CONTAINER SECURITY IN MICROSOFT AZURE

August 2018

Executive summary

Container technology is causing a structural change in the cloud-computing world. Containers make it possible to run multiple instances of an application on a single instance of an operating system, thereby using resources more efficiently. Containers give organizations consistency and flexibility. They enable continuous deployment because the application can be developed on a desktop, tested in a virtual machine, and then deployed for production in the cloud. Containers provide agility, streamlined operations, scalability, and reduced costs due to resource optimization.

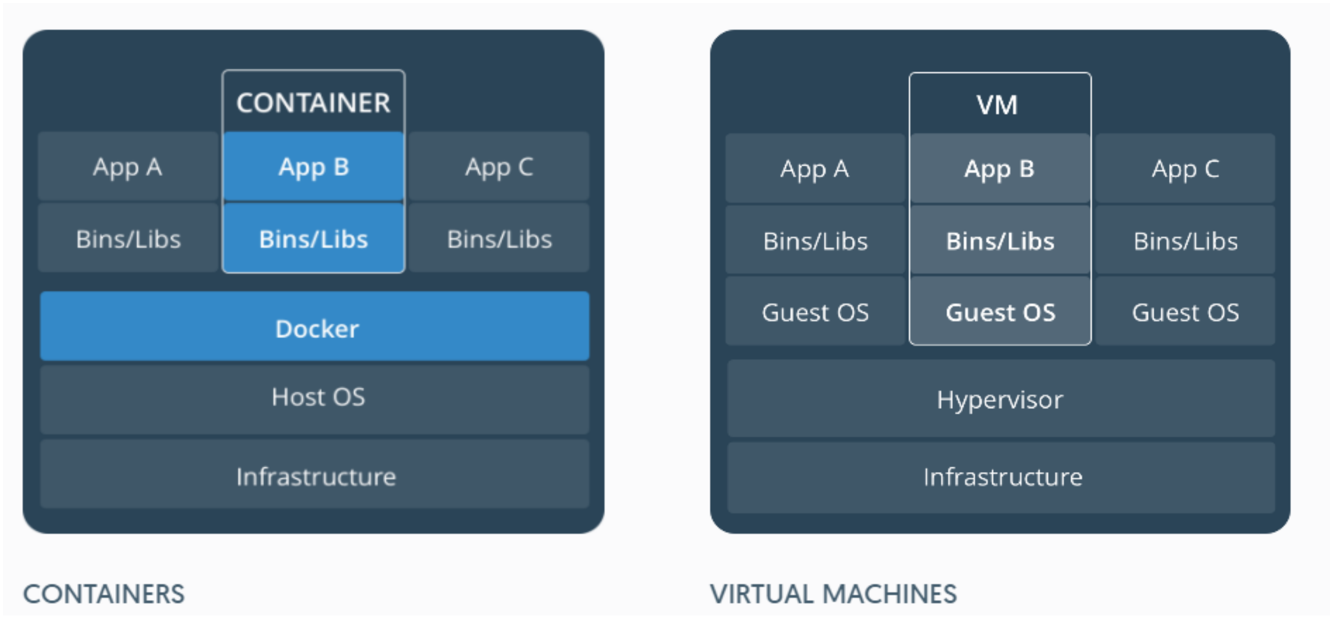
Because container technology is relatively new, many IT professionals have security concerns about the lack of visibility and usage in a production environment. Development teams are often unaware of security best practices. This white paper can help security operations teams and developers in selecting approaches to secure container development and deployments on the Microsoft Azure platform.

This paper describes containers, container deployment and management, and native platform services. It also describes runtime security issues that arise with the use of containers on the Azure platform. In figures and examples, this paper focuses on Docker as the container model and Kubernetes as the container orchestrator. Most of the security recommendations also apply to other container models from Microsoft partners on the Azure platform.

