Wintellect
Know how.

Understanding
Azure Container Solutions

A Synopsis of Container Solutions on Azure

Microsoft has incorporated containers into many Azure services but figuring out where containers fit into various types of solutions is no simple task. This document provides a better understanding of when and how to use specific container solutions and how they integrate with Azure as a whole.

Blaize Stewart
Azure Architect, Azure MVP

March 2020

Contents

1 Synopsis

Containers are no longer one of the best kept secrets in IT. According to a survey by Portworx and Aqua, nearly 90% of organizations use containers in some manner. Containers can be used for everything from hosting a simple website to applying machine learning models to data coming out of devices and everything in between. Ecosystems have grown up around containers to support these workloads, some general, some niche. Microsoft has incorporated containers into all kinds of solutions on Azure, from simple things like hosting a blog to managing entire datacenters on a container platform. Since multiple, often overlapping solutions can be found on Azure, either first-party or third-party, getting the lay of the land is sometimes difficult.

While many guides exist for point solutions on Azure, as well as more general guides within specific use cases, their disparate nature does not provide a synoptic view of the container landscape on Azure (or any cloud for that matter) in a way that illustrates how containers overlap and integrate with one another and with Azure. The goal of this document is to help the reader get a better understanding of which solutions to use for a given use case and how these solutions integrate within the larger scope of Azure, not merely within a given solution.

1.1 How to Use this Document

This document is not intended to be read from start to finish; rather it is intended to be read to get specific information as needed.

- Section 2 talks about general container principles and terminologies.
- Section 3 discusses different container workloads, then offers a list of Azure solutions that can be applied to each.
- Section 4 contains an overview of each of the Azure solutions mentioned in Section 3 with links to resources for further learning.
- Section 5 compares and contrasts solutions.
- Section 6 gives an overview of supporting services that can be applied to all the Azure solutions mentioned in Section 4
- Section 7 concludes the document.

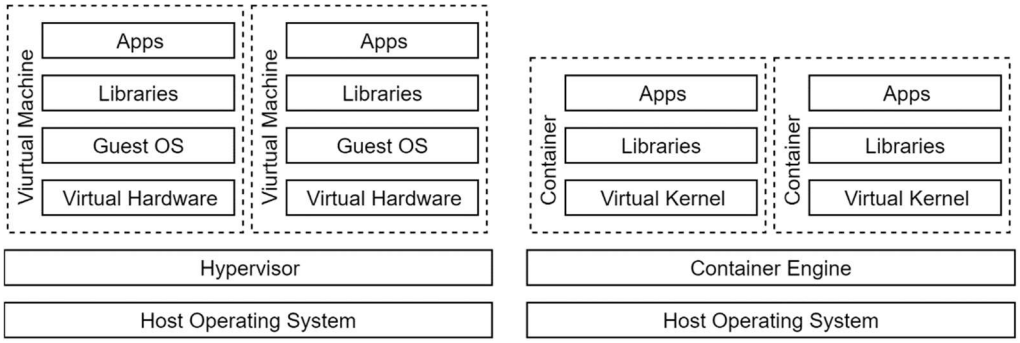
Those new to containers should read Section 2 first to get a primer on container technology and workflows. Next, read Section 3 as it relates to your particular use case. Finally, read Section 4 based on the Azure solutions found in Section 3.

2 What is a Container?

Containers are a virtualization technology that works at the operating system level. One helpful way to understand containers is to compare them to virtual machines.

Virtual machines are virtual computers provided by virtualized hardware. All the necessary components to make a computer work are represented by an abstraction layer. Instead of a physical CPU, RAM, hard disk, and so on, each is represented by a software-based virtual component. Together, the virtual machine has all the same components a physical machine has, and an operating system can therefore run on the virtual hardware. Virtual machines are composited and managed on top of a host operating system through a hypervisor.

Containers are also an abstraction layer. The primary difference is that containers do the virtualization at the kernel level within an operating system. Kernel level virtualization APIs are used by applications to run apps, such as a filesystem, security, networking, device input and output, process management, interprocess communication, and other tasks. Within the container, these APIs act as a stand-in for the operating system kernel. Containers on a host operating system are managed by a *container engine*.



Both containers and virtual machines are delivered as *images*. An image is a file or set of files that contain an entire filesystem (and thus all the files) for the virtual machine or container. Images create instances of whatever is contained within the filesystem, which for virtual machines would be an operating system, libraries, and applications; a container typically only contains libraries and applications.

The main advantage of containers over virtual machines is that fewer system resources are required to create and maintain an instance. Containers eliminate the need to have the operating system stored and loaded, which reduces RAM, CPU, and storage requirements. This improves application density.

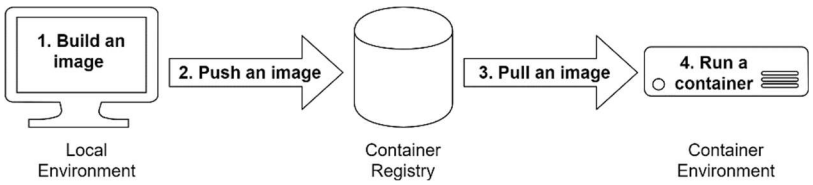
The tradeoff of using containers, of course, is that they require an operating system. Linux containers need a Linux host; Windows containers need a Windows host. A virtual machine can run an operating system different than that of the host, Windows can run Linux virtual machines and vice versa.

2.1 Docker

Docker can refer to the tools used for creating container images, an environment used for running Docker containers, or the company behind the Docker container tools, runtime, and image format. Docker containers are the de facto standard, with over 99% of containers in use being this type. Typically, when somebody is talking about containers, they are talking about Docker containers.

2.2 Container Workflows

Container workflows all follow the same basic pattern from getting an application into a container image to deploying that application in a container instance.



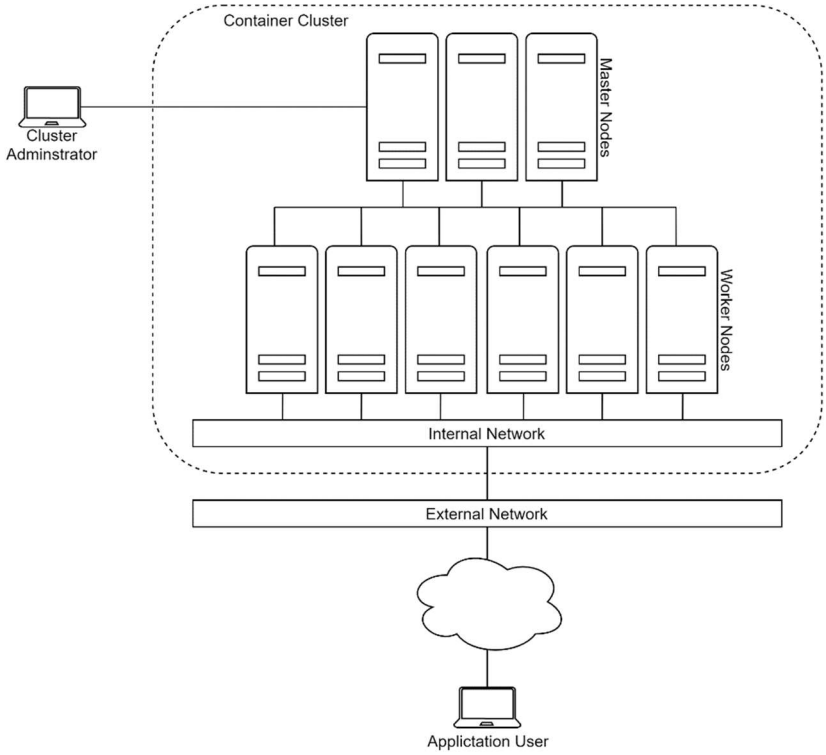
1. ***Build* a container image**. Building an image starts with a script that contains instructions the container runtime needs to build the image. In the Docker context, this script is a *Dockerfile*.

Once a container image is built, that image exists in a local context. Container builds can be done "from scratch," meaning that the image contains nothing other than an empty file system, or they can be built from preexisting images. New images share file system layers with the base image to reduce their size, rather than creating completely new copies.

2. ***Push* a container image to a container registry**: A *container registry* is a centralized repository for container images that can be accessed by any agent that needs to use the registry. Registries can be either public or private.
    - *Public registries* make container images available to any user anonymously who can reach the registry over a network.
    - *Private registries* make container images only available to those who first authenticate against the registry.
3. ***Pull* a container image from a container registry:** Pulling an image simply means to download a container image from its registry into a runtime environment. (Pulling happens also at build time when using base images.) Pulled images are then cached in a runtime environment's local container registry.
4. ***Run* a container image as a container**: The fourth and final step is to deploy an image as a running container. This is typically accomplished by an orchestration engine. Multiple container instances can use the same image.

2.3 Container Orchestration

Container orchestration is the deployment and management of containers at scale (the last two workflow steps). Rarely, if ever, does an application use a single container for production-oriented workloads. Workloads are typically composed of multiple components, each with its own redundancy requirements, to ensure the application as a whole is highly available. A container orchestration engine is needed to manage applications at scale



Container orchestrators typically deploy a cluster of compute nodes. Those that manage the cluster are called *master nodes*. Others host the containers that compose applications; these are the *worker nodes*. To make applications highly available and scalable, container orchestration itself is typically highly available with multiple master and work nodes. A cluster administrator or developer interacts with the master nodes through a client that tells the master nodes how to orchestrate the worker nodes. Application users interact with containers running on the worker nodes through network connectivity provided by the orchestration environment.

Container orchestrators provide much of the infrastructure needed to run and manage containers at scale, including:

- Pulling containers from container registries and maintaining local registries so containers can be deployed rapidly.
- Deploying containers into worker nodes to do the work of an application.
- Automate scheduling for certain kinds of container workloads.
- Monitoring containers for health probes or other methods used to bring down unhealthy containers or increasing container counts to meet the desired thresholds for an application.
- Scaling container nodes for applications up and down based on metrics such as CPU, RAM, and I/O and distributing the load across available worker nodes.
- Providing redundancy for containers across multiple nodes to ensure application availability in the event of a cluster node failure.
- Migrating containers from one node to another in the event of resource exhaustion.
- Exposing containers to internal and external networks
- Load-balancing network traffic across container deployments.
- Providing connections to external storage for data persistence outside of containers.

- Maintaining access to secrets used by containers.
- Providing configuration to containers so containers can appropriately operate.
- Securing container environments – for example, isolating workloads and providing Role Based Access Control (RBAC) to a container environment.

Not all container orchestrators perform all these tasks because they may not be required for its platform workload. More robust, general-purpose platforms perform all of these and perhaps more, depending on the platform.

2.4 CaaS

Containers as a Service (CaaS) from cloud providers typically present the environment for container orchestration as a service, so that the user does not have to maintain the environment. These services invariably obfuscate some of the more complex details of a container cluster, and in some cases the user of the service is not even aware of their presence. The user can then focus on configuring the workload to run on the container environment without having to worry about the platform.

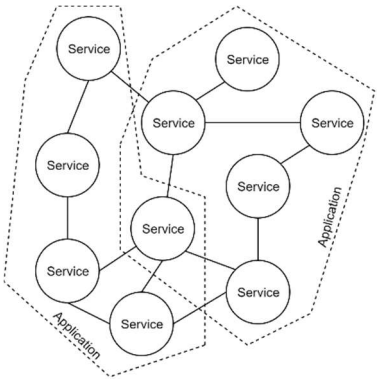
3 Container Use Cases

Containers provide a way to encapsulate applications in a lightweight package that can be rapidly created, moved, and deployed in multiple environments. Because the abstraction in the container provides a clear separation of concerns at the kernel level, containers do not prescribe any specific environment or stack for them to run. The only requirement is a container engine. For this reason, containers can be deployed in a wide range of environments for a wide range of purposes.

3.1 Microservices Architecture

Microservices architecture is an approach to building applications in which the application is broken into multiple independent components. These components provide various services, such as data mailing, to other application components. There is no universally accepted definition for a microservice, but they can be described by a set of principles:

- Microservices may be shared among multiple applications of different types. An application is defined by a set of microservices, but the microservices within that application might also be part of another application.



- Microservices communicate over a network through lightweight protocols (such as HTTP) or may be brokered by an asynchronous messaging platform (such as Azure Service Bus) that decouples services and prevents them from having tight dependencies.
- Microservices are independently deployable; a given microservice does not require a complete redeployment of an entire suite of services.
- Microservices are independently scalable. Any one service can be scaled without having to scale all other services in the microservice suite.
- Microservices can be implemented using a variety of technologies, such as runtime stacks, databases, and supporting libraries.

Container technology supports microservice architecture well; using containers, an application can be broken into multiple components without much additional overhead. These components then can be encapsulated and deployed as independent containers into a container environment. The application then is defined by a suite of containers. The container environment, through orchestration, can be the infrastructure to facilitate networking, monitoring, scaling, healing, and security for microservice-based applications.

3.1.1 Azure Solutions

Azure provides two primary services for running microservices: Azure Kubernetes Services (4.3) and Azure Service Fabric (4.5). Microservices can also run-on third-party offerings such as DC/OS (4.7), Docker CE/EE in Swarm mode (4.6), or Azure Red Hat OpenShift (4.4).

3.2 Internet of Things

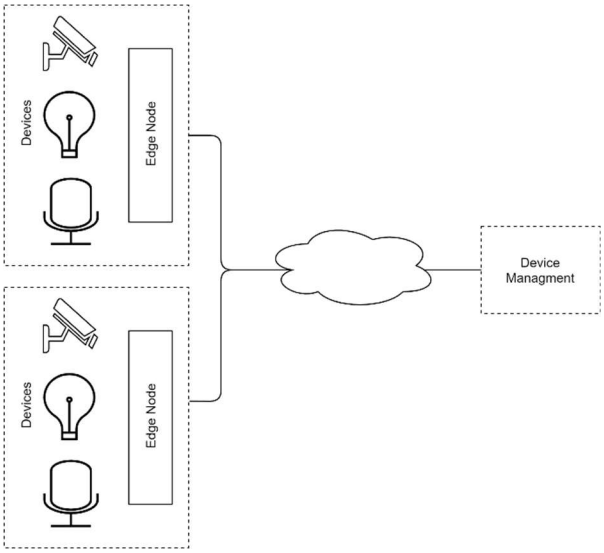
The Internet of Things (IoT) is loosely defined as devices that send and receive data over networks principally without human interaction. IoT devices usually do not have human interface devices (keyboards, touchscreens, and so on) so the category excludes such things as smart phones, laptops, and televisions. Other examples include security cameras, thermometers, IR sensors, traffic signals, electric/water/gas meters, smart lights, thermostats, motion detectors, and door sensors.

IoT deployments typically have device counts orders of magnitude greater than those of the population that is using them. A smart home, for instance, may only have four or five occupants but may have twenty or thirty devices such as smart bulbs, door sensors, cameras, and thermostats. Managing these devices at scale requires a whole new approach to computing and orchestration. IoT management is responsible for:

- Registration when the device is shipped from the factory.
- Provisioning and setting up a device once it is deployed.
- Authentication and authorization to send and receive data securely to and from appropriate sources.
- Configuration to get and change appropriate settings and apply updates as needed.
- Monitoring the status and health of a device.
- Troubleshooting to help remedy problems as they arise in devices.

The complexities introduced by managing devices at scale created the need for solutions that allow quick responses to software changes on the devices and within the supporting infrastructure. This involves two major architectural components:

- **Centralized management usually in a cloud-based solution:** A centralized management solution allows for device registration and device provisioning and the rest of the functionality for a specific customer.
- **Edge computing:** In a local context, edge computing brings IoT management closer to the devices to provide better latency and redundancy as well as protection against network outages.



Containers provide core functionality and extensibility either on the devices themselves, to edge computing nodes, or both. Many devices run a full operating system, typically some version of Linux tailored for the device. The OS permits customer-specific applications through containers. Edge computing also allows for customer specific code to be deployed to these devices.

3.2.1 Azure Solutions

Azure IoT Edge (4.11) offers extensibility through containers. IoT Edge is used in conjunction with Azure's IoT suite such as Azure Device Provisioning Service (https://docs.microsoft.com/en-us/azure/iot-dps/about-iot-dps) and Azure IoT Hub (https://docs.microsoft.com/en-us/azure/iot-hub/).

3.3 Cloud-Agnostic and Multi-cloud Apps

Cloud service offerings are said to be "opinionated:" the more managed a service is, the more assumptions it has. At some point, some of these assumptions require serious workarounds or accommodations. Less managed services, such as virtual machines, still carry with them certain assumptions that vary from vendor to vendor. In principle, cloud agnosticism remains to the degree possible independent of any assumptions baked into the cloud provider's platform. Pure cloud agnosticism is impossible. It is possible to abstract away assumptions to make cloud agnosticism at least manageable from an application deployment perspective.

Containers as a technology are an open standard. This implies that containers are by their nature cloud agnostic. The container separation at the kernel makes developing and offering Container as a Service (CaaS) common among cloud providers, often with multiple options within a cloud provider. Containers can then be deployed to any number of these CaaS without regard for the vendor or location. Moreover, ISVs can run their software in whatever cloud, clouds, or mashups an organization adopts.

- Public clouds refer to a multitenant offering from a large vendor to organizations and individuals through a self-service model. Prominent examples include Microsoft Azure, Amazon Web Services (AWS), and Google Compute Platform (GCP).
- Private clouds are typically single-tenant clouds for a single organization. A private cloud would be composed of all the organization's compute assets spanning all its own datacenters.

- Multicloud clouds span multiple public cloud vendors. These deployments are designed to protect against vendor lock-in and outages.
- Hybrid clouds are a composite of public and private cloud resources that deliver services between the public and private components seamlessly. The components typically share a high-speed connection between the private and the public cloud components.

The permutations of cloud offerings make it challenging to deploy applications that do not require a specific vendor or service. Containers mitigate this by including necessary runtimes and libraries in the container as well as clearly differentiating between a platform and the container.

Many cloud vendors offer a proprietary container orchestrator that can make container deployment easier, but these are not available on all clouds. Using open-source container orchestrations gives some level of uniformity to container environments.

3.3.1 Azure Solutions

Azure Kubernetes Services (4.3) is a standard Kubernetes deployment using managed infrastructure. Because Kubernetes is open-source, it is also offered on other clouds as a service and can be deployed on-premise as well. It is the most popular platform that is not vendor specific.

For managing applications deployed across hybrid clouds and multicloud deployments, Azure offers Azure Arc (6.2), which extends Azure management services such as RBAC and policy compliance beyond the borders of Azure to VMs and Kubernetes clusters.

3.4 "Lift and Shift" Apps, App Migrations, and Legacy Apps

Application migrations from on-premise systems to clouds typically involves moving applications from a legacy environment to a Platform as a Service (PaaS) offering that eliminates the need for virtual machines. One of the biggest drivers for this is the value added by cloud environments through service management, which eliminate the overhead required to maintain virtual-machine deployments.

When moving applications, there can be a gap between what PaaS will support and what an application needs to run. Making some applications run in a particular PaaS offering might require minor or even major refactoring to make the application work within the assumptions of the PaaS offering. Major refactoring to applications is more likely in legacy applications. In some cases, it might not even be possible because the source code is unavailable. The default, then, for many organizations is to "lift and shift" entire virtual machines from on-premise datacenters to the cloud by way of virtual-machine migrations.

Containers offer a middle ground between full PaaS offerings and moving virtual machines to the cloud. Containerizing applications require that the platform running an application be refactored, but do not necessarily require that the application itself be refactored to run in the container. Thus, one can lift and shift the application to a container without the need to touch source code or make any changes to the application itself. The container will also carry with it all the supporting libraries for the application that might not be available on a PaaS offering. Containers eliminate the need to carry along an operating system as well. Once an application is containerized, it can run in one of many different Containers as a Service (CaaS) offerings on a cloud, which are PaaS offerings designed for running containers, and take advantage of the value added by using PaaS.

Whether to lift and shift entire virtual machines or lift and shift applications from virtual machines to containers is weighed between the effort to migrate and manage the virtual machines against the effort to refactor and migrate applications to containers and run those containers in a PaaS offering.

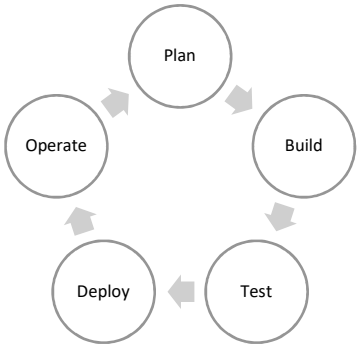
3.4.1 Azure Solutions

Azure provides several general solutions for running containerized applications, including Azure Container Instances (4.1) and Azure Kubernetes Services (4.3). Web apps can be easily run on Web Apps for Containers which is built on Azure App Services (4.2). Third-party offerings for running lift and shift apps and legacy apps include Docker CE/EE (4.6), DC/OS (4.7) and Azure Red Hat OpenShift (4.4).

3.5 DevOps

"DevOps" is a portmanteau of "development" and "operations," two historically distinct units within IT. DevOps is a philosophy about building a collaborative culture between organizational silos. In doing so, DevOps aims to improve organizational agility and quality. Practically speaking, DevOps practitioners implement a process using purpose-built tooling. The tools provide much of the infrastructure for automation in building, testing, releasing, and managing software as well as collaboration at every point in the process.

DevOps applied to software is one philosophical approach to a Software Development Life Cycle (SDLC) that focuses on ongoing software development, or in broader terms Application Lifecycle Management (ALM), which focuses on application management in general, which may or may not include software development. There is no universally accepted set of steps that define SDLC, but generally speaking it includes:



1. **Plan:** Teams assess problems that need to be fixed or list features that need to be added or changed.
2. **Build:** Teams make the changes, usually by generating new code or changing existing code.
3. **Test:** Teams perform unit tests, integration tests, regression tests, and user acceptance tests against the code.
4. **Deploy:** Once tests are passed, teams deploy the changes into a production environment.
5. **Operate:** Once changes are in production, the environment is monitored, and feedback is sent back to the team to go through the next iteration in SDLC.

Containers support SDLC and thus DevOps at the technical level for building, testing, deploying and operating applications. Because containers offer application encapsulation, they can serve as the lowest common denominator for applications, especially applications employing myriad different technologies. Thus, containers can standardize DevOps pipelines for testing, building, and deployments with a single set of tools while platform specifics are left to be handled by the containers.

There is no one way to do container-based SDLC. One approach is to build several container images used downstream of the build pipeline. These would be container images for various kinds of testing as well as an image intended for deployments. The build pipeline would do a hand off to testing, which would then run the test containers. If the tests pass, then the testers give it to a deployment pipeline that deploys the container.

Another approach is to integrate testing into the container build process itself. Docker multistage builds allow for multiple steps within a dockerfile and the output of one stage can serve as input for the next stage. One stage can be used for building software, the second stage can be used for testing, and then the third stage can be used for building the final version intended for deployment.

Likewise, containers can be used for builds and test, but the artifacts intended for deployments may not be containers at all. It may be the other way around, too, with builds and tests being done outside of containers, but the container becomes the build artifact intended for deployment. Containers can play a part at every step in the DevOps process.

3.5.1 Azure Solutions

Azure offers a suite of tools for container-based SDLC.

- Azure DevOps (6.4) offers a suite of tools for building, testing, and deploying containerized applications.
- Azure Container Registry (6.1) is used for hosting build artifacts whenever they are staged for testing and deployments.
- Azure Web Apps for Containers (4.2), Azure Kubernetes Services (4.3), Azure Container Instances (4.1), and Azure Service Fabric (4.5) are used to run containerized applications.
- Azure Monitor (6.3) offers purpose-built monitoring tools designed specifically for containers.

3.6 Data Science, Big Data, and AI

Data Science, Big Data, and AI are all distinct, related disciplines for analyzing data. The methods and techniques vary within each of the disciplines, but the patterns re similar. Principally, these kinds of workloads rely on long-running processes that churn through large volumes of data to produce a desired result. Multiple types of processes are often chained together to create a full pipeline that may include extracting, transforming, loading, cleaning, and normalizing data, along with data analysis, data mining, among many other tasks.

The myriad of tools used to work with data form an ecosystem oriented around a cornerstone tool such as Hadoop. Its ecosystem that includes Hbase, Hive, Spark, Sqoop, Pig, ZooKeeper, Flume, and other tools. Maintaining a consistent toolchain, therefore, becomes one of the major challenges faced by those who work with data.

Another challenge in the data field is the constraints imposed by the available compute resources. Large-dataset analysis typically requires High-performance Computing (HPC) and frequently benefits from specialized Graphics Processing Units (GPUs). HPC distributes work across multiple nodes in a compute cluster so that the work can be performed in parallel to shorten the time needed for data workloads. GPUs allow work to be parallelized within a node.

Overlap between the challenges imposed by tools and compute compounds these problems. Maintaining HPC systems with the tools needed to perform data-oriented workloads can be expensive if the systems are required to stay on even when they are not being used. Moreover, inconsistencies between development workstations and HPC clusters can create impedance between the two if the tooling gets out of sync.

Containers help ease some of the pain points created by maintaining and using consistent tooling in data analysis:

- Many data-analytics tools have already been containerized, so creating environments to use these tools is a matter of selecting the correct container image.
- The same container images can be used both in development and on HPC clusters for data workloads.
- Provisioning HPC clusters are simplified because the container brings the tooling within the container when they are ready to use.

3.6.1 Azure Solutions

Azure offers many solutions that leverage containers to help execute data-oriented workloads.

- Azure Container Instances (4.1) are billed by the second and can quickly be employed for batch processing, then quickly shut down once a process completes.
- Azure Batch with Batch Shipyard (4.8): Azure Batch can be used as an HPC solution for running data workloads in virtual machines. Batch Shipyard extends Azure Batch, adding container support.
- Azure Machine Learning with Containers (4.10) can containerize machine-learning models that can then be deployed to other container runtimes such as AKS, ACI, and IoT Edge.
- Azure Databricks with Containers (4.9) manages a Databricks cluster on Azure that can run containers.

3.7 Containers and Databases

Databases pose a unique challenge concerning containers. Best practice has historically been not to use containers for databases for two main reasons.

The first is data persistence. Containers require that any data that needs to be persisted be stored outside of the container. Containers, therefore, mount external storage into the container as a *volume*. Numerous technologies can serve up volumes from storage devices and servers in the container space. Adding volumes to containers, though, does introduce abstractions and latency to databases, especially if the volumes are coming from network-based storage.

The second challenge containers face is server state. Database clusters can be run as a cluster of containers. Nodes in a database cluster need to be able to discover and communicate with one another to share state information. Containers alone cannot accomplish this, and therefore rely on container orchestration to make it work.

When considering containers as an option for databases in the cloud, the best practice is to prefer *Database as a Service (DBaaS)*, which is a database platform offered as PaaS. DBaaS technologies are fully managed database environments with encryption, dynamic scale, automatic backups, and built-in

high availability. To accomplish these with container orchestrators is possible but requires a great deal of complex configuration of the database software and container orchestrator. DBaaS simplifies this by providing all these tasks as a service.

On premise, though, DBaaS is not an option, so the best option is non-containerized Database Management Systems (DBMS). Database scale hinges on multiple factors including traffic volume, compute needs, database complexity, database size, and established SLAs. A multi-node DBMS can accommodate databases that have one or more of these factors in quantities that exceed what can be provided by a single compute node. A DBMS would demand all the resources of a single node, so it would only allow a single container on a node in a container cluster. This would nullify application density gains offered by containers.

Containers do become a possibility for smaller databases on a smaller scale – specifically, those with lower volume, complexity, and size. Redundancy, however, can be accomplished with container clusters for demanding SLAs.

3.7.1 Azure Solutions

The best practice for cloud databases is to use DBaaS. Azure offers DBaaS for most popular database platforms, including Azure SQL for SQL Server databases, Cosmos DB for NoSQL databases which includes MongoDB, Azure Database for MariaDB, Azure Database for MySQL, Azure Database for PostgreSQL, and Azure Cache for Redis.

4 Azure Solutions Using Containers

Microsoft has built containers into many different services on Azure. Most are geared towards hosting applications, but a few are aimed at specific problems.

4.1 Azure Container Instances (ACI)

Azure Container Instances (ACI) are perhaps the easiest way to run containers on Azure. ACI does not require any sort of middleware, orchestration, or clusters. This means that running containers is as simple as creating any other service using the Azure Portal, the Azure CLI, PowerShell, the Azure Cloud Shell, or ARM templates.

ACI supports multi-container applications with a concept called a *container group*, which is analogous to a Kubernetes pod. A container group can contain multiple containers in the group, but only single instances of each container are created. The containers share a common internal network that resolves to *localhost* to allow them to communicate with one another.

ACI is intended for smaller container deployments and has commensurate limitations. Containers running on ACI do not scale dynamically or even manually. Additionally, ACI does not have integrated load balancing that can redirect requests to two similar containers. Given these limitations, ACI is not an ideal solution for running highly available (HA) applications. It can be used for demos, tests, and transient type workloads like batch processing.

ACI also integrates well with other Azure services, including Azure File Storage, Azure Key Vault, and Azure VNets for isolation.

Purpose:	General-purpose for small scale container applications
Service Type:	PaaS
Orchestration:	ARM
Supported Platforms:	Windows, Linux
Scaling:	No
Scheduling	No
Monitoring:	Yes, through Azure Monitor
Load Balancing:	No
VNet integration:	Yes, through Azure Network Profiles which assigns a private IP address
Azure File Integration:	Yes. (File permissions are set to allow only root to read and write to Azure Files.)
Key Vault Integration:	Yes, through mounted volumes in containers

For more information on ACI, view the Microsoft Docs (https://docs.microsoft.com/en-us/azure/container-instances/).

4.2 Web Apps for Containers (Azure App Services)

Azure App Services is the flagship PaaS offering that backs all sorts of web-based applications. Web Apps for Containers allows Docker containers to run within the context of Azure App Services. Azure App Services supports popular platforms for applications such as .NET, .NET Core, Java, PHP, NodeJS, and Python on Linux and Windows. Adding containers to App Services enables it to run virtually any platform that can be packaged inside a container.

App Services offers several tools for scaling, monitoring, and maintaining applications without the need to manage infrastructure. Containers running on Azure App Services can take advantage of all these features in the same way the supported platforms do. Azure App Services supports orchestration through tools such as Azure Resource Monitor (ARM) as well as preview support for Docker Compose YAML files, which are used to define services for Docker Swarm.

Web Apps for Containers on Azure, while not as simple as ACI, is not as complex as Kubernetes or Service Fabric. Web Apps for Containers on Azure lacks some of the more advanced features for running large-scale application deployments offered by Service Fabric and AKS, but this is not its intended purpose. It is for small to medium-sized web-based applications running in containers.

Purpose:	Web Based Applications
Service Type:	PaaS
Orchestration:	ARM, Docker Compose
Supported Platforms:	Windows, Linux
Scaling:	Yes, through App Services defined scaling
Scheduling:	Yes, but only for scaling, not deployments
Monitoring:	Yes, through logging and Azure Monitor
Load Balancing:	Yes, built-in
VNet integration:	Yes, through App Service Endpoints
Azure File Integration:	Yes, through mounted volumes
Key Vault Integration:	No. Applications in containers, however, can use KeyVault.

For more information on Web Apps for Containers, view the Microsoft Docs (https://docs.microsoft.com/en-us/azure/app-service/containers/)

4.3 Azure Kubernetes Services (AKS)

Azure Kubernetes Services (AKS) is the flagship offering for all sorts of containerized applications. Azure Kubernetes Services provides a full Kubernetes deployment as managed infrastructure. Kubernetes is a wildly popular container orchestration platform with a vibrant and active development community. It is deployed onto a set of virtual machines that can integrate with an Azure VNet and other services on Azure. This infrastructure is exposed to the user; however, Azure manages that infrastructure in the background. The user only pays for the worker nodes on AKS. The master nodes are provided for free as part of the service.

Load balancing is provided by Azure Load Balancer for TCP and UDP traffic. HTTP load balancing is managed by Azure Application Gateway. These integrations are optional if the user opts to not expose services through a Load Balancer or an Application Gateway on an Azure VNet, nor does the user have to allocate and configure these Azure services.

Data volumes are provided through any number of channels:

- Azure File Storage provides volumes from Storage Accounts that can easily be mounted in containers.
- Azure Disks provide high-performance storage for more demanding data workloads.
- Blobfuse allows Azure Blob storage to be mounted in AKS for more modest data workloads.
- Numerous third-party extensions for Kubernetes can provide volumes to containers.

AKS is dynamically scalable from within the Azure Portal, Azure CLI, or PowerShell. Nodes can be added or removed without interrupting the container cluster. AKS also provides the ability to easily upgrade versions of Kubernetes once they are released. Upgrading Kubernetes on Azure requires that each node on the cluster be patched and will cause the cluster to be unavailable while upgrading.

Of the PaaS offerings on Azure (ACI, AKS, Web Apps for Containers, and Service Fabric), AKS offers the most flexibility and extensibility when it comes to running containers. Moreover, Kubernetes is an open-source tool with a vibrant community. There are numerous value-added services for Kubernetes that can work with AKS and community-based support is broadly available for most questions related to Kubernetes and AKS.

Purpose:	General Purpose
Service Type:	PaaS (Managed Infrastructure)
Orchestration:	Kubernetes
Supported Platforms:	Windows, Linux
Scaling:	Yes, for both the cluster node and containers
Scheduling:	Yes
Monitoring:	Yes, natively or through Azure Monitor
Load Balancing:	Yes, built-in or through Azure Load Balancer or Application Gateway
VNet integration:	Yes, through load balancers or cluster node IP addresses
Azure File Integration:	Yes, through mounted volumes (AKS also supports disks, Blob storage, and third-party storage providers as mounted volumes)
Key Vault Integration:	Yes, as mounted volumes

For more information on Azure Kubernetes Services, view the Microsoft Docs (https://azure.microsoft.com/en-us/services/kubernetes-service/).

4.3.1 AKS Engine

AKS Engine is an open-source tool for creating Kubernetes clusters on Azure. It's useful for Azure regions in which AKS in not available. It accepts a configuration file, then interacts with Azure Resource Manager to create a cluster of virtual machines and the network infrastructure to go with it. Foundationally, AKS Engine is IaaS. The major difference between AKS Engine clusters and AKS clusters is that AKS Engine clusters are not managed by Microsoft, so patching and maintaining the virtual machines must be done by the user. AKS Engine deployments also require that the user pay for the management nodes created by the cluster. Otherwise, Kubernetes deployed by AKS Engine functions in much the same way as AKS does.

For more information on AKS Engine, see the project on GitHub (https://github.com/Azure/aks-engine).

4.4 Azure Red Hat OpenShift

Azure Red Hat OpenShift is a fully managed deployment of OpenShift on Azure developed through a partnership between Red Hat and Microsoft. OpenShift is built on Kubernetes but adds support for build pipelines and build artifacts hosted on an integrated container registry in addition to the native capabilities provided by Kubernetes. OpenShift also adds numerous Kubernetes related tools into the environment, such as Prometheus and Istio.

OpenShift on Azure is managed infrastructure, so the user does not have to patch and maintain the virtual machines deployed to an OpenShift cluster. OpenShift can be joined to existing Azure deployments through Azure VNet Peering, which is slightly different from the more integrated approach offered by Azure Kubernetes Services. RBAC can be provided through Azure Active Directory.

OpenShift, like Azure Kubernetes Services, is a fully managed platform. OpenShift's integrated build pipelines and integrated applications are its value proposition beyond a standard Kubernetes deployment, with the principal difference being that support for OpenShift is provided by two vendors, namely Microsoft and Red Hat. If a user already has existing DevOps pipelines like those supported by Azure DevOps, then OpenShift may not be the best tool.

Purpose:	General Purpose
Service Type:	PaaS (Managed Infrastructure)
Orchestration:	Kubernetes
Supported Platforms:	Linux
Scaling:	Yes, for both the cluster and containers
Scheduling:	Yes
Monitoring:	Yes, natively or through Azure Monitor
Load Balancing:	Yes
VNet integration:	Yes, through VNet Peering
Azure File Integration:	Yes, through mounted volumes (OpenShift also supports disks, Blob storage and third-party storage providers as mounted volumes)
Key Vault Integration:	Yes, as mounted volumes

For more information on Azure Red Hat OpenShift, check out the Azure Docs (https://docs.microsoft.com/en-us/azure/openshift/intro-openshift).

4.5 Service Fabric

Service Fabric is a homegrown Microsoft solution for building and managing microservices at scale. Much of Azure is built on Service Fabric, as it provides the essential frameworks for managing large-scale service deployments. Service Fabric has historically been a Windows first product. It was originally designed as a clustering model for on-premise computers and still many of the deployments for Service Fabric are on premise. Microsoft extended support for Service Fabric to Linux to support Linux-based workloads as well as Linux containers. Much like other clustering technologies on Azure, Service Fabric creates and manages infrastructure for the user.

Service Fabric has two primary models for deploying microservice applications. The first is its native model that has a prescriptive set of patterns for writing apps. The second model is for containerized applications, more specifically containerized applications that are architected for microservices.

Service Fabric as a general-purpose tool for deploying containers adds a lot of complexity to container deployments that other orchestrators do not. It tends to be very "opinionated," meaning that the service model has many baked-in assumptions that may not exist in other orchestrators. Service Fabric attempts to wrap containers in the same model that it uses to deploy its own native applications while also running them as containers. While this is not a barrier for adoption, it does remove the flexibility and extensibility granted by other container orchestration engines such as Kubernetes.

Service Fabric orchestration is managed through its own CLI tools and through service definitions that define container workloads. Scaling, storage, load balancing, and many other are features supplied by Service Fabric.

Purpose:	Microservices
Service Type:	PaaS (Managed Infrastructure)
Orchestration:	Service Fabric
Supported Platforms:	Windows, Linux
Scaling:	Yes, for both the cluster and containers
Scheduling:	Yes
Monitoring:	Yes, through Azure Monitor
Load Balancing:	Yes, built-in and through Azure Load Balancer or Application Gateway integration.
VNet integration:	Yes, through load balancers or cluster IPs
Azure File Integration:	Yes, through mounted volumes
Key Vault Integration:	Yes

For more information on Service Fabric, visit the Microsoft Docs (https://docs.microsoft.com/en-us/azure/service-fabric/).

4.6 Docker CE/EE

Docker Community Edition (Docker CE) and Docker Enterprise Edition (Docker EE) are two tools used for creating container clusters on Azure. Docker CE and Docker EE support Windows and Linux containers. Docker EE is provided as a feature on Windows Server 2016 and Windows Server 2019 and is supported by Microsoft.

Clusters built on top of Docker Community Edition (CE) or Docker Enterprise Edition (EE) use Docker Swarm for orchestration. Docker Swarm follows the master/node configuration common to all container orchestration platforms. Swarm orchestration is not as feature-rich as Kubernetes, but it suffers from less overhead, too. Otherwise, Swarm is a capable orchestration engine for almost all container workloads. Swarm uses the Docker client and can accept Docker Compose files to define containerized workloads running on Docker CE or Docker EE clusters.

Deploying Docker CE or Docker EE to Azure requires that the user build the deployment on Azure infrastructure using the tools available such as Azure Resource Manager (ARM) and ARM templates. Docker CE/EE is a reasonable migration path for container workloads that are being ported from on-premise infrastructure already running Docker CE/EE in Swarm Mode. For workloads that are running on Kubernetes, using AKS reduces the overhead by removing the need for master nodes and eases administrative burdens through managed infrastructure.

Purpose:	General Purpose
Service Type:	IaaS
Orchestration:	Swarm
Supported Platforms:	Windows, Linux
Scaling:	Yes, for containers
Scheduling	Yes
Monitoring:	Yes, through native logging

Load Balancing:	Yes, built-in as part of Swarm
VNet integration:	Yes, through virtual machine attachments.
Azure File Integration:	Yes (Docker also supports several third-party storage drivers)
Key Vault Integration:	No

Configuring Docker CE or Docker EE for orchestration first requires that Docker be installed on a node. Then the node must be placed in Swarm mode and added to a Swarm cluster. For more information on setting up a Docker Swarm Cluster, see Docker's documentation (https://docs.docker.com/engine/swarm/swarm-tutorial/).

4.7 DC/OS

DC/OS is short for Data Center Operating System. The Azure Marketplace has templates for deploying DC/OS from Mesophere (now D2iQ), the group behind DC/OS. DC/OS is based on Apache Mesos, which is an open-source platform for managing compute clusters. DC/OS is a tool for managing not only containers but also the nodes on which they run. Fundamentally, all workloads on DC/OS get containerized either at runtime or prior to runtime as a container image.

DC/OS is squarely aimed at large compute clusters with nodes numbering in the tens of thousands, while orchestrators like Swarm and Kubernetes typically manage clusters from just a few nodes to loads with node counts in the hundreds to thousands. DC/OS follows the same basic pattern implied by other container orchestrators using master and worker nodes.

DC/OS on Azure is provided by Mesophere through a template for automating DC/OS clusters on Azure. Management of the clusters is provided by DC/OS, which includes patching nodes and upgrading cluster components. As a choice for a container orchestrator, DC/OS comes second to offerings such as Service Fabric or Azure Kubernetes Services when the scope and scale of these two does not meet the demands needed for a given application workload. Alternately, multiple AKS or Service Fabric instances may be a better choice than a single DC/OS deployment on Azure.

Purpose:	General purpose for large scale container deployments
Service Type:	IaaS
Orchestration:	DC/OS
Supported Platforms:	Linux
Scaling:	Yes, for containers
Scheduling:	Yes
Monitoring:	Yes, through a rich native logging solution and GUI
Load Balancing:	Yes, built-in to DC/OS
VNet integration:	Yes, through virtual machine attachments
Azure File Integration:	Yes (DC/OS also supports several third-party storage drivers)
Key Vault Integration:	No

Provisioning DC/OS on Azure is a straightforward process using the template provided by Mesosphere for creating DC/OS clusters (https://azuremarketplace.microsoft.com/en-us/marketplace/apps/mesosphere.dcos). Documentation for managing a DC/OS cluster can be found at D2iQ's site for DC/OS (https://docs.d2iq.com/mesosphere/dcos/)

4.8 Azure Batch with Batch Shipyard

Azure Batch is the Azure High-Performance Computing (HPC) solution that provides batch process orchestration on Azure for long-running jobs such as report generation, big-data analysis, AI and Machine Learning, and content rendering. These sorts of workloads require multiple machines running in parallel. Batch provides a framework for distributing workloads across Azure virtual machines through the Azure Portal or a declarative format using configuration files.

One of the principal advantages of Azure Batch is that it helps reduce cost. It only uses compute resources when needed rather than requiring you to pay for compute when it is not being used. A batch pipeline can provision and start virtual machines on Azure, install the necessary software to make them ready for work, run a job, and then shut down the virtual machines.

Batch Shipyard is an add-on for Azure Batch that allows batch jobs to run in containers on Azure virtual machines. This approach simplifies provisioning the virtual machines by using a container that is already configured with the tools to support batch processes. Batch Shipyard supports many distributed file systems such as GlusterFS and HDFS and can also integrate with Azure Blob storage and Azure File Storage.

Azure Batch and Batch Shipyard are a niche tool for specific kinds of workloads. For smaller batch processing that does not require multiple compute nodes, Azure Container Instances run containers from start to completion, then deallocate the resources. Moreover, Azure Batch and Batch Shipyard have a moderate learning curve to understand the paradigms as well as the technical details.

Purpose:	Batch Processing for large batch jobs
Service Type:	PaaS (Managed Infrastructure)
Orchestration:	Azure Batch with Batch Shipyard files and CLI
Supported Platforms:	Linux, Windows
Scaling:	Yes, across multiple compute nodes
Scheduling:	Yes, for batch jobs
Monitoring:	Yes, through Batch Insight and the Shipyard GUI
Load Balancing:	Yes, Work is distributed across multiple compute nodes
VNet integration:	Yes, through virtual machine attachments
Azure File Integration:	Yes (Shipyard supports multiple distributed file systems as well)
Key Vault Integration:	Yes

For more information, check out Azure Batch in the Azure Docs (https://docs.microsoft.com/en-us/azure/batch/batch-technical-overview) and the Batch Shipyard project on Github (https://github.com/Azure/batch-shipyard).

4.9 Azure Databricks with Containers

Azure Databricks is the product of a partnership between the Databricks company and Microsoft. Databricks as a service on Azure integrates tightly with it. It is part Software as a Service (SaaS) and part Platform as a Service (PaaS).

The SaaS is provided by Apache Spark, atop which Databricks is built. Apache Spark provides a framework for a variety of data-handling and analytical tasks such as SQL, stream analytics, machine

learning, and graph processing, using custom code written in common languages such as Java, Scala, Python, and R.

The PaaS components of Databricks comes in the form of a managed compute cluster on which Databricks performs many of these tasks. The compute cluster is provided by Azure virtual machines running a Databricks cluster.

Databricks as of version 6 added the Databricks Container Service, which allows containers to be deployed to a Databricks cluster. Databricks provides several base images that can be customized for specific tasks. Once a Databricks image is built, it can be hosted in a public container registry or in Azure Container Registry.

Databricks attempts to create an all-in-one solution for data analytics. The addition of containers does provided extensibility for the platform. However, the platform is "opinionated," meaning that it has numerous assumptions that must be accounted for. Moreover, all-in-one solutions are often too general in some applications. Containers do help mitigate this problem but may force a user to look for more purpose-built solutions. The trade-off is that Databricks makes managing compute clusters simpler because of the abstractions it provides on top of managed infrastructure.

Purpose:	Data Analytics, Machine Learning, Data Transformation
Service Type:	SaaS, PaaS (Managed Infrastructure)
Orchestration:	Databricks Container Service
Supported Platforms:	Linux
Scaling:	Yes, across multiple compute nodes
Scheduling:	Yes
Monitoring:	Yes
Load Balancing:	Yes, work is distributed across multiple compute nodes
VNet integration:	Yes
Azure File Integration:	No (Databricks does integrate with numerous other data stores and databases)
Key Vault Integration:	Yes

For more information on Azure Databricks, check out the Azure Docs on Azure Databricks (https://docs.microsoft.com/en-us/azure/azure-databricks/what-is-azure-databricks). For information on integrating containers with Databricks, check out Databricks Container Service (https://docs.databricks.com/clusters/custom-containers.html)

4.10 Azure Machine Learning with Containers

Azure Machine Learning (AML) is a cloud-based service that provides tools and orchestration for designing, building, testing, and publishing machine-learning models. It supports many types of machine learning, such as deep learning, supervised learning, and unsupervised learning. AML also supports R and Python, two common languages used in machine learning. Azure Machine Learning has two different offerings: Azure Machine Learning Studio Classic and the newer Azure Machine Learning with its designer. Classic uses several Microsoft proprietary learning algorithms, while the new suite relies on open standards.

AML services has integration with Docker to build containers for deployments once models have been built and tested. Microsoft provides several base images that can be extended or customized for specific purposes depending on the needs of the user. AML can integrate with Azure Container Registry to host images and can deploy these images to AKS, ACI, or IoT Edge, where the model can be used in a production environment for filtering actions or as a web API.

AML as a suite of tools is best understood as Software as a Service (SaaS) that produces containers. It, in and of itself, does not supply an environment to run containers. It does give users the ability to do some orchestration through its deployments, but users can also deploy the containers produced by AML using other container orchestrators such as AKS or integrate these containers into deployment pipelines using tools like Azure DevOps.

AML services are aimed squarely at integrations with other Azure services and products. This does not imply that the services cannot be used in other contexts outside of Azure.

Purpose:	Machine Learning
Service Type:	SaaS
Orchestration:	Azure Machine Learning Deployments
Supported Platforms:	Linux
Scaling:	N/A
Scheduling:	N/A
Monitoring:	N/A
Load Balancing:	N/A
VNet integration:	N/A
Azure File Integration:	N/A
Key Vault Integration:	N/A

Check out the Azure Docs for Azure Machine Learning (https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-ml) and its use with containers (https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-and-where).

4.11 Azure IoT Edge

Azure IoT Edge brings Azure to the "edge" of on-premise networks so that rules and analytics can be performed closer to the devices that are managed by Azure's IoT Suite. Azure IoT Edge performs the following tasks:

- Responds locally to incidents such as fires, floods, or other disasters in an environment when a connection to the centralized management on Azure is disconnected.
- Performs local, real-time analytics for data and telemetry coming from devices in a local context.
- Filters and aggregates telemetry and data from devices to reduce the amount of data flowing out of devices in the local context into central data repositories on Azure.
- Acts as a message gateway to either transparently send message from devices to the cloud or translate messages from one format to another and send these to the cloud.

Azure IoT Edge's extensibility is handled through containers. Azure IoT Edge runs the containers on an Edge device; the containers interact with messages and telemetry that are flowing from devices to the

cloud and from the cloud to devices. Some of these modules may also be the output of Azure Machine Learning services.

Azure IoT service can manage devices directly without the need for Azure IoT Edge. However, if one of the aforementioned tasks is required in a local context, Azure IoT Edge software is provided for free, but the user must supply the hardware to run Azure IoT Edge.

Purpose:	IoT
Service Type:	On Premise
Orchestration:	IoT Edge Runtime with Azure Components
Supported Platforms:	Linux
Scaling:	N/A
Scheduling:	N/A
Monitoring:	N/A
Load Balancing:	N/A
VNet integration:	N/A
Azure File Integration:	N/A
Key Vault Integration:	N/A

For more information, check out the Azure Docs for Azure IoT Edge (https://docs.microsoft.com/en-us/azure/iot-edge/about-iot-edge).

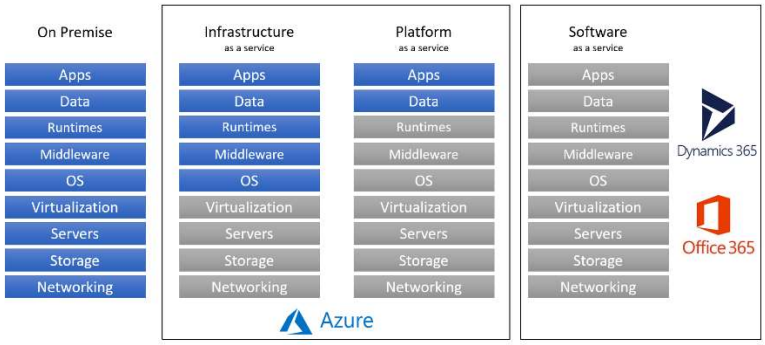
Solution	Purpose	Type	Orchrestration	Platforms	Scaling	Monitoring	Scheduling	Load Balancing	VNet integration	Azure Files	Key Vault
Azure Container Instances	General	PaaS	ARM	Windows, Linux	No	Yes	No	No	Yes	Yes	Yes
Web Apps for Containers	Web Apps	PaaS	ARM, Docker Compose	Windows, Linux	Yes	Yes	No	Yes	Yes	Yes	No*
Azure Kubernetes Service	General	PaaS (Managed IaaS)	Kubernetes	Windows, Linux	Yes	Yes	Yes	Yes	Yes	Yes	Yes
AKS Engine	General	IaaS	Kubernetes	Windows, Linux	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Service Fabric	Microservices	PaaS	Service Fabric	Windows, Linux	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Azure Red Hat Openshift	General	PaaS	Kubernetes	Linux	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Azure Batch /w Shipyard	Batch Workloads	PaaS (Managed IaaS)	ARM, Azure Batch	Windows, Linux	No	Yes	Yes	Yes	Yes	Yes	Yes
Azure Machine Learning w/ Containers	Machine Learning	SaaS	Deployment for ML	Linux	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Azure Databricks w/ Containers	Data Workloads, ML	SaaS, PaaS (Managed IaaS)	Databricks Container Service	Linux	Yes	Yes	Yes	Yes	Yes	No**	Yes
Azure IoT Edge	IoT	On Premise	IoT Edge	Linux	No	Yes	Yes	No	N/A	N/A	N/A
Docker CE/EE	General	IaaS	Swarm	Windows, Linux	Yes	Yes	Yes	Yes	Yes	Yes	No*
DC/OS	General	IaaS	DC/OS	Linux	Yes	Yes	Yes	Yes	Yes	Yes	No*
*Containers applications can be integrated with Key Vault at the container level, but there is no native way to integrate this through the Service Offering.											
**Azure Databricks can integrate with multiple other storage options beyond Azure File Storage											

5 Choosing the Right Options

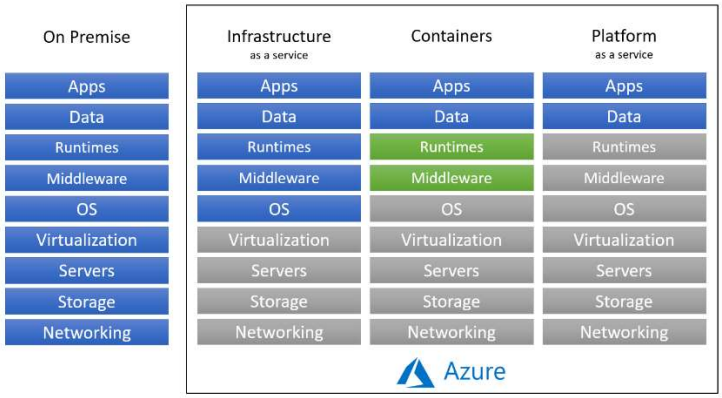
Deciding which solution to use is not always as simple as selecting the service that best fits the use case, because many of the solutions on Azure often overlap. Other factors can help you choose between two or more solutions on Azure.

5.1 Infrastructure as a Service (IaaS) vs Platform as a Service (PaaS)

General-purpose container platforms on Azure come in two basic configurations, either using Infrastructure as a Service (IaaS) or Platform as a Service (PaaS). The entire stack needed to run workloads runs from the bottom up as Networking, Storage, Servers, Virtualization, OS, Middleware, Runtimes, Data, and Apps. In the Microsoft space, Azure is principally responsible for providing IaaS and PaaS; Dynamics 365 and Office 365 are the primary Software as a Service (SaaS) offerings. (Azure does have some SaaS offering in point solutions). The principal difference between IaaS and PaaS is how much management you are responsible for. In IaaS, you are responsible for everything from the OS and up, while in PaaS, you are principally only concerned with the Apps and Data; the cloud provider manages everything beneath Apps and Data. The configuration of a PaaS or IaaS offering does require user inputs such as the app runtime to use or what ports to open on a network.



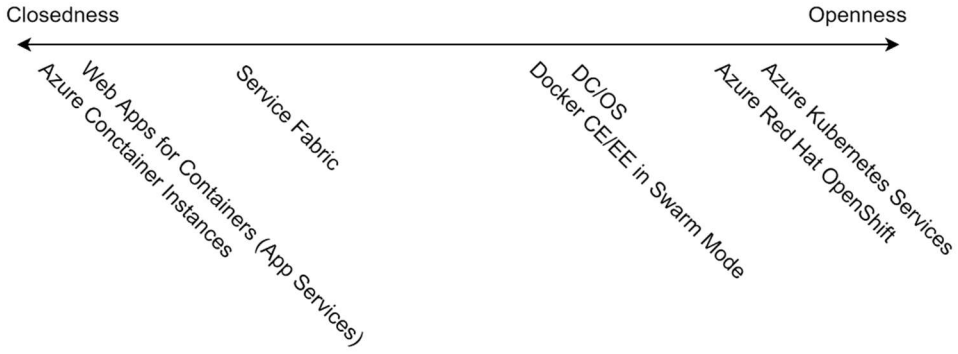
Containers do not necessarily fit cleanly into either category because containers still require that the user maintain middleware and runtimes to run applications as part of the container build process. When a container is built, the container must include these so the software will run properly in the container, but the container is not responsible for managing the Operating System.



Using containers requires maintaining container image middleware and runtimes, and therefore applies to both IaaS and PaaS. Therefore, which to choose depends whether you need to have access to and control of the managed infrastructure. In most cases for container platforms, this is not necessary -- only the ability to scale the infrastructure is needed. On Azure, most of the general-purpose offerings run on a PaaS model using managed infrastructure, with the notable exceptions of DC/OS and Docker CE/EE. DC/OS does provide the tools to manage infrastructure. Docker CE/EE are not intended to manage the infrastructure, so doing so requires additional tools.

5.2 Openness vs. Closedness

Openness versus closedness has implications for a platform's ecosystem. Open platforms tend to have more community support and extensibility, while more closed platforms tend to have smaller communities and less extensibility. Even if a platform is open-source, that does not imply that the platform is inherently as open as others. Open-source and closed-source projects can come from a single vendor who is the main contributor to the project. That vendor dictates the standards of that project. Closed systems are driven by a single entity or vendor, so they have more predictable iterations. Open platforms tend to evolve at a rapid pace, and the iterations tend to be chaotic because many people are developing solutions for many interrelated components and releasing on different schedules. The tension between open and closed platforms is a tradeoff between massive support and chaos against vendor lock-in and predictability.



5.3 Simplicity vs. Scale

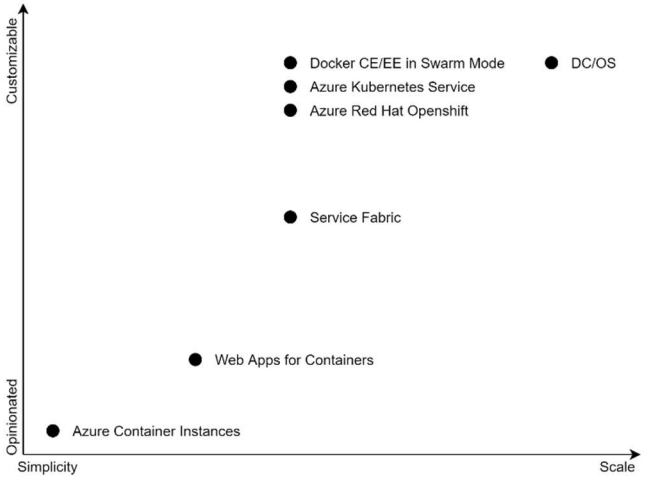
A second consideration when choosing a container platform is simplicity and scale. Simple solutions that do not need scaling have relatively simple infrastructure requirements. Such solutions will only provide basic scaling and infrastructure configurations if any. For general-purpose container platforms, Azure supports the gamut of options, ranging from simple solutions for running a few containers to solutions that scale to tens of thousands of compute nodes.

5.4 Opinionated vs. Customization

A third factor is to consider the degrees of opinionatedness against the degrees of customization. "Opinionated" implies that a platform has more baked in assumptions, therefore typically requires less or offers less configuration to the user. More customization conversely has fewer assumptions and requires more configuration on the part of the user. Opinionated systems tend to require less management and are easier to use, but they may not work for all applications. Customizable systems typically have a steeper learning curve because they require knowledge of how the system works.

5.5 Putting it Together: Ease of Use

Combined with Simplicity vs. Scale, Opinionated vs. Customization can be plotted against one another. Solutions towards the lower left would be more opinionated and simpler, thus easier to use, while ones towards the top right would be less opinionated and more scalable. but more difficult to use.



6 Supporting Solutions

6.1 Azure Container Registry

Azure Container Registry (ACR) is Microsoft's offering for hosting a public or private container registry. ACR also offers value-added services such as replication to protect against regional outages and security features such as container trust, authentication with Azure AD, VNet integration, and security scanning. These features improve the security posture of any container that is pushed to the container registry.

ACR is not necessary for any container services on Azure to work, but one advantage to using it is the tighter integration that ACR has with other Azure services through managed identities and service principals. Using service principals and managed identities to access containers improves security because brute-force attacks on a container registry are significantly mitigated and credentials for the registry do not have to be stored in insecure ways.

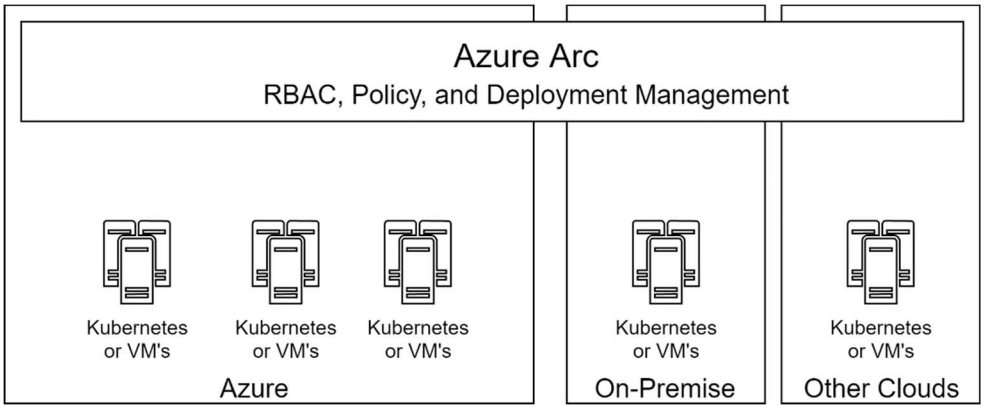
A container registry is not always necessary because existing images for more popular software can be configured through environmental variables and volumes without the need to rebuild the image. However, any sort of application that intends to integrate additional software, particularly custom software, into the container will need a container registry.

For more information, check out the Azure Docs for Azure Container Registry (https://docs.microsoft.com/en-us/azure/container-registry/container-registry-intro).

6.2 Azure Arc

Azure Arc extends Azure management to Linux and Windows virtual machines and Kubernetes clusters hosted on-premises, in other clouds, and in edge environments. With Azure Arc, users can build containerized apps that are deployed, configured, and managed using GitOps-based configuration management. Azure Arc also provides Azure-based RBAC and policy-based security such as network securing ports on cluster.

The value added by Azure Arc is centralized governance of infrastructure. Azure Arc is intended to extend Azure management to clusters, but not directly managed Kubernetes clusters. Tangentially, the DevOps tools used by Azure Arc do affect Kubernetes clusters. The principal use for Azure Arc is for distributed systems that are managed by a single organization, which may or may not include Kubernetes.



For more information, check out the Azure product page for Azure Arc (https://azure.microsoft.com/en-us/services/azure-arc/).

6.3 Azure Monitor with Containers

Azure Monitor has two differing monitoring capabilities for containers:

- **Azure Monitor for Containers** monitors container environments running on Azure including Azure Kubernetes Services (AKS), Azure Container Instances (ACI), and Azure Red Hat OpenShift. Azure Monitor can also monitor self-hosted Kubernetes clusters hosted on-premise or in other clouds. Azure Monitor for Containers can pull telemetry and log information from both environments and containers running on those environments to provide reporting and diagnostic services. Alerts can be configured to provide actionable workflows such as notification or scaling through automation. Aside from on-premise Kubernetes clusters, using Azure Monitor for Containers for this context does not require any complex configuration.
- **Container Monitoring in Azure Monitor** is a more general-purpose tool for containers deployed to several different orchestrators including Kubernetes, DC/OS, Docker Swarm, Service Fabric, OpenShift, and stand-alone instances of Docker running on Windows or Linux. Container monitoring in these environments can monitor image inventories, container inventories, and container performance. Monitoring can also retrieve logs and container events such as starts, stops, and restarts. This solution does not provide environment-specific telemetry like that found in Azure Monitor for Containers.

The overlap between these two options on Azure Monitor can create confusion. Azure Monitor for Containers is usually preferable because it encapsulates much of the functionality offered by the more general-purpose solution. When this is not possible, use the Container Monitoring in Azure Monitor.

Azure Monitor can also work alongside many other container monitoring solutions if they are deployed to clusters and containers alike.

For more information for Azure Monitor for Containers, see the Azure Docs (https://docs.microsoft.com/en-us/azure/azure-monitor/insights/container-insights-overview), and for more information about Container Monitoring in Azure Monitor, see the Azure Docs as well (https://docs.microsoft.com/en-us/azure/azure-monitor/insights/containers).

6.4 Azure DevOps

Azure DevOps is a set of integrated tools that provide automation for building and deploying software as well as collaborating on software projects.

Containers can be a build environment, a build artifact, or a deployed application to any number of container environments. Azure DevOps offers several integrated solutions for building container images or using containers as environments to perform builds.

- Azure build pipeline jobs can be hosted on containers instead of building software on a build host itself. Container-based builds can use existing images that have build tools already installed so the build host does not have to be configured before software can be built. (https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases)

- Container images built on Azure DevOps can pull code from an Azure DevOps repository or third-party repository, execute the build, and then containerize the result, or execute the build of the software in the container as specified in a Dockerfile. Once containers are built, they can be pushed from the Azure host into a container registry (including ACR) and used for deployments and tests. (https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/cicd-for-containers)
- Kubernetes integration on Azure DevOps deploys Kubernetes manifest files to Azure Kubernetes Services (AKS) or any other Kubernetes cluster. The pipelines can consume YAML files hosted in code repositories and can be customized for deployments across multiple environments, including those hosted in other clouds. (https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/kubernetes/deploy)
- Azure DevOps can deploy Azure Machine Learning models that have been containerized. (https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/azure-machine-learning)

Azure DevOps is not prescriptive, so you can mix and match different container-based solutions for builds and deployments.

6.5 Key Vault

Key Vault is Azure's secrets-management solution storing sensitive information such as passwords, certificates, and connection strings.

Key Vault on Azure's secrets management that can be integrated into container solutions so that sensitive data can be presented to an application in a container securely. Kubernetes offers Flex Volumes that can be mounted as part of the file system in a container so the container can read the secret data as a file (https://github.com/Azure/kubernetes-keyvault-flexvol). Key Vault can similarly be integrated with apps running on Azure Containers Instances (https://docs.microsoft.com/en-us/azure/container-instances/container-instances-managed-identity).

For more information about Key Vault, read the Azure Docs to get an overview (https://docs.microsoft.com/en-us/azure/key-vault/).

6.6 Azure Storage Accounts

Azure Storage offers three forms of storage that can be used with many kinds of container environments including Azure Container Instances, AKS, and Web Apps for Containers.

- Azure Blob Storage (https://azure.microsoft.com/en-us/services/storage/blobs/) is a means to store unstructured data that can be accessed through an API. The API itself has been abstracted in such a way that allows Linux hosts and containers to mount Blob storage as part of the file system through BlobFuse (https://github.com/Azure/azure-storage-fuse), which works with Kubernetes and Web Apps for Containers. BlobFuse, however, is not intended for read- and write-heavy loads.
- Azure File Storage (https://docs.microsoft.com/en-us/azure/storage/files/storage-files-introduction) provides an SMB share that can be mounted as part of a container's filesystem for persisting data outside the container. SMB is better suited for random read and writes to the file system; most container environments support using Azure File storage.
- Azure Data Lake (https://docs.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction) provides an HDFS file system that can be used by many of the big-data workloads.

Azure Data Lake is based on blob storage but extends it to support petabyte-scale data storage for unstructured data on Azure. Data Lake is supported by many Azure big-data solutions such as Azure Batch and Azure Databricks.

6.7 Azure Networking

Azure networking provides the backbone for communication between nodes running containers as well as connections to other Azure resources and application users. Almost all solutions on Azure provide some level of integration with Azure Virtual Networks (VNets). Azure VNet integration allows solutions to access resources that are hosted on the VNet through a private channel that is not traversing the internet, which improves security. There are four ways container solutions integrate with Azure VNets.

- Virtual machines connected to a VNet through a NIC is the principal way most of the larger scale orchestrators such as DC/OS, AKS, and Service Fabric integrate with Azure virtual networks. The nodes in the container cluster each have their own private IP address on the VNet, but the orchestrators provide abstractions that can expose the nodes through load balancers and public IP addresses if the applications need to be publicly accessible.
- Azure Network Profiles is a newer approach that is leveraged by Azure Container Instances. It provides a private and/or public IP address to the Azure Container Instance on an existing Azure VNet, allowing to operate like a virtual machine on that VNet.
- Azure Service Endpoints provide a way for resources on an Azure VNet to access whatever is exposed through the Azure Service Endpoint through name resolution without traversing the public Internet. However, Azure Service Endpoints are not exposed as IP addresses on the virtual network. This method is used by Web Apps for Containers on Azure App Services.
- VNet Peering is used by Azure Red Hat OpenShift because the Azure VNet used by OpenShift is entirely managed and abstracted away from the end user. VNet peering creates a private connection between two VNets that allows traffic to seamlessly be routed between them, even across Azure regions.

7 Conclusion

There is no single solution within the container space that does everything well. That's why the container space has evolved very rapidly to create solutions for all kinds of workloads, some more general than others. Azure offers dozens of solutions for building and running containers to address whatever needs the user may have.

The one container option that stands out from the rest when it comes to versatility and scalability is Azure Kubernetes Services (AKS). AKS as a general-purpose platform can be scaled from a single node to thousands of nodes depending on the needs of the applications. Its extensibility and community are derived from the Kubernetes ecosystem and are among the largest of any container orchestration platform available today. Kubernetes has, in effect, won the orchestrator wars and continues to be the go-to solution for container-based applications.

Apart from AKS, ACI and Web Apps for Containers offer viable options for smaller-scale deployments that do not need multiple applications running in the same environment. Point solutions for Big Data and IoT will continue to evolve as well as containers continue to grow in maturity in their respective domains.