Overview of Docker in the Cloud

docker

Table of Contents

Introduction

This paper will discuss the options for running containers on the three largest cloud providers: Amazon Web Services, Microsoft Azure and Google Cloud Platform. For pragmatic purposes, we will limit the discussion to Docker-as-a-container technology, which has emerged as the de facto standard for container runtimes.

The target audience for this article are those familiar with software development and/or hardware infrastructure with a basic knowledge of containers. The goal is to leave the reader with an understanding of the different approaches of each cloud provider along with a level 200 evaluation of each of their different offerings. This will aid the reader in a next-step decision of which cloud provider they should choose for their own container platform.

Container technology has existed in some form for several decades. What began as a low-level Unix system for process isolation in the late 1970s has evolved into a fully-featured operating system virtualization technology that works on both Linux and Windows operating systems.

Containers have exploded in popularity in recent years for a variety of reasons. Containerization has played a key role in the evolution of deployment models, from physical machines to virtual machines to virtualized operating systems. Because of the nature of containers, they can be ultra-lightweight and have startup times that are an order of magnitude better than virtual machines, thus improving performance and virtualized density per physical/logical host. More recently, the availability on Windows operating systems has made the world of containers more accessible to a larger subset of organizations.

Containers address several traditional challenges around the software development lifecycle (which will not covered in this piece), but because of this, containers are seeing considerable demand in all phases of development lifecycles and support paradigms - namely DevOps efforts and continuous integration / continuous deployment strategies). At the same time, the emergence of cloud platforms as commodity computing resources has meant that a container deployment model fits particularly well today.

However, while containers solve several historical challenges, they also introduce several other characteristics that warrant consideration. The container itself is only one facet of a potential solution, but we must also consider additional aspects, such as how to distribute containers across nodes (hosts), how to manage load-balancing and failover, how to instrument containers, and a handful of other factors.

Each cloud provider has container offerings to accelerate and solve many of the challenges that systems of complexity and scale require. These offerings fall into the Platform as a Service or Container as a Service category to further reduce the burden on technical teams for building, deploying, and managing a Docker solution through development to production. In each cloud provider section, individual solutions or offerings will be reviewed against the characteristics.

NOTE: Infrastructure-as-a-Service (IaaS) solutions are not covered in this piece unless there is something compelling to note, as they will be common across the three cloud platforms. This also would result in an evaluation of the cloud providers IaaS platform instead of the Container offerings.

In the next section, we cover the characteristics that are important to a potential container strategy. While most of these elements can be managed manually, a container solution must support scale, in which case mature and robust management of a container solution becomes critically important.

Implementation Attributes

For each solution, we discuss its fit against a set of canonical characteristics. The list of characteristics used are below.

Runtime Characteristics

For each possible solution, we cover the following runtime characteristics.

CHARACTERISTIC	DESCRIPTION
Scheduling and Orchestration	Containers are deployed to a single node (this can be a virtual or physical machine). When and where a specific container is deployed is a function of the scheduler. This can be a function of available node resources, choices for high availability, or a variety of other concerns.
Routing and Load Balancing	Multiple containers may be running the same image for availability and failover. How a request is fulfilled between the same services is managed by the routing and load balancing characteristic of the solution.
Service Discovery	When a new instance of a container is started, the routing and load balancing solution needs to be made aware so that it can be included in the routing strategy.
Failover and Recovery	The solution should support detection of failed containers and restart as needed.
Cluster State	The context of a running container solution must be managed. How this context managed and stored is an important function of the solution.
Container Deployment Unit	While a container is the unit of deployment for an image (typically a single service/function), containers are often deployed in logical groupings. The definition of how containers are deployed together, how many of each, etc. can differ across strategies.

Table 1: Runtime Characteristics

DevOps Characteristics

Instrumentation is a critical element of a container strategy. Support personnel must have visibility into not only the behavior and performance of the container, but also the host nodes and solution infrastructure. The following characteristics are important for supporting this transparency.

CHARACTERISTIC	DESCRIPTION
Logging	We need support for forwarding logs from the container itself as well as any logs produced internally that may be of interest.
Monitoring	We need support for monitoring behavior of the container ecosystem, single containers, or other elements of the solution strategy.

Table 2: DevOps Characteristics

Management and Orchestration Platforms

Many platforms exist today to manage and orchestrate container environments such as Docker Swarm or Kubernetes. These existing platforms are integral to some cloud provider's container offering. This may be built-in or a configuration option at run-time. Since many organizations already leverage some of these existing platforms, this was also included in the evaluation.

Image Management

Containers rely on the notion of an image, the definition (or template) of a running container instance. Building this template results in an artifact that instantiates into a running container. Container-based strategies often include the creation and management of images as part of a development lifecycle; this supports traceability and security of production instances.

The process of creating an image can be done manually or automatically (for example, in response to a change in a template definition within source control). The resulting image is typically stored in an image registry, a platform that tracks and secures container images.

In most scenarios, the choice of how to build images and where to store the resulting artifact can be opaque to a solution. A few solutions will implicitly support image creation and registry capabilities, but most do not. If the solution chosen supports enterprise requirements (for example, images need to be signed and verified before production deployments), then the choice is largely arbitrary.

As such, we don't explicitly cover Image Management when we examine the solutions -- except in cases where it may provide a unique set of capabilities.

Solution Rubric

For each strategy discussed for supporting Docker in the cloud, we evaluate each characteristic using the following generalized rating values.

VALUE	DESCRIPTION
-1	The solution doesn't support the characteristic in any way.
0	The solution partially supports the characteristic. Caveats or notes are included in the description.
1	The solution supports the characteristic.

Table 3: Rubric Ratings

For specific characteristics, see the following table for additional descriptions:

CHARACTERISTIC	-1 (Not Supported)	0 (Partially Supported)	1 (Supported)
Scheduling and Orchestration	You must provide your own scheduling and orchestration support.	Scheduling and orchestration of containers / services may require third-party services to automate or require some manual actions.	Scheduling and orchestration is fully managed in the solution.
Routing and Load Balancing	You must provide your own routing and load balancing support.	Routing and load balancing may require third-party services to automate or require some manual actions.	Routing and load-balancing is fully and automatically managed in the solution.
Service Discovery	You must provide your own service discovery support.	Inclusion of new container instances into the routing and load-balancing solution may require wiring in third-party services to automate or require some manual actions.	Service discovery is fully supported in the solution.