Contents

Container overview and use cases

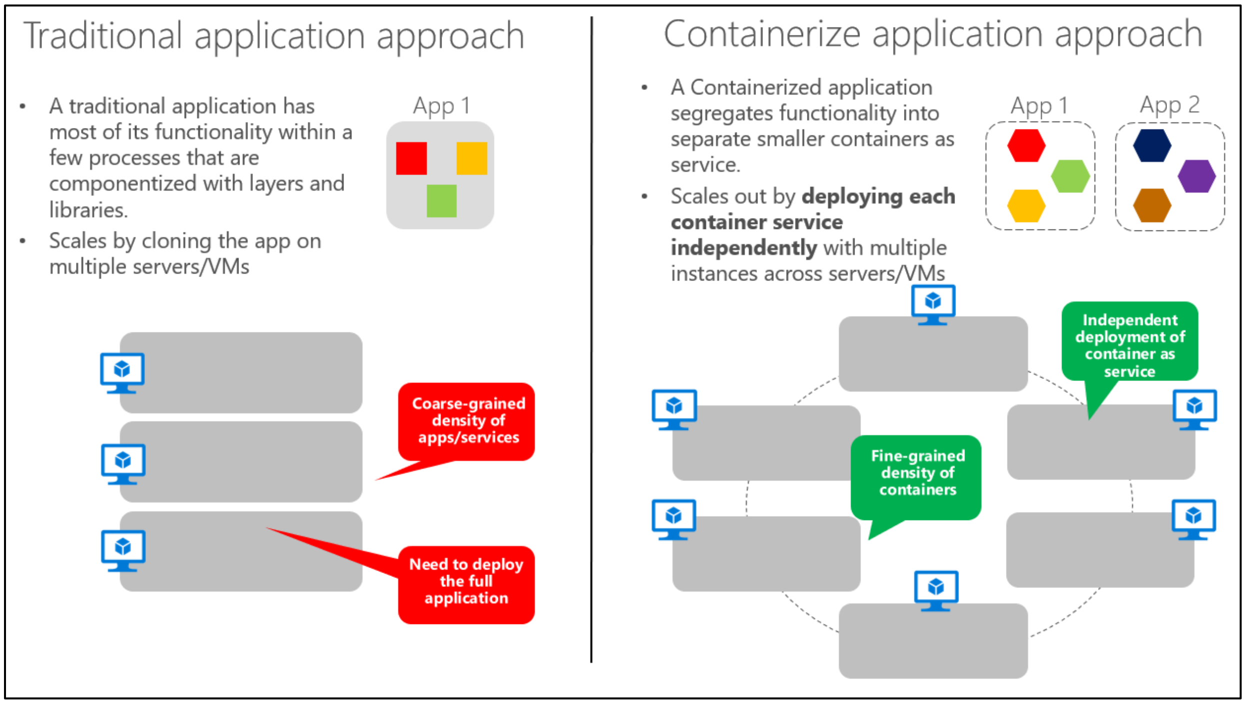
A container image is a lightweight, standalone, executable package that includes everything needed to run an application. It includes the app, system tools, system libraries, settings for the application, and more. Containers differ from hypervisor-based virtual machines (VMs) in that they run on a single copy of an operating system that's running on the hardware. The following figure compares containers and VMs:



When an app is containerized, the app and the components needed to run the app are combined in a single image. Containers are then created from this image as needed. Companies can use images as a baseline and build on them to create other images, making subsequent image creation faster. Also, multiple containers can share the same image, which allows containers to start very quickly and use fewer resources.

Although containers have existed for decades, companies like Docker have popularized the approach of container images and container management, making them viable in today's datacenter. Historically, container technology was exclusively implemented on Linux-based operating systems. But in the past few years, other modern operating systems, like Microsoft Windows Server, have embraced container technology. In turn, the container development community has embraced Windows Server.

The following chart compares traditional and containerized applications:


Traditional application approach
A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
Scales by cloning the app on multiple servers/VMs
App 1
Coarse-grained density of apps/services
Need to deploy the full application
Containerize application approach
A Containerized application segregates functionality into separate smaller containers as service.
Scales out by deploying each container service independently with multiple instances across servers/VMs
App 1
App 2
Independent deployment of container as service
Fine-grained density of containers

Containers as microservices are getting a lot of attention these days. In the modern world, applications and their supported features are growing large and complex. Instead of building a monolithic application, the increasingly popular idea among developers is to split applications into sets of smaller, interconnected services.

Azure Service Fabric is a container and process orchestrator that's part of the Microsoft Azure platform. It can deploy applications that are running either in containers or as processes.

Containers have become popular in application development, especially in cloud computing. This is largely due to containers becoming more portable, scalable, and simple to deploy. Here are some common use cases for containers:

- **Continuous integration (CI) and deployment**:

  Integrate containers into your existing agile and DevOps workflows by using Visual Studio Team Services. Push Docker images to an authenticated registry and use Docker command-line operations. Secure DevOps incorporates the security team and their capabilities into your DevOps practices, making security a responsibility of everyone on the team. Adding continuous security validation each step from development through production helps ensure that the application is always secure.

- **Containers as a service**:

  A Containers as a Service (CaaS) security strategy includes knowing when you're at risk through comprehensive monitoring and alerts. CaaS solutions enable you to query the complete state of your container server cluster. They also enable you to access familiar security features, such as security groups, user and access roles, and inter-container linkage rules. You can install additional security monitoring tools from Azure to enhance monitoring, log management, and intrusion detection. This enables the use of security and monitoring dashboards, which provide vulnerability and compliance information at a glance.

- **Application modernization**:

  Docker Enterprise Edition (EE) enables IT operations to modernize traditional applications, and then deploy them onto partners' hybrid cloud infrastructures, without requiring modifications to the source code. Containerizing the application without adjusting the source code helps legacy applications gain hybrid cloud portability, increased security, and cost efficiency.

- **IT infrastructure optimization**:

  Containers offer portability. A Docker workload deployed to the cloud can be easily migrated to the datacenter, either on "bare metal" or into a virtual machine. Containers offer greater efficiency by using fewer resources than both virtual machine–based and bare-metal deployments of the same workloads. For virtual machines, greater efficiency comes from containers sharing the underlying operating system, versus requiring a dedicated operating system for each application instance. For the bare-metal case, Docker container isolation enables organizations to run multiple workloads on the same server with little overhead.

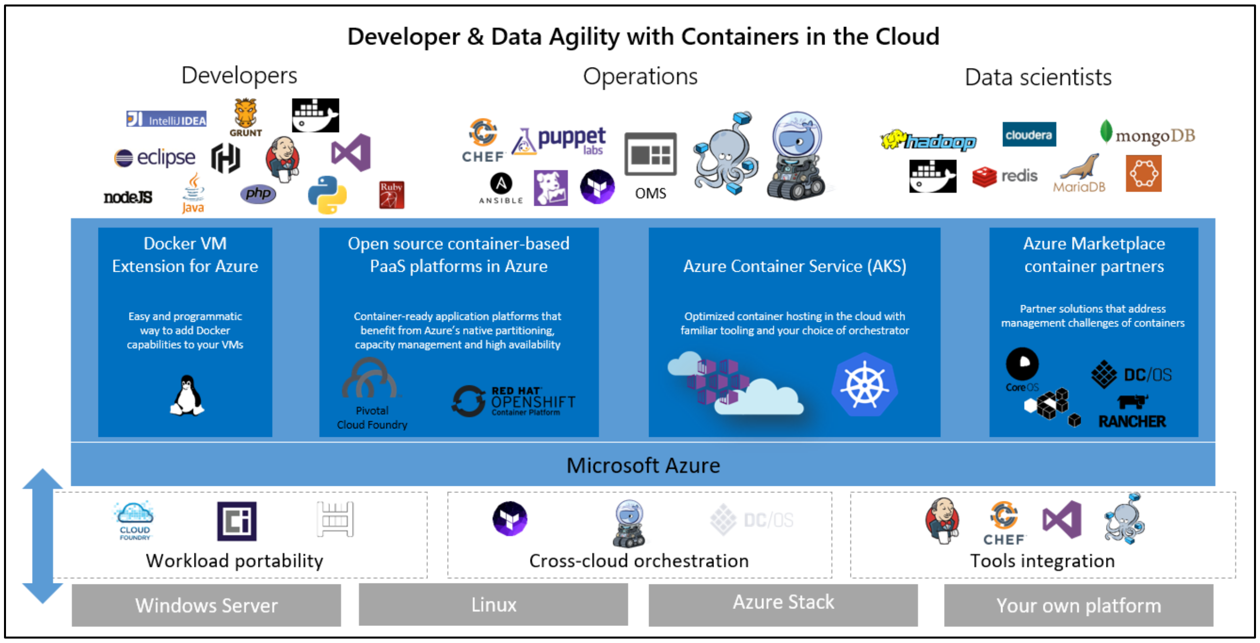
- **Hybrid cloud**:

  Containers provide a couple of key advantages for hybrid clouds. First, they provide a layer of abstraction between the containerized application and the native cloud platform on which that application runs. Second, this abstraction from the underlying cloud platform allows applications to move more easily from one cloud environment to another, including between private and public clouds. Protecting your investments in legacy applications, tools, and IT infrastructures while leveraging the benefits of more agile and scalable cloud services is the catalyst for hybrid cloud deployments.

Container services and technologies

Containers have changed the way developers develop their applications, the way applications are deployed, and the way system administrators manage their environments. Containers offer a universally accepted and open standard, enabling simple portability between platforms and between clouds.

The Microsoft Azure platform offers managed container services and supports additional container technologies through the Azure Marketplace. The following figure illustrates the Azure open-source container portfolio:


Developer & Data Agility with Containers in the Cloud
Developers
Operations
Data scientists
Docker VM Extension for Azure
Easy and programmatic way to add Docker capabilities to your VMs
Open source container-based PaaS platforms in Azure
Container-ready application platforms that benefit from Azure's native partitioning, capacity management and high availability
Pivotal Cloud Foundry
RED HAT OPENSHIFT Container Platform
Azure Container Service (AKS)
Optimized container hosting in the cloud with familiar tooling and your choice of orchestrator
Azure Marketplace container partners
Partner solutions that address management challenges of containers
CoreOS
DC/OS
RANCHER
Microsoft Azure
Workload portability
Cross-cloud orchestration
Tools integration
Windows Server
Linux
Azure Stack
Your own platform

Azure Kubernetes Service

Azure Kubernetes Service (AKS)—previously called Azure Container Service—makes it simple to create, configure, and manage a cluster of virtual machines that are preconfigured to run containerized applications. This enables you to use your existing skills, or draw upon a large and growing body of community expertise, to deploy and manage container-based applications on Microsoft Azure.

The goal of AKS is to provide a container-hosting environment by using open-source tools and technologies that are popular among customers today. AKS exposes the standard Kubernetes API endpoints. By using these standard endpoints, you can take advantage of any software that can talk to a Kubernetes cluster. For example, you might choose kubectl, Helm, or Draft.

AKS reduces the complexity and operational overhead of managing a Kubernetes cluster by offloading much of that responsibility to Azure. Azure handles critical tasks like health monitoring and maintenance for you.

AKS provides:

- Automated Kubernetes version upgrades and patching.
- Easy cluster scaling.
- Self-healing hosted control plane (masters).
- Cost savings—pay only for the agent pool nodes that are running in your clusters.

With Azure handling the management of the nodes in your AKS cluster, you no longer need to perform many tasks manually, like cluster upgrades. AKS does not provide direct access (such as with SSH) to the cluster.

AKS is also integrated with Azure Active Directory (Azure AD) and the following controls:

- Azure Active Directory Integration
  - kubectl OIDC authentication
  - Azure AD device authentication for 2FA
- Kubernetes role-based access control (RBAC)

Monitoring Azure Kubernetes Service with Azure Monitor container health

Container health offers performance monitoring by collecting memory and processor metrics from controllers, nodes, and containers available in Kubernetes through the Metrics API. After you enable container health, these metrics are automatically collected for you. They're collected through containerized version of the OMS Agent for Linux and stored in your Azure Log Analytics workspace.

The predefined views show the residing container workloads and what is affecting the performance health of the Kubernetes cluster. You can then:

- See what containers are running on the node and their average processor and memory utilization to identify resource bottlenecks.
- Identify where the container resides in a controller and/or pods to see the overall performance for a controller or pod.
- Review the resource utilization of workloads that are running on the host but are unrelated to the standard processes that support the pod.
- Understand the behavior of the cluster under average and heaviest load to help identify capacity needs and determine the maximum load that it can sustain.

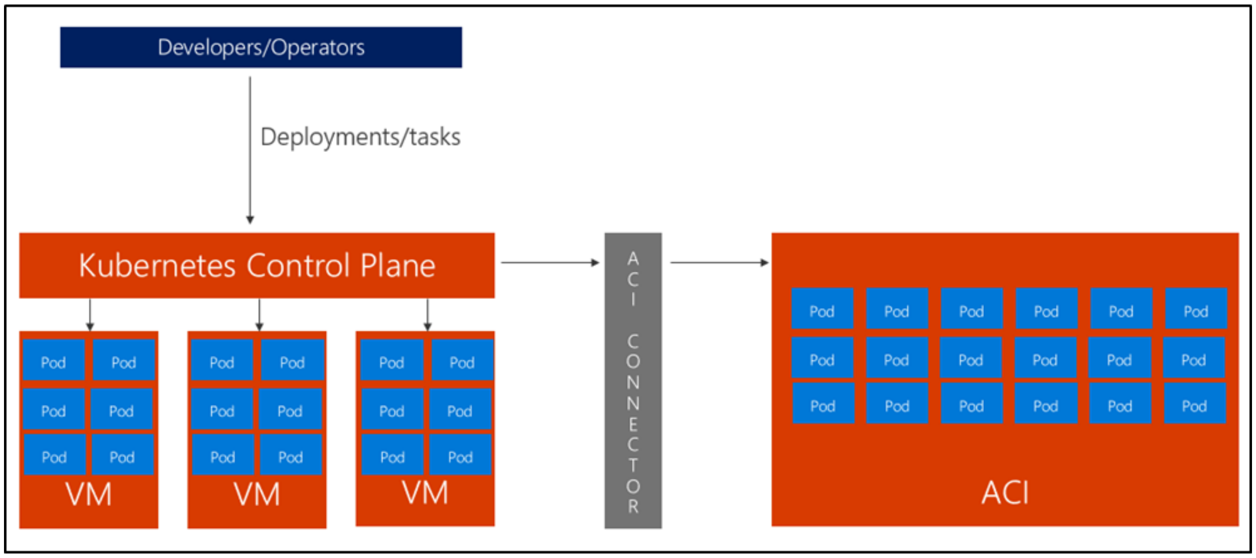
For details, see the blog post Monitoring Azure Kubernetes Service (AKS) with Azure Monitor container health (preview).

Azure Container Instances

Azure Container Instances offers the fastest and simplest way to run a container in Azure, without having to provision any virtual machines and without having to adopt a higher-level service. With ACI, containers offer RBAC on the instance and billing tags to track usage at the individual container level.

The ACI Connector for Kubernetes is an open-source connector that enables Kubernetes clusters to deploy to Azure Container Instances. It enables on-demand and nearly instantaneous container compute, orchestrated by Kubernetes, without having VM infrastructure to manage. It takes advantage of the portable Kubernetes API.

The ACI Connector allows Azure Container Instances and VMs to be used simultaneously in the same Kubernetes cluster. Azure Container Instances can be used for fast bursting and scaling, whereas VMs can be used for the more predictable scaling. Workloads can even migrate back and forth between these underlying infrastructure models. This migration offers a level of agility for deploying Kubernetes. It enables services that start in seconds without any underlying VMs and are billed and scaled per second.



Here's a sample project that shows how to perform ETL (extract, load, transform) jobs by using Azure Container Instances: Extracting, transforming, and loading data with Azure Container Instances.

Azure Container Registry

You can use Azure Container Registry to store images for all types of container deployments. These deployments include DC/OS, Docker Swarm, Kubernetes, and Azure services such as Azure App Service, Batch, and Service Fabric. Your DevOps team can manage the configuration of apps that are isolated from the configuration of the hosting environment.

To provide scale across the global footprint of Azure, Azure Container Registry supports geo-replication. Through the click of a map, you can manage a single registry that's replicated across any number of regions. Any push/pull of a container image to Azure Container Registry is routed to the closest registry.

Azure Container Registry geo-replication enables customers to manage their global deployments as one entity. The geo-replication feature caters to customers who operate at global scale.

You can create container groups by using the following samples:

- To create a container group by using images from a private registry by using C#, see Getting started with ACI - Manage with Azure Container Registry - in .NET.
- To create a container group with multiple instances and container images by using C#, see Getting started with ACI - Manage with multiple container images - in .NET.

Security concerns for containers and their solutions

Containers are not inherently vulnerable. But as with all IT technologies, it's important to adhere to strict guidelines and procedures for security. The following guidelines and procedures address operational practices more than technical practices.

For example, privilege escalation, repository validation, and image signing represent special threats in the container world. These are new constructs, and people who are unfamiliar with them might not know the proper way to govern and work with them.

Networking in a container deployment is another special area that needs to be addressed in security scenarios. Unlike VMs, containers have open network traffic across services and a shared kernel—which is a serious security concern. However, you could make the case that VMs are less secure than containers because breaking applications into microservices with well-defined interfaces and limited packaged services reduces the overall attack surface.

To use containers safely, you need to be aware of the potential security issues and the major tools and techniques for securing container-based systems.

Kernel exploits

Unlike in a VM, the kernel is shared among all containers and the host. This sharing magnifies the importance of any vulnerabilities in the kernel.

Denial-of-service attacks

All containers share kernel resources. If one container can monopolize access to certain resources—including memory and user IDs—it can starve out other containers on the host. The result is a denial of service (DoS), whereby legitimate users are unable to access part or all the system.

For example, opening sockets repeatedly will slow the entire host machine and eventually cause it to stop working.

Container breakouts

An attacker who gains access to a container should not be able to gain access to other containers or the host. By default, users are not included in the container namespace, so any process that breaks out of the container will have the same privileges on the host as it did in the container. If you were root in the container, you will be root on the host. You need to prepare for potential privilege escalation attacks—whereby a user gains elevated privileges such as those of the root user.

Poisoned images

How do you know that the images you're using are safe, haven't been tampered with, and come from where they claim to come from? If an attacker can trick you into running an image, both the host and your data are at risk. Similarly, you want to be sure that the images you're running are up to date and don't contain versions of software with known vulnerabilities.

Compromising secrets

When a container accesses a database or service, it likely requires a secret, such as an API key or a username and password. An attacker who can get access to this secret will also have access to the service. This problem becomes more acute in a microservice architecture in which containers are constantly stopping and starting, as compared to an architecture with small numbers of long-lived VMs.

Container Security Issues and Approaches

Compared to VMs, containers are much smaller and more efficient. For an application to run in a virtual machine, the application runs on a guest operating system, and the guest operating system requires a hypervisor on a server. Containers share the host operating system kernel with other containers via API calls.

Organizations need to understand the security issues that arise from the differences in how VMs and containers function. Prepare for the glut of additional files that need protection with containers and the unwieldy nature of partner libraries that containers use. Organizations that adopt containers need to accept responsibility for securing them, and should stay informed about new container vulnerabilities as the industry discovers them.

Sharing of the host operating system kernel is one of the primary benefits of containers, but it's also the crux of security concerns with containers. The lack of proper isolation between containers and the kernel during runtime means that a vulnerability in the shared OS kernel can be used to gain access to or exploit the containers.

Another element of concern in securing containers is their volatile and dynamic nature. Hundreds or thousands of containers can be created or destroyed in an instant to scale with demand. They are often short lived and have dynamic IP addresses. The volume of potentially vulnerable endpoints makes identifying and resolving security issues a challenging task.

The following security measures, implemented well and managed effectively, can help you secure and protect your container ecosystem.

Use vulnerability management as part of your container development lifecycle

By using effective vulnerability management throughout the container development lifecycle, you improve the odds that you can identify and resolve security concerns before they become a more serious problem.

Scan for vulnerabilities before pushing images to the registry

A container image registry is a service that stores container images and is hosted either by a partner or as a public/private registry such as DockerHub and Quay.

As a final check after container development is complete, you should perform a vulnerability scan on containers before pushing the images to the registry.

Continue scanning in the registry

New vulnerabilities are discovered all the time, so scanning for and identifying vulnerabilities is a continuous process. Continue to scan container images in the registry both to identify any flaws that were somehow missed during development and to address any newly discovered vulnerabilities that might exist in the code used in the container images.

Map image vulnerabilities to running containers

You need to have a means of mapping vulnerabilities identified in container images to running containers, so security issues can be mitigated or resolved.

Ensure that only approved images are used in your environment

There's enough change and volatility in a container ecosystem without allowing unknown containers as well. Allow only approved container images. Have tools and processes in place to monitor for and prevent the use of unapproved container images.

An effective way of reducing the attack surface and preventing developers from making critical security mