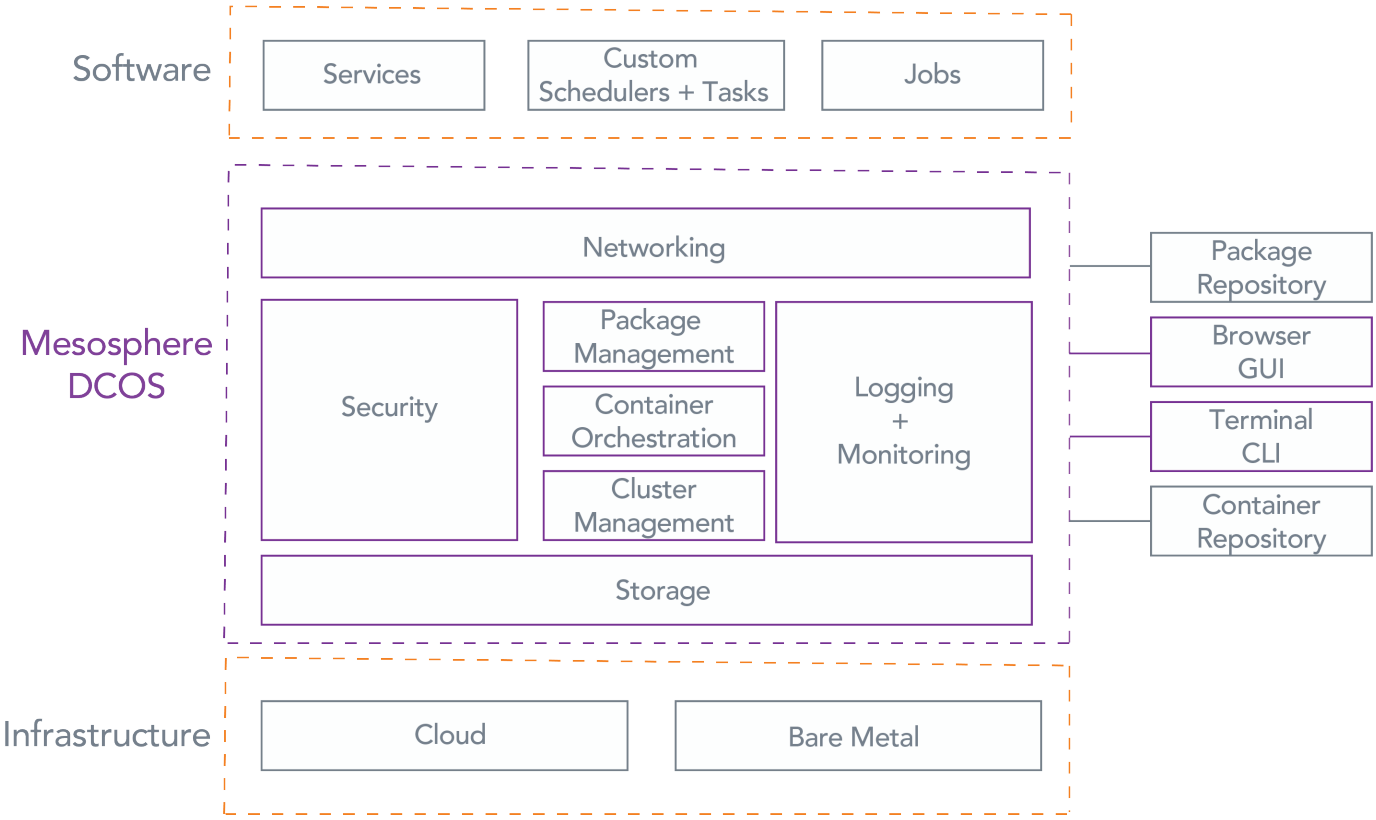
Marathon Scheduler: registers with Mesos master to receive offers
Docker Executor: executes tasks from Marathon scheduler
Zookeeper: enables high availability of Mesos and Marathon

There are a number of different components in Mesos and Marathon. The following list provides some common terms:

- **Mesos Master:** This type of node enables the sharing of resources across frameworks such as Marathon for container orchestration, Spark for large-scale data processing, and Cassandra for NoSQL databases.
- **Mesos Slave:** This type of node runs agents that report available resources to the master.
- **Framework:** A framework registers with the Mesos master so that the master can be offered tasks to run on slave nodes.
- **Zookeeper:** This component provides a highly available database that can the cluster can keep state, i.e. the active master at any given time.
- **Marathon Scheduler:** This component receives offers from the Mesos master. Offers from the Mesos master list slave nodes' available CPU and memory.
- **Docker Executor:** This component receives tasks from the Marathon scheduler and launches containers on slave nodes.

Mesosphere DCOS

Mesosphere Enterprise DC/OS leverages the Mesos distributed systems kernel and builds on it with container and big data management, providing installation, user interfaces, management and monitoring tools, and other features. The diagram below shows a high-level architecture of DCOS.

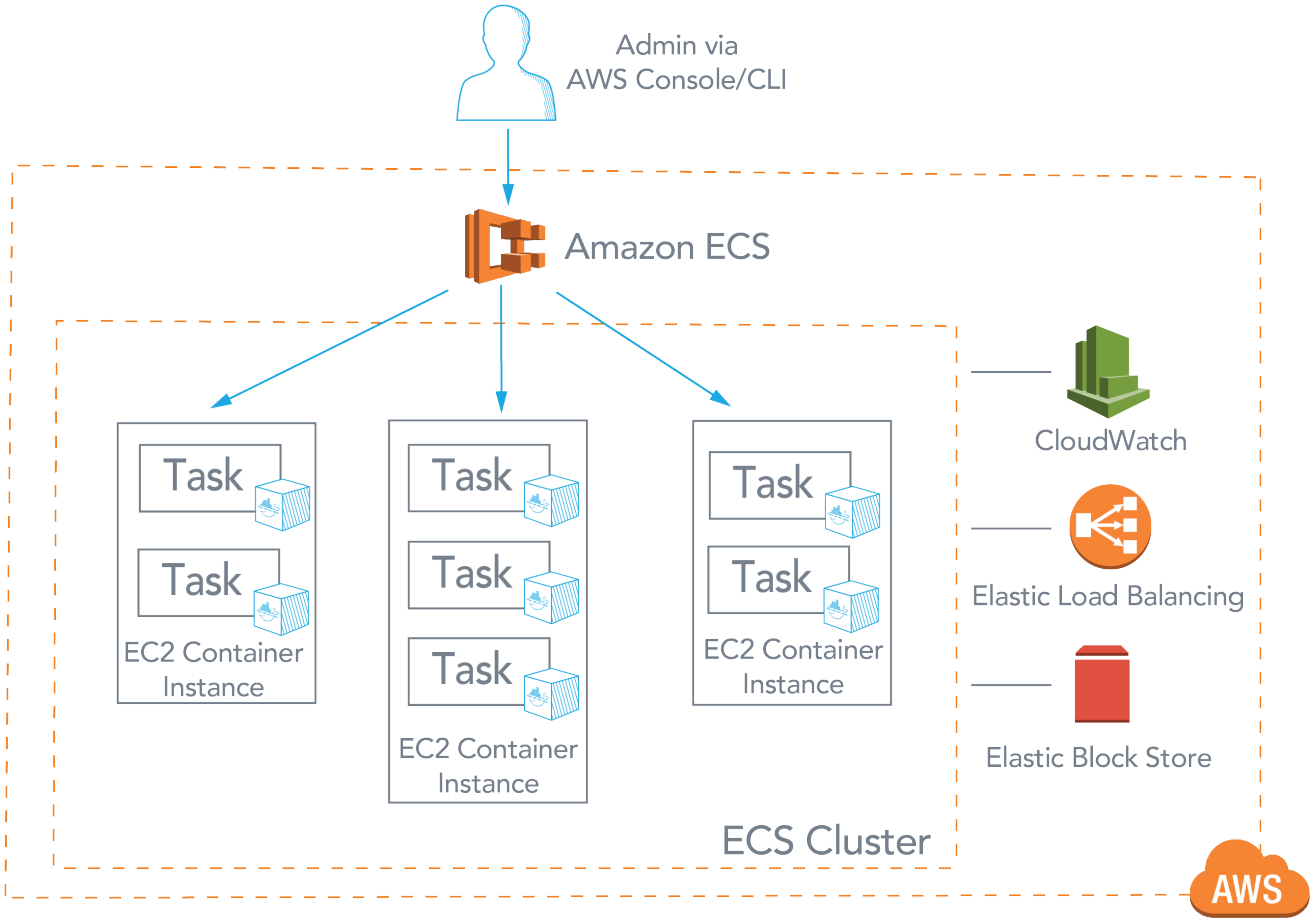


Source: DCOS Documentation – Architecture

As shown above, DCOS is comprised of package management, container orchestration (derived from Marathon), cluster management (derived from Mesos), and other components. Further details on Mesosphere DCOS can be found in DCOS documentation.

Amazon ECS

Amazon ECS is the Docker-compatible container orchestration solution from Amazon Web Services. It allows you to run containerized applications on EC2 instances and scale both of them. The following diagram shows the high-level architecture of ECS.



Amazon ECS: EC2 Container Service
Task: a unit of work, which usually consists of one container
EC2 Container Instance: compute environment for running tasks
ECS Cluster: a logical placement boundary for container instances

CloudWatch: monitoring service from Amazon
Elastic Load Balancing: load balancing service
Elastic Block Store: persistent storage service

As shown above, ECS Clusters consist of tasks which run in Docker containers, and container instances, among many other components. Here are some AWS services commonly used with ECS:

- **Elastic Load Balancer:** This component can route traffic to containers. Two kinds of load balancing are available: application and classic.
- **Elastic Block Store:** This service provides persistent block storage for ECS tasks (workloads running in containers).
- **CloudWatch:** This service collects metrics from ECS. Based on CloudWatch metrics, ECS services can be scaled up or down.
- **Virtual Private Cloud:** An ECS cluster runs within a VPC. A VPC can have one or more subnets.
- **CloudTrail:** This service can log ECS API calls. Details captured include type of request made to Amazon ECS, source IP address, user details, etc.

ECS, which is provided by Amazon as a service, is composed of multiple built-in components which enable administrators to create clusters, tasks and services:

- **State Engine:** A container environment can consist of many EC2 container instances and containers. With hundreds or thousands of containers, it is necessary to keep track of the availability of instances to serve new requests based on CPU, memory, load balancing, and other characteristics. The state engine is designed to keep track of available hosts, running containers, and other functions of a cluster manager.

- **Schedulers:** These components use information from the state engine to place containers in the optimal EC2 container instances. The batch job scheduler is used for tasks that run for a short period of time. The service scheduler is used for long running apps. It can automatically schedule new tasks to an ELB.

- **Cluster:** This is a logical placement boundary for a set of EC2 container instances within an AWS region. A cluster can span multiple availability zones (AZs), and can be scaled up/down dynamically. A dev/test environment may have 2 clusters: 1 each for production and test.

- **Tasks:** A task is a unit of work. Task definitions, written in JSON, specify containers that should be co-located (on an EC2 container instance). Though tasks usually consist of a single container, they can also contain multiple containers.

- **Services:** This component specifies how many tasks should be running across a given cluster. You can interact with services using their API, and use the service scheduler for task placement.

Note that ECS only manages ECS container workloads – resulting in vendor lock-in. There's no support to run containers on infrastructure outside of EC2, including physical infrastructure or other clouds such as Google Cloud Platform and Microsoft Azure. The advantage, of course, is the ability to work with all the other AWS services like Elastic Load Balancers, CloudTrail, CloudWatch etc.

Further details about Amazon ECS can be found in AWS ECS Documentation.

PLATFORM9

Kubernetes vs Swarm vs Mesos vs ECS Comparison

Kubernetes

Docker Swarm

Mesos + Marathon

Amazon ECS

Application Definition

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Applications can be deployed using a combination of pods, deployments, and services. A pod is a group of co-located containers and is the atomic unit of a deployment. A deployment can have replicas across multiple nodes. A service is the "external face" of container workloads and integrates with DNS to round-robin incoming requests. Load balancing of incoming requests is supported.	Applications can be deployed as services in a Swarm cluster. Multi-container applications can specified using YAML files. Docker Compose can deploy the app. tasks (an instance of a service running on a node) can be distributed across datacenters using labels. Multiple placement preferences can be used to distribute tasks further, for example, to a rack in a datacenter.	From the user's perspective, an application runs as tasks that are scheduled by Marathon on nodes. For Mesos, an application is a framework, which can be Marathon, Cassandra, Spark and others. Marathon in-turn schedules containers as tasks which are executed on slave nodes. Marathon 1.4 introduces the concept of pods (like Kubernetes), but this isn't part of the Marathon core. Nodes can be tagged based on racks, type of storage attached, etc. These constraints can be used when launching Docker containers.	Applications can be deployed as tasks, which are Docker containers running on EC2 instances (aka container instances). Task definitions specify the container image, CPU, memory and persistent storage in a JSON template. Clusters comprise of one or more tasks that use these task definitions. Schedulers automatically place containers across compute nodes in a cluster, which can also span multiple AZs. Services can be created by specifying number of tasks and an Elastic Load Balancer.

Application Scalability Constructs

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Each application tier is defined as a pod and can be scaled when managed by a deployment, which is specified declaratively, e.g., in YAML. The scaling can be manual or automated. Pods are most useful for running co-located and co-administered helper applications, like log and checkpoint backup agents, proxies and adapters, though they can also be used to run vertically integrated application stacks such as LAMP (Apache, MySQL, PHP) or ELK/Elastic (Elasticsearch, Logstash, Kibana).	Services can be scaled using Docker Compose YAML templates. Services can be global or replicated. Global services run on all nodes, replicated services run replicas (tasks) of the services across nodes. For example, A MySQL service with 3 replicas will run on a maximum of 3 nodes. Tasks can be scaled up or down, and deployed in parallel or in sequence.	Mesos CLI or UI can be used. Docker containers can be launched using JSON definitions that specify the repository, resources, number of instances, and command to execute. Scaling-up can be done by using the Marathon UI, and the Marathon scheduler will distribute these containers on slave nodes based on specified criteria. Autoscaling is supported. Multi-tiered applications can be deployed using application groups.	Applications can be defined using task definitions written in JSON. Tasks are instantiations of task definitions and can be scaled up or down manually. The built-in scheduler will automatically distribute tasks across ECS compute nodes. For a vertically integrated stack, task definitions can specify one tier which exposes an http endpoint. This endpoint can in-turn be used by another tier, or exposed to the user.

PLATFORM9

Kubernetes

Docker Swarm

Mesos + Marathon

Amazon ECS

High Availability

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Deployments allow pods to be distributed among nodes to provide HA, thereby tolerating infrastructure or application failures. Load-balanced services detect unhealthy pods and remove them. High availability of Kubernetes is supported. Multiple master nodes and worker nodes can be load balanced for requests from kubectl and clients. etcd can be clustered and API Servers can be replicated.	Services can be replicated among Swarm nodes. Swarm managers are responsible for the entire cluster and manage the resources of worker nodes. Managers use ingress load balancing to expose services externally. Swarm managers use Raft Consensus algorithm to ensure that they have consistent state information. An odd number of managers is recommended, and a majority of managers must be available for a functioning Swarm cluster (2 of 3, 3 of 5, etc.).	Containers can be scheduled without constraints on node placement, or each container on a unique node (the number of slave nodes should be at least equal to the number of containers). High availability for Mesos and Marathon is supported using Zookeeper. Zookeeper provides election of Mesos and Marathon leaders and maintains cluster state.	Schedulers place tasks, which are comprised of 1 or more containers, on EC2 container instances. Tasks can be increased or decreased manually to scale. Elastic Load Balancers can distribute traffic among healthy containers. ECS control plane high availability is taken care of by Amazon. Requests can be load-balanced to multiple tasks using ELB.

Load Balancing

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Pods are exposed through a service, which can be used as a load-balancer within the cluster. Typically, an ingress resource is used for load balancing.	Swarm mode has a DNS component that can be used to distribute incoming requests to a service name. Services can run on ports specified by the user or can be assigned automatically.	Host ports can be mapped to multiple container ports, serving as a front-end for other applications or end users.	ELB provides a CNAME that can be used within the cluster. This CNAME serves as a front-facing resolvable FQDN for multiple tasks. Two kinds of service load balancers with ELB: application or classic.

PLATFORM9

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Auto-scaling for the Application			
Auto-scaling using a simple number-of-pods target is defined declaratively using deployments. Auto-scaling using resource metrics is also supported. Resource metrics range from CPU and memory utilization to requests or packets-per-second, and even custom metrics.	Not directly available. For each service, you can declare the number of tasks you want to run. When you manually scale up or down, the Swarm manager automatically adapts by adding or removing tasks.	Marathon continuously monitors the number of instances of a Docker container that are running. If one of the containers fail, Marathon reschedules it on other slave nodes. Auto-scaling using resource metrics is available through community-supported components only.	CloudWatch alarms can be used to auto-scale ECS services up or down based on CPU, memory, and custom metrics.
Rolling Application Upgrades and Rollback			
A deployment supports both "rolling-update" and "recreate" strategies. Rolling updates can specify maximum number of pods.	At rollout time, you can apply rolling updates to services. The Swarm manager lets you control the delay between service deployment to different sets of nodes, thereby updating only 1 task at a time.	Rolling upgrades of applications are supported using deployments. A failed upgrade can be healed using an updated deployment that rolls-back the changes.	Rolling updates are supported using "minimumHealthyPercent" and "maximumPercent" parameters. The same parameters can be adjusted to do blue-green updates, which adds a whole new batch of containers in parallel with the existing set.
Health Checks			
Health checks are of two kinds: liveness (is app responsive) and readiness (is app responsive, but busy preparing and not yet able to serve).	Docker Swarm health checks are limited to services. If a container backing the service does not come up (running state), a new container is kicked off. Users can embed health check functionality into their Docker images using the HEALTHCHECK instruction.	Health checks can be specified to be run against the application's tasks. Health check requests are available in a number of protocols, including HTTP, TCP, and others.	ECS provides health checks using CloudWatch.

PLATFORM9

Kubernetes

Docker Swarm

Mesos + Marathon

Amazon ECS

Storage

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Two storage APIs: The first provides abstractions for individual storage backends (e.g. NFS, AWS EBS, Ceph, Flocker). The second provides an abstraction for a storage resource request (e.g. 8 Gb), which can be fulfilled with different storage backends. Kubernetes offers several types of persistent volumes with block or file support. Examples include iSCSI, NFS, FC, Amazon Web Services, Google Cloud Platform, and Microsoft Azure. The emptyDir volume is non-persistent and can used to read and write files with a container.	Docker Engine and Swarm support mounting volumes into a container. Shared filesystems, including NFS, iSCSI, and fibre channel, can be configured nodes. Plugin options include Azure, Google Cloud Platform, NetApp, Dell EMC, and others.	Local persistent volumes (beta) are supported for stateful applications such as MySQL. When needed, tasks can be restarted on the same node using the same volume. The use of external storage, such as Amazon EBS, is also in beta. At the present time, applications that use external volumes can only be scaled to a single instance because a volume can only attach to a single task at a time.	Block storage support is limited to Amazon Web Services. EBS volumes can be specified by using ECS task definitions (JSON files) and connected to container instances. Task definitions have a "containerDefinitions" section which can be used to enable "mountPoints." Multiple containers on a container instance can be connected to an EBS storage volume. (Reference: Amazon Web Services Docs)

Networking

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
The networking model is a flat network, enabling all pods to communicate with one another. Network policies specify how pods communicate with each other. The flat network is typically implemented as an overlay. The model requires two CIDRs: one from which pods get an IP address, the other for services.	Node joining a Docker Swarm cluster creates an overlay network for services that span all of the hosts in the Swarm and a host only Docker bridge network for containers. By default, nodes in the Swarm cluster encrypt overlay control and management traffic between themselves. Users can choose to encrypt container data traffic when creating an overlay network by themselves.	Networking can be configured in host mode or bridge mode. In host mode, the host ports are used by containers. This can lead to port conflicts on any given host. In bridge mode, the container ports are bridged to host ports using port mapping. Host ports can be dynamically assigned at time of deployment.	ECS is supported in a VPC, which can include multiple subnets in multiple AZs. Communication within a subnet cannot be restricted using AWS tools.

PLATFORM9

Kubernetes

Docker Swarm

Mesos + Marathon

Amazon ECS

Service Discovery

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
Services can be found using environment variables or DNS. Kubelet adds a set of environment variables when a pod is run. Kubelet supports simple "{SVCNAME_SERVICE_HOST}" and "{SVCNAME_SERVICE_PORT}" variables, as well as Docker links compatible variables. DNS Server is available as an addon. For each Kubernetes service, the DNS Server creates a set of DNS records. With DNS enabled in the entire cluster, pods will be able to use service names that automatically resolve.	Swarm Manager node assigns each service a unique DNS name and load balances running containers. Requests to services are load balanced to the individual containers via the DNS server embedded in the Swarm. Docker Swarm comes with multiple discovery backends: • Docker Hub as a hosted discovery service is intended to be used for dev/test. Not recommended for production. • A static file or list of nodes can be used as a discovery backend. The file must be stored on a host that is accessible from the Swarm Manager. You can also provide a node list as an option when you start Swarm.	Services can be discovered via "named VIPs," which are DNS records that are are associated with IPs and ports. Services are automatically assigned DNS records by Mesos-DNS. An optional named VIP can be created; requests via the VIP are load-balanced.	Services can be found using an ELB and a CNAME. A single ELB can be used per service. Route53 private hosted zones can be used to ensure that the ELB CNAMEs are only resolvable within your VPC.

Performance and Scalability

Kubernetes	Docker Swarm	Mesos + Marathon	Amazon ECS
With the release of 1.6, Kubernetes scales to 5,000-node clusters. Multiple clusters can be used to scale beyond this limit.	According to the Docker's blog post on scaling Swarm clusters, Docker Swarm has been scaled and performance tested up to 30,000 containers and 1,000 nodes with 1 Swarm manager.	Mesos' 2 tier architecture (with Marathon) makes is very scalable. According to Digital Ocean, Mesos and Marathon clusters have been scaled to 10,000 nodes.	ECS has been scaled-up to over a 1000 container nodes without noticeable performance degradation. (Deep Dive on Microservices and Amazon ECS, skip to 11:27)

PLATFORM9

Advantages/Disadvantages of Container Orchestration Solutions

Kubernetes

Docker Swarm

Mesos + Marathon

Amazon ECS

Advantages

- Based on extensive experience at Google.
- Deployed at scale more often. Backed by enterprise offerings from both Google (GKE) and RedHat (OpenShift).
- Can overcome constraints of Docker and Docker API.
- Largest community among container orchestration tools. Over 50,000 commits and 1200 contributors.
- Wide variety of storage options, including on-premises SANs and public clouds.

- Simple deployment. Swarm mode is included in Docker Engine.
- Integrates with Docker Compose and Docker CLI – native Docker tools.
- Many of the Docker CLI commands will work with Swarm. Easier learning curve.
- Various storage options Plugins include Azure, Google Cloud Platform, NetApp, Dell EMC, etc.

- Single vendor control may allow for accountability with bug fixes and better coordination with feature development.
- The 2-tier architecture allows for deploying other frameworks (workloads). Examples include Spark, Chronos, and Redis.
- Can overcome constraints of Docker and Docker API.
- Organizations have deployed Mesos at massive scale greater than 10,000 nodes. (Source: Mesosphere blog)

- ECS does not require installation on servers. ECS CLI installation is simple.
- Based on experience from operating scalable public clouds.
- Single vendor control may allow for accountability with bug fixes and better coordination with feature development.

PLATFORM9

Kubernetes

Docker Swarm

Mesos + Marathon

Amazon ECS

Disadvantages

- Do-it-yourself installation can be complex. Further details on deployment tools in Kubernetes Deployment Guide.
- Uses a separate set of tools for management, including kubectl CLI.
- Lack of single vendor control can complicate purchasing decisions. Community includes Google, Red Hat, and over 2000 authors. (Source: CNCF)

- Does not have much experience with production deployments at scale.
- Limited to the Docker API's capabilities.
- Services have to be scaled manually.
- Smaller community and project. Over 3,000 commits and 160 contributors.

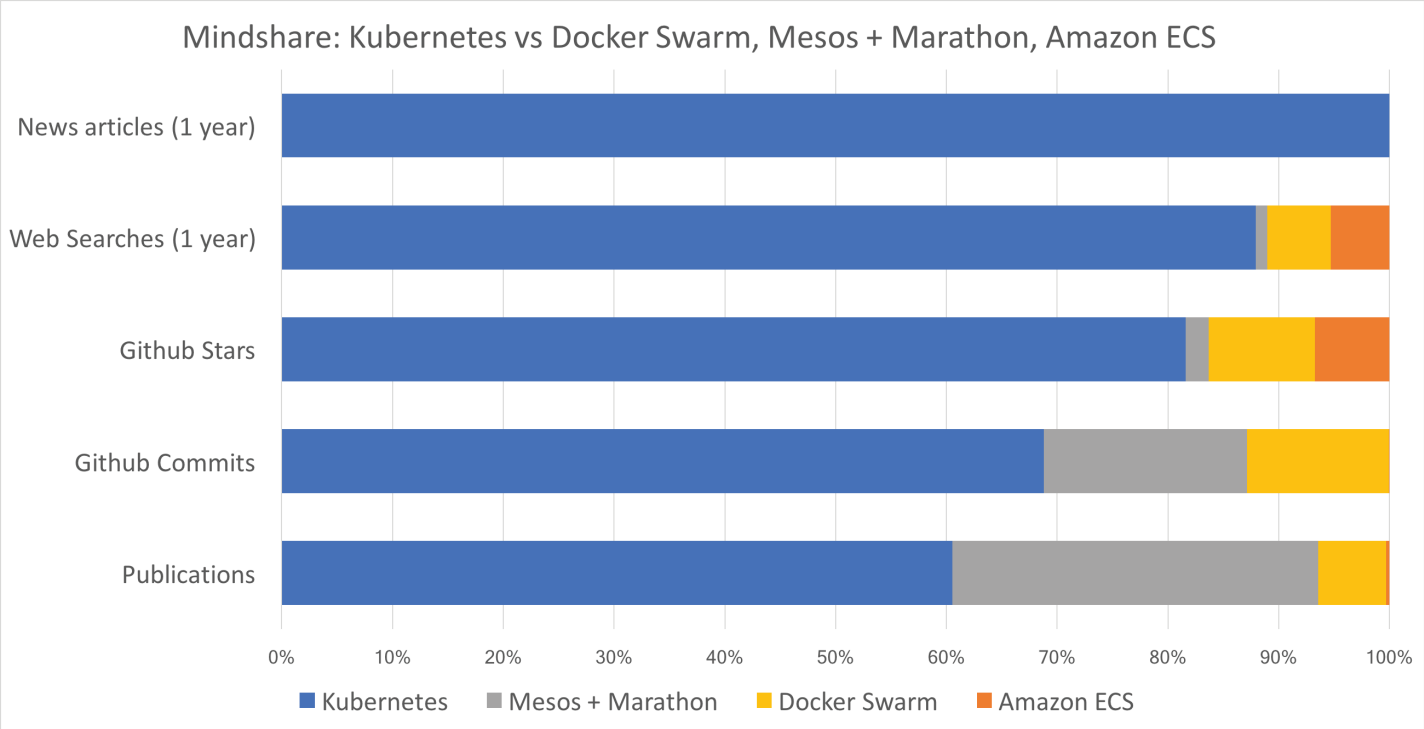
- External storage on Mesos + Marathon, including Amazon EBS is in beta.
- Mesosphere's DCOS product is the only commercial distribution of Mesos.
- Smaller community. Over 12,000 commits and 240 contributors.
- Installation can be complex due to the 2 tier architecture, Zookeeper for cluster management, HA Proxy for load balancing, etc.
- Multiple sets of tools for management: Mesos CLI, Mesos UI, Marathon CLI, Marathon UI.

- Vendor lock-in. Containers can only be deployed on Amazon, and ECS can only manage containers that it has created.
- External storage is limited to Amazon.
- Validated within Amazon. ECS is not publicly available for deployment outside Amazon.
- Much of ECS code is not publicly available. Parts of ECS, including Blox, a framework that helps build custom schedulers, are open source. 200 commits and 15 contributors.

PLATFORM9

Conclusion

The chart below provides a summary of mindshare for Kubernetes, Docker Swarm, Mesos + Marathon, and Amazon ECS.



As you can see, Kubernetes has an overwhelming presence in online news and web searches. On other metrics, Kubernetes has over 65% of mindshare on Github, and leads on number of publications, as measured by search results. Here are the top three reasons why Kubernetes is the #1 container orchestration solution today:

- Much wider adoption by DevOps and containers communities
- Backed by enterprise offerings such as Google Container Engine (GKE) and Red Hat OpenShift
- Based on over a decade of experience at Google

However, Kubernetes has been known to be difficult to deploy and manage. Platform9's Managed Kubernetes product can fill this gap by letting organizations focus on deploying microservices on Kubernetes, instead of deploying, operating, and upgrading a highly available Kubernetes deployments themselves. Specifically, Platform9 Managed Kubernetes can provide these benefits:

- **Speed:** Gain from rapid roll-out of enterprise Kubernetes. Our Sandbox takes seconds to setup, and production installations usually take a few hours.
- **Simplicity:** Organizations leveraging Platform9 Managed Kubernetes do not need to deploy, monitor, and manage highly available Kubernetes, including etcd databases, API servers, kubelets, etc. Kubernetes version upgrades are performed with zero downtime.
- **Scale:** Customers can scale cost-effectively by provisioning containers on-premises and in public clouds, including Amazon Web Services, Google Cloud Platform, and Microsoft Azure.

Take a guided tour of Platform9's Managed Kubernetes using a free Sandbox. You can also learn more about various deployment options for Kubernetes, including local installations, managed services, and public clouds, using The Ultimate Guide to Deploy Kubernetes.

For information about our enterprise proof of concept program, please email us at info@platform9.com.