Table 1: Ratings Descriptions

CHARACTERISTIC	-1 (Not Supported)	0 (Partially Supported)	1 (Supported)
Failover and Recovery	You must provide your own failover and recovery support.	Health checks and recovery from failed services may require third-party services to automate or require some manual actions.	Failover and recovery is fully managed by the solution.
Cluster State	You must provide your own failover and recovery support.	Management and persistence of container / host state may require third-party services to automate or require some manual actions.	Cluster state is fully managed by the solution.
Container Deployment Unit	There is no notion of container deployment unit.	(Not a characteristic possibility.)	Container deployment units are a fundamental characteristic of the solution.
Logging	You must provide your own logging capture support.	Log forwarding / capture may require third-party services to fully automate.	Logging capture is provided natively in the solution.
Monitoring	You must provide your own monitoring support.	Monitoring and notification from aggregate system logs may require third-party solutions to automate.	Monitoring is provided natively in the solution.
Orchestration Platform	An industry-standard orchestration tool is not supported outside of IaaS.	Industry standard orchestration tool supported through third party accelerators or other tooling. Not supported natively on the provider container solution.	One or more industry standard orchestration options are supported by the platform.

Table 1: Ratings Descriptions (cont.)

Docker Solutions on Amazon Web Services

Elastic Container Service (Container as a Service)

AWS offers a managed container orchestration service call Elastic Container Service (ECS). Amazon ECS is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.

Both Linux and Windows are supported with AWS in the form of a customized Amazon Machine Image (AMI). ECS dynamically selects an optimized Linux EC2 instance to run your containers. However for windows, an instance must be created up front. Windows 2016 with container support is available when launching your windows instance. One interesting note is that a PowerShell script is required for registration into a cluster when launching the windows instance but its documented in great detail here.

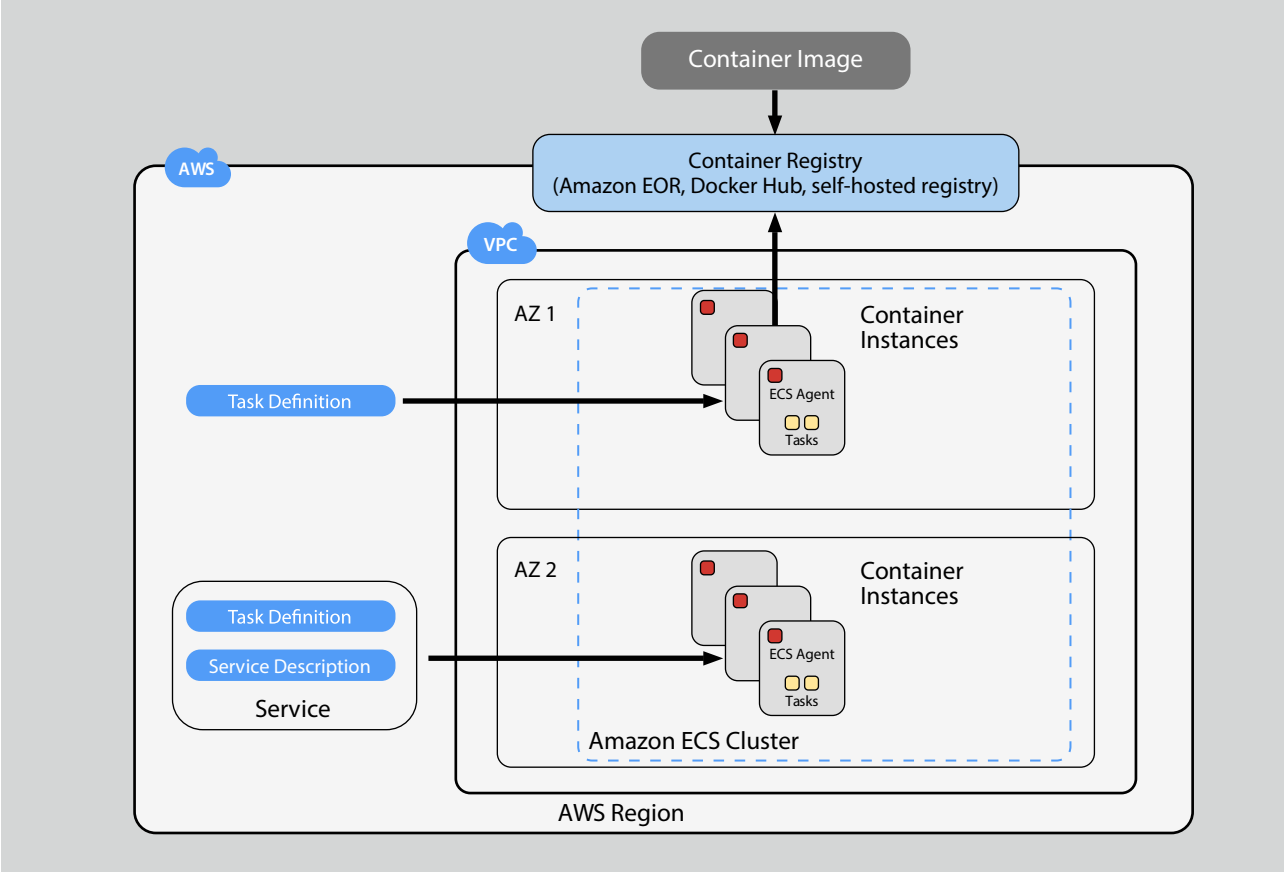
There are number of scheduling options available with ECS, but in order to run a container users must first create a task definition. Task definitions specify the container information for applications, such as how many containers are part of the task, what resources they will use, how they are linked together, and which host ports they will use. Tasks define your unit of deployment which can be a single container or an integrated set of related containers.

The service scheduler ensures that the specified number of tasks are constantly running and reschedules tasks when a task fails. Tasks can also be manually run, use a cron-like, or a custom open source scheduler like Blox. The service scheduler is also responsible for load balancing. This includes both the desired counts and registering and deregistering the instances with the load balancer. The load balancer can make routing decisions at the application layer with path-based routing and dynamic port mappings.

Dynamic discovery container services can be achieved by harnessing other AWS services including Cloud Trail, Route 53, and Lambda. The solution is quite extensive and is described here in greater detail.

ECS also monitors tasks, the state of the containers within the cluster as well as overall cluster state. Detailed logs of change events are sent to CloudWatch where additional actions can be performed. For example, events can be sent to Simple Notification Service (SNS) for addressing stopped task events. Lambda functions can listen for ECS events and performs some other actions. Two examples can be referenced here.

The ECS Service scheduler will also automatically recover containers that become unhealthy or stop running to ensure you have the desired number of healthy containers supporting an application.



CHARACTERISTIC	SUPPORT	SCORE
Scheduling and Orchestration	Fully supported	1
Routing and Load Balancing	ECS automatically routes requests to available containers via the Application Elastic Load Balancer.	1
Service Discovery	ECS includes new containers in the cluster when they are created. Dynamic discovery requires using other AWS service to update DNS entries.	0
Failover and Recovery	Fully supported	1
Cluster State	Fully supported	1
Container Deployment Unit	It can either be a single container or integrated set of related containers.	1
Orchestration Platforms	Multiple third party tools exist to enhance and ease the creation of AWS clusters using orchestration platforms. See Additional Orchestration Tools below for other options.	0
Logging	Fully supported vial CloudWatch and CloudTrail.	1
Monitoring	Fully supported vial CloudWatch and CloudTrail.	1
Composite Rubric		**7**

Table 5: AWS EC2 Rubric

Elastic Beanstalke (Platform as a Service)

Elastic Beanstalk (EB) is one of the AWS PaaS solutions that has the application as the focal point. You can build a classic three tier web application architecture without having to worry about the underlying infrastructure. EB handles the details of capacity provisioning, load balancing, scaling and application health monitoring.

AWS Elastic Beanstalk supports the deployment of web applications from Docker containers. With Docker containers, you can define your own runtime environment. You can choose your own platform, programming language, and any application dependencies (such as package managers or tools), that aren't supported by other platforms. Docker containers are self-contained and include all the configuration information and software your web application requires to run.

Runtimes for Elastic Beanstalk are similar to CaaS services. Linux and Windows are supported.
For Docker, specifically, you can choose either single or multiple container support, or pre-configured solutions including GlassFish, Go, and Python environments.

Docker images can be pulled from public registries including Docker hub, AWS EC2 Container Registry Service (ECR), and private locations such as an AWS S3 bucket.

For multiple container applications, Elastic Beanstalk uses Elastic Container Service (ECS) under the hood. It creates a cluster, and task definitions exactly like the CaaS solution works including scheduling, routing, load-balancing, fail over, recovery and monitoring. For the rest of this section on PaaS we will concentrate on single Docker containers and how they work with Elastic Beanstalk.

Single Docker containers within EB are deployed along with numerous other AWS services. Specifically Auto Scaling that can schedule scaling out or scaling in depending time of day, network and service load, or some other trigger event from within the EB echo system.

As before with scheduling, routing and load balancing are handled with other AWS services. Primarily the Auto Scaling engine that's attached to the EB environment for the application.

The Elastic Beanstalk solution handles service discovery but automatically updating load balancing when EC2 instances are added and removed.

Single Docker containers are managed like any other three tier architectures. The concept of a cluster is not present in this situation. State of the EC2 instances are monitored and EB provides insight into the health of the containers.

The unit of deployment is a single container per each EC2 instance.

Logging and monitoring for EB is like the CaaS solution. Events, logs, and all activities are sent to CloudWatch and CloudTrail.

AWS provide a command line interface to configure EB from many environments. See Configure the EB CLI and Managing Elastic Beanstalk Environments with the EB CLI for details.

CHARACTERISTIC	SUPPORT	SCORE
Scheduling and Orchestration	Fully supported. (Multi-container only)	1
Routing and Load Balancing	ECS automatically routes requests to available containers via the Application Elastic Load Balancer.	1
Service Discovery	ECS includes new containers in the cluster when they are created. Dynamic discovery requires using other AWS service to update DNS entries. (Multi-container only).	0
Failover and Recovery	Fully supported	1
Cluster State	Fully supported	1
Container Deployment Unit	Both single container and integrated set of related containers are supported.	1
Orchestration Platforms	Not supported using EB with containers.	-1
Logging	Fully supported vial CloudWatch and CloudTrail.	1
Monitoring	Fully supported vial CloudWatch and CloudTrail.	1
Composite Rubric	**Rubric assumes multi-container.**	**6**

Table 6: AWS EB Rubric

Additional Container Support

Docker Solutions on AWS

Elastic Container Service (Container as a Service)

AWS has other services and offerings that support Docker containers that are worth mentioning that bring greater support and control of your deployments.

Tools for Amazon Web Services

There are a wide variety of developer tool available for most AWS services for users that want to fully harness the power of building and managing containers that extend the capabilities found in the web console. This includes support for Windows, Linux, and OSX.

Additional Orchestration Tools

AWS has multiple third party partners and tools to enhance orchestration of Docker Containers including:

- Kubernetes Operations
- CoreOS Tectonic
- Kube-AWS

AWS EC2 Container Registry Service (ECR)

ECR is fully function Docker registry that you can push, pull, and manage images securely and reliably. Users can create repositories of images that have defined access controls based from IAM policies. AWS provides a command line interface to build containers and push them directly into ECR. As with all AWS services, activities performed with ECR can be sent to CloudTrail to keep a complete audit trail of API calls.

AWS provide a command line interface to configure ECS from many environments. See ECS CLI for details.

AWS Marketplace

The Marketplace is an online store for third parties to create and distributes solutions that extend the built-in offerings provided by AWS. Searching for "Docker" in the web console produces dozens Amazon Machine Image (AMI's) you can use to build custom solutions including subjects such as deep security, third party orchestration, and private repositories.

Docker on Google Cloud Platform

Google has been evangelizing containers as the solution for large-scale application deployment for over 15 years and has made significant contributions to open-source that have driven adoption of containers by others. All Google services are developed and managed as containers, including Gmail and YouTube. As proven by Google, containers are ready to run applications on truly global scale and with very high SLA requirements.

Docker Solutions on Google Cloud Platform

Google Container Engine (Container as a Service)

Google Container Engine (GKE) is a fully managed Kubernetes solution that brings advanced container orchestration and management capabilities to GCP. Kubernetes is itself the result of Google's own experience running and managing containers at scale, and has been open-sourced and provided to the community to advance the state of large scale container deployment. Many of the attributes of GKE are due to Kubernetes, and can be found in other provider's offerings. This makes Kubernetes a compelling solution for container deployment with no vendor lock-in; it is simple to redeploy an entire application to another Kubernetes platform, if a vendor specific database, for example, is not in use. Kubernetes is the only orchestration option that is supported by GKE.

At is most basic level GKE supports any Linux image that meets the Docker container specification, and that can be accessed via a published repository. Support is built-in for deploying from private GCR repositories, but others may be used as well with some additional configuration required to pull container images from non-GCR private repositories.

Containers are assembled and managed as groups of dependent containers named pods. Very often a pod will consist of a single container image, but the value of pods come when containers have a required dependency. For example, if a third-party logging solution is to be used for whatever reason, it makes sense that all containers need to be deployed with a logging container that deals with accumulation and delivery of logs to the third-party. To enforce this dependency, Kubernetes, and by extension GKE, deploys a pod that has been defined to be the application container and the third-party logging container. The pod will be scheduled and managed, ensuring the third-party logging container is always available to the application container.

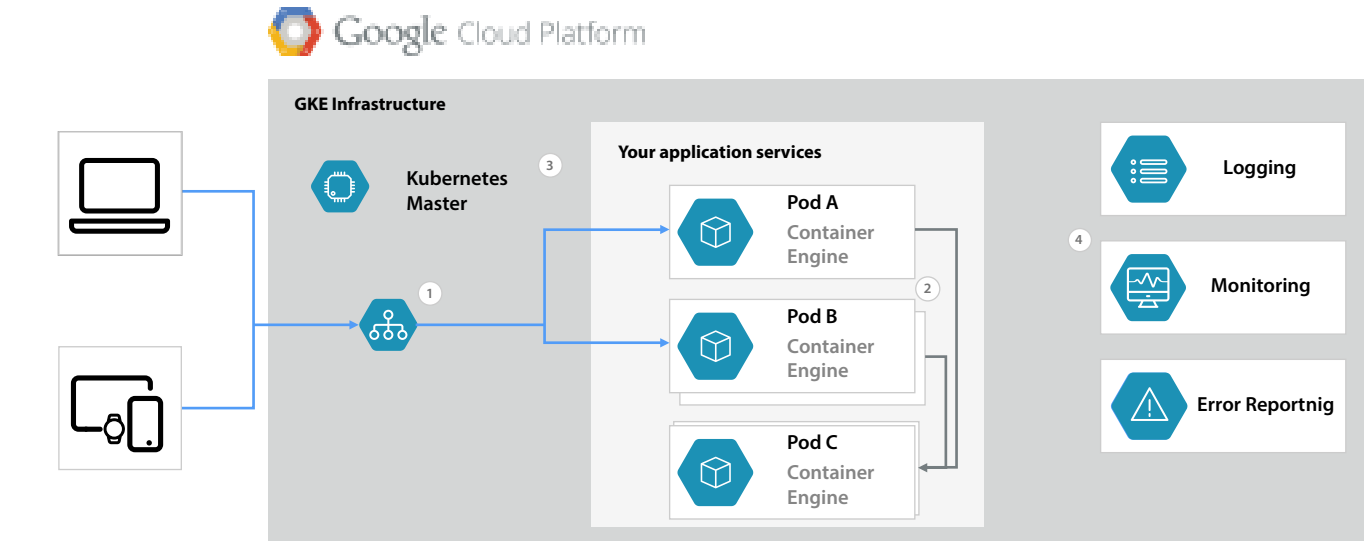
Strategies such as affinity/anti-affinity and tainting are available in GKE. Affinity and tainting are ways of defining constraints for the default scheduler such that pods are deployed in a manner to avoid each other or be logically close. For example, if your application contained two CPU-constrained services you can indicate that these should never be scheduled on the same node for performance reasons.

Regardless of how GKE chooses to distribute your pods, it has built-in support for routing and load balancing of services to make sure that the services exposed by the containers can be found and used. In addition, more complicated deployment scenarios can be applied on top of Kubernetes (and GKE) to allow for canary and blue-green type deployments to happen with minimum effort. Kubernetes has built-in service-discovery mechanism for services using private DNS that is automatically updated as services are deployed and redeployed. Combined with the use of namespaces, this simple but effective strategy allows for services to refer to each other using a simplified hostname which is resolved at runtime to a known running instance. Additional service discovery strategies may be layered on top of GKE using third-party solutions such as linkerd or istio as needed.

GKE monitors and automatically redeploys pods and services based on health checks. If a node fails, all pods on the node will be automatically redeployed such that there is minimal interruption to services. So long as enough instances are deployed, GKE will ensure that a minimum level of pods are running always. Short of a catastrophic failure, this all but guarantees a high SLA for your services. If there are problems, Stackdriver logging is enabled by default, and will collect all pod and container logs to a central location. You can review the logs using GKE web console or Kubernetes tools. Due to the very open nature of Kubernetes and GKE, third-party and alternate logging solutions can be used by disabling Stackdriver and adding the logging solution to pod definitions.

If needed, Stackdriver monitoring may be enabled on a GKE cluster. When enabled, an API may be called to inspect the current state of the cluster or the simple web UI can be used to review status. For applications where the application source code is stored in a supported version control system, and is in a supported programming language, Stackdriver even allows run-time inspection and debugging of live systems.

Architecture: Simplified Container Engine



1. Incoming HTTP requests are distributed to running containerized services
2. Pods are monitored and managed automatically based on rules
3. Master node is updated as pods go through their lifecycle, LB updated
4. Logs collected automatically, other tools can be added to support

CHARACTERISTIC	SUPPORT	SCORE
Scheduling and Orchestration	GKE fully manages all aspects of scheduling and orchestration and allows customization as needed	1
Routing and Load Balancing	Multiple strategies provided and can be customized to suit application requirements	1
Service Discovery	GKE automatically updates to changing pod deployments and node health	1
Failover and Recovery	Containers and nodes are health checked, with automatic recovery options	1
Cluster State	GKE abstracts all cluster management away from the application	1
Container Deployment Unit	Unit of deployment is a pod of one or more containers	1
Orchestration Platforms	Kubernetes is the orchestration option on GKE and is the underlying platform for the container offering.	1
Logging	GKE provides integration with Stackdriver logging, and other third-party solutions	1
Monitoring	GKE supports Stackdriver monitoring, and other third-party solutions	1
Composite Rubric		**9**

Table 6: AWS EB Rubric

Google Container Engine (Container as a Service)

Google APP Engine (GAE) provides a sandboxed environment for web applications that are automatically scaled and managed by the platform. Usually your development team will write an application in one of the supported languages (Go, Java, Node.js, etc.), and describe the runtime parameters with a simple YAML file. This YAML file is a concise instruction set that drives an automated container build and deploy process, compiling and packaging the application from source into a managed application.

Since GAE uses containers as the final deployment mechanism it also supports deploying containers directly from a registry, providing that the containers are configured to accept HTTP connections on port 8080. This is a key limitation when deciding to use GAE for container deployment; the container must expose an HTTP listener on port 8080 or GAE will mark the deployment as failing, since health checks are via the same exposed port and a fixed relative URL of /_ah/health.

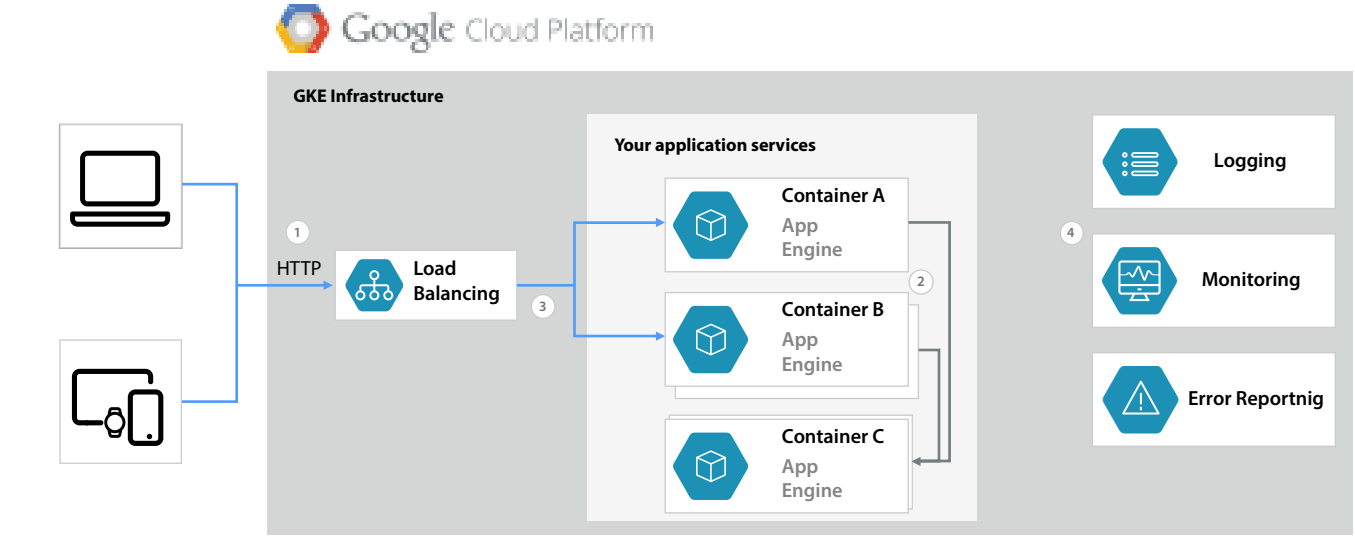
Note: GAE is broad in its definition of "healthy"; any HTTP response code that is NOT 502, 503 or 504 will be considered healthy.

Another limitation that will be evident when deploying a micro-services application is that intra-service discovery is non-existent in GAE; if a service in Container A needs to communicate with Container C, as shown below, then the URL for Container C must be embedded in service deployed to Container A. Finally, GAE does not support or provide access to the underlying orchestration platform; all decisions about container placement and affinity is made by GAE and cannot be influenced by the application.

If your application can function with these restrictions, then GAE makes for a compelling solution for auto-managed containers. GAE will handle all scheduling and routing, scaling the number of containers to match usage constraints such as CPU utilization or network throughput, and updating load balancer appropriately. A nice feature is a built-in versioning scheme that allows rolling deployments of container updates; when it is time to launch a new version of your services, GAE provides a mechanism to rollover new requests, while preserving access to prior instances for live HTTP connections.

GAE automatically logs request and response details, and captures entries anything written to the container stdout and stderr streams. The logs are kept for a maximum of 90 days, and are automatically truncated to a maximum size of 1 GB. In addition to application logging, the logs generated by GAE's monitoring framework are available. For alerting, third-party services can be used to determine service availability, or the log files may be exported into other GCP products such as BigQuery or Pub/Sub for additional processing.

Architecture: App Engine with Containers



1. Incoming HTTP requests are distributed to running containerized services
2. Pods are monitored and managed automatically based on rules
3. Master node is updated as pods go through their lifecycle, LB updated
4. Logs collected automatically, other tools can be added to support

CHARACTERISTIC	SUPPORT	SCORE
Scheduling and Orchestration	App Engine fully manages all aspects of scheduling and orchestration	1
Routing and Load Balancing	Incoming HTTP traffic is automatically routed to port 8080 on containers	1
Service Discovery	GAE does not provide intra-service discovery, but does update routing and load balancing based on container lifecycle	0
Failover and Recovery	Containers are monitored and restarted automatically when problems are detected	1
Cluster State	Provided natively by the platform and invisible to the implementation	1
Container Deployment Unit	Unit of deployment is a single container instance	1
Orchestration platforms	GAE does not provide any access to other orchestration platforms	-1
Logging	Partially supported through log forwarding or whatever	0
Monitoring	GAE allows the inspection of log files from monitoring framework	0
Composite Rubric		**4**

Table 8: Google App Engine Rubric

Additional Container Support

GCP provides access to the open-source Docker engine through managed services by using existing OS images with Docker pre-installed. Docker customers that prefer to use an installation managed by familiar Docker management tools can look at Docker CE for GCP[1] , currently in beta. Google Cloud Platform does not provide an out-of-the-box OS image with a commercially supported Docker instance. You must launch a Windows Server VM, or Linux VM, and install Docker manually to have a supported version available for use. GCP provides alternative tools that can be used in place of standard Docker offerings; while not required, these tools are closely integrated into the GCP ecosystem and management suite.

Google Container Registry (GCR)

Google Container Registry is a Docker-compatible registry that supports private (default) or public repositories for Docker containers. While it is possible to use other container repositories, such as Docker Hub, GCR is integrated with GCP authentication and authorization framework and you can configure access to the registry using the same IAM mechanisms that are used to control access to the GCP project.

Google Container Builder

Google Container Builder is a build tool that produces Docker containers that can be automatically triggered by changes in source code. While not a full replacement for a tool such as Jenkins or Drone, container builder is an alternative to installation and configuration of those tools that is provided by GCP.

Container-Optimized OS (COS)

Container Optimized OS is a minimal Linux OS based on Chromium OS that is designed to run Docker containers as a first-class option. Instead of installing applications using a package manager, containers can be pulled directly from repositories and run at startup. All configuration is provided during startup via cloud-init parameter, a YAML formatted string that describes how the instance(s) are to be configured during boot before accepting network connections.

In many ways, Container-optimized OS fills the same niche as CoreOS or Mesosphere DCOS, but is optimized to run in the Google Cloud Platform environment. Container-optimized OS is integrated into GCP's IAM framework and other GCP services that make it easier to deploy containers on COS. Container-optimized OS is used as the foundation for all GCP managed container solutions.

[1]https://blog.docker.com/2017/03/beta-docker-community-edition-google-cloud-platform/

Docker on Azure

Microsoft has embraced open source solutions across the organization and is reflecting this throughout their major platforms. Linux has becom a first class platform in Azure. SQL Server runs on Linux and .Net Core has not only been open sourced, but also is built to run on Linux. This shift in direction has led to bringing in support for containers, especially as their popularity grew. Microsoft's commitment to containers can be seen in the major investment needed to bring Windows Containers to the Windows Server 2016.

Docker Solutions on Microsoft Azure

Azure Container Service (Container as a Service)

Microsoft leverages mature and popular management and orchestration platforms for containers through Azure Container Service (ACS). This approach is similar to Google Cloud utilizing Kubernetes, although ACS allows for multiple potential platforms. ACS currently supports three modes; DC/OS (Mesos), Docker Swarm, and Kubernetes. Each mode uses the named orchestration platform, backed by the scalability and high availability of Azure through scale sets.

ACS provisions a Docker cluster in an Azure VM Scale Set secured with TLS by default. Azure VM Scale Sets provide autoscaling (depending on the mode), load balancer, storage, and NAT rules. Rules can be applied to assist with the management of routing and load balancing. The lifecycle of nodes is managed using auto-scaling groups or similar constructs, so that if a node enters an unhealthy state for unforeseen reasons, the node will be taken out of the load balancer rotation and replaced automatically. All of its container tasks will be rescheduled.

Service discovery utilizes the internal discovery mechanisms of the orchestration platform. However, several Azure features can also enhance service discovery. Additional nodes can be added outside of the orchestration platform through the ACS console and the platform will manage adding the node to the cluster. Azure creates a secure network and instance configuration through scaled sets along with secure deployment of public endpoints.

VM sets are leveraged for cluster state, although the orchestration platform (mode) can also provide insight into the state of the cluster. For example, when Docker EE is utilized, the internal cluster manager is the primary method to administer the cluster state. ACS can use OMS for monitoring the cluster state as well which gives a single location for all management and health issues. ACS also has an open API to gain insights into the health of the containers. More proactive monitoring methods such as alerts can be configured through the Azure portal in addition to OMS or pulling from the API.

Docker on Azure

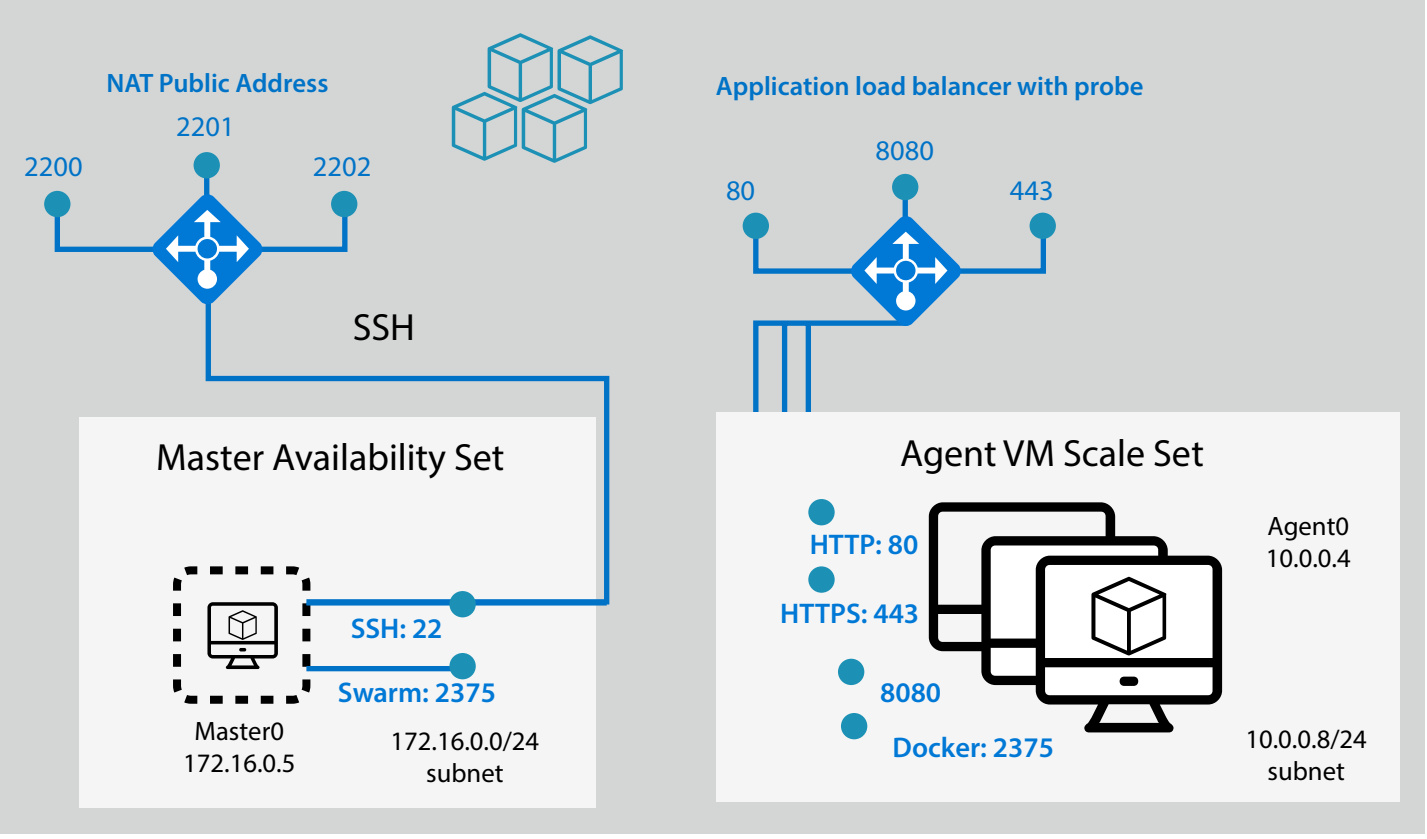
ACS can be configured using both the CLI and Azure portal. Since ACS utilizes VM's, many of the same features can be configured for container instances. The deployment can be done in an automated fashion using templates to define instance size and networks. Currently, the service can support agent nodes between 1 and 100. Azure cores quota can also be used to limit the number of agent nodes in a cluster. Agent node scaling operations are applied to an Azure virtual machine scale set that contains the agent pool. In a DC/OS cluster, only agent nodes in the private pool are scaled by the operations. Depending on the orchestrator, clusters can separately scale the number of instances of a container running on the cluster. Currently, autoscaling of agent nodes in a container service cluster is not supported.

Centralized logging is a critical component of many modern infrastructure stacks. ACS can forward logs from containers to a native cloud provider abstraction or a storage account. Log rotation is configured for you automatically, so chatty logs won't use up excessive disk space. Likewise, the "system prune" option allows you to ensure unused Docker resources such as old images are cleaned up automatically.

Lifecycle events and performance counters from Docker containers can be charted on Application Insights. Install the Application Insights image in a container on the host, and it will display performance counters. Installing Application Insights image on your Docker host provides these benefits:

- Lifecycle telemetry about all the containers running on the host - start, stop, etc.
- Performance counters for all the containers. CPU, memory, network usage, and more.

The standard Docker ACS deployment is shown in the image below.



Azure Container Instances (Containers as a Service) – Public Preview

Azure Container Instances (ACI) is a fast container deployment service that removes the complexity of setting up a cluster. With ACI a developer can run a container in Azure, without having to manage any VM (virtual machines). ACI intentionally does not provide a full orchestration and management solution to stay with its intent of being a fast and simpler option for running Docker solutions.

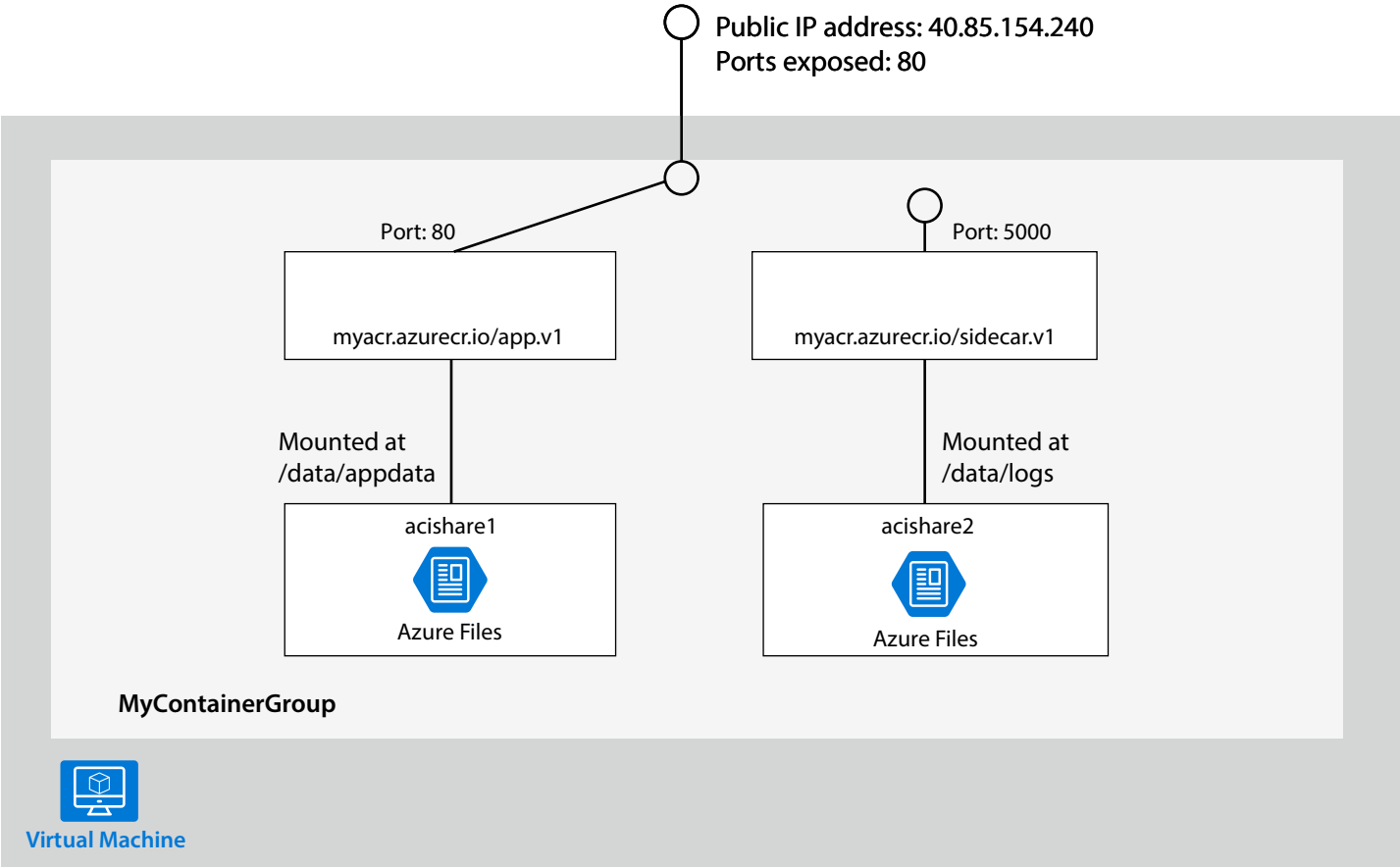
CHARACTERISTIC	SUPPORT	SCORE
Scheduling and Orchestration	Fully supported via ACS	1
Routing and Load Balancing	Azure Container Service automatically routes requests to available containers	1
Service Discovery	Azure Container Service includes new containers in the cluster when they are created	1
Failover and Recovery	Probes and failures are managed at the platform level and container instances restarted in the case of failures	1
Cluster State	Provided natively by the platform and invisible to the implementation	1
Container Deployment Unit	Unit of deployment is a single container instance	1
Orchestration Platforms	ACS provides a choice of Docker Swarm, Docker Enterprise Edition, Kubernetes, DOC/OS for orchestration	1
Logging	Fully supported through Azure portal and Azure OMS	1
Monitoring	Fully supported via Azure portal and OMS	1
Composite Rubric		**9**

Table 8: Google App Engine Rubric

Azure Container Instances target any scenario that can operate in independent containers, like simple applications, task automation, and build jobs. Although ACI limits its target scenarios, it still provides a solid feature set such as:

- Fast startup times
- Hypervisor-level security
- Custom sizes
- Public IP connectivity
- Persistent storage
- Linux and Windows containers
- Co-scheduled group
- Azure Active Directory integration

ACI can use either a single instance or a container group. Container groups are like the Pods concept described previously for the Google Container Engine. The group shares local network, storage, and other resources and are hosted on the same virtual machine. The group can expose one public IP Address to allow external access. Below is an example container group. This approach allows an application or API to break up functionality into different sub-systems based on separate Docker images. Instances can still communicate with each other in the group while sharing resources. Below is a logical view of an example container group using a web endpoint with a supporting logging instance.



Logging and monitoring options are similar to ACS.

Since the target use case for ACI involves a simple managed deployment of a Docker instance or group, many characteristics reviewed in this document are not included which is by design. For example, scheduling, orchestration and load balancing are not built into ACI. Integration with ACS will allow layering over ACI to achieve these requirements if needed. The layering capability is in early stages and expected to be complete before ACI exits preview.

In other cases, some characteristics are only partially supported by design. Routing of a public IP address to a container instance in the group and discovery of other instances in a group through localhost is supported, but not full load balancing. Also, the idea of groups, similar to Pods in Kubernetes is supported, although the design of a single instance or group deployment limits the need for more complex features like server affinity.

CHARACTERISTIC	SUPPORT	SCORE
Scheduling and Orchestration	Not supported directly by design. Orchestration and management can be layered on top of ACI through ACS although this feature is not production ready.	0
Routing and Load Balancing	Azure Container instance automatically routes to containers as part of the deployed service. Load balancing is not supported as only one instance or group runs at a time.	0
Service Discovery	Azure Container Instance allows other container instances in a group to discover each other through localhost and port mappings	0
Failover and Recovery	Containers in the service will be restarted if failed	1
Cluster State	State is available through the portal or Azure API. Keep in mind ACI is designed to contain one instance or container group and not a scaled out cluster.	1
Container Deployment Unit	Azure Container Instance supports container groups, although features are limited due to the nature of a single group deployment	0
Orchestration Platforms	Not supported directly by designed. Orchestration and management can be layered on top of ACI through ACS. This feature is not production ready.	0
Logging	Logging is integrated to the platform and can be managed through Azure CLI.	1
Monitoring	Monitoring can be done through the Azure Portal and integration with OMS.	1
Composite Rubric		**4**

Table 10: Azure Container Instance Rubric

Additional Container Support

Microsoft Azure provides a number of other services and tools that support Docker containers worth mentioning and are listed below.

Docker for Azure

Docker for Azure is a Docker-native solution optimized to work on the Azure platform utilizing the underlying Azure IaaS services and does not require any additional software to be installed. Latest Docker platform versions with native orchestration (clustering and scheduling), runtime security, container networking and volumes are all included with Docker for Azure. Both Docker Community Edition and Enterprise Edition are supported in this deployment strategy.

Azure Marketplace

The Azure Marketplace contains a number of Docker-specific solutions that will augment existing Docker solutions or create prescribed Docker solutions based on best practices.

Azure Container Registry

Azure Container Registry Services are a great way to share Docker images. Organizations need to trust the images they are loading into production. Images need to be certified and approved. Azure Container Registry is Microsoft's solution to these and other challenges. Azure Container Registry can host private company repositories for trusted container images. ACR supports the standard Docker Registry v2, so pushing and pulling images can be integrated with existing tools and platforms.

CI/CD with Azure Container Registry

One of the biggest challenges when developing modern applications for the cloud is supporting continuous delivery. Azure and Docker provide a path to implement full continuous integration and deployment (CI/CD) pipeline using Azure Container Service, Azure Container Registry, and Visual Studio Team Services.

Conclusion

Declaring a clear "winner" for the flagship container service offering would be difficult based on the maturity of each cloud provider in supporting Docker. Each cloud provider also takes a different approach to their container offerings. GCP naturally chose the Kubernetes platform to run GCE which was born out of their deep history of solving challenging problems to support other services from their global business. AWS leveraged their EC2 platform - a mature, proven, and reliable solution - and then added additional capabilities forming their container offering. Microsoft went with a highly-flexible model, looking to support many of the most popular existing orchestration and management platforms and supplementing or enhancing features with the Azure platform.

Each provider does provide Docker support through existing Platform-as-a-service offerings or a specialized use case such as Azure Container Instance. In these cases, scoring for the rubric are generally lower. In the case of the PaaS offerings from GCP and AWS, this is intentional as Docker serves more as a deployment mechanism of the code base. The PaaS services also support deploying code directly to be hosted and managed, so this further solidifies the PaaS service intention to be more than a container service. Core scaling capabilities will be available, but specific container orchestration concerns are generally not built-in. In the case of Azure Container Instance, the expected use cases are simpler and instance focused, so many features are excluded on purpose. Users should understand the purpose of the service and not expect a full container orchestration platform if these services are used.

CHARACTERISTIC	GCP-CCE	GCP-AE	AWS-ECS	AWS-EB	Azure-ACS	Azure-ACI
Scheduling and Orchestration	1	1	1	1	1	0
Routing and Load Balancing	1	1	1	1	1	0
Service Discovery	1	0	0	0	1	0
Failover and Recovery	1	1	1	1	1	1
Cluster State	1	1	1	1	1	1
Container Deployment Unit	1	1	1	1	1	0
Orchestration Platforms	1	-1	0	-1	1	0
Logging	1	0	1	1	1	1
Monitoring	1	0	1	1	1	1
Composite Rubric	**9**	**4**	**7**	**6**	**9**	**4**

Table 11: Summary View of Rubric

Conclusion

Each provider has reached a solid level of maturity for their core container offering, so at the level of review done in this document, there will not be significant feature gaps. There will likely be some differentiation reached in each area if we further drill into level 300 and 400 details, although those gaps will likely be closed by each provider over time. In sum, being comfortable with the approach of the provider is likely going to be a significant driver in the decision making, especially how that solution aligns with existing skill sets or current approaches to managing containers in an organization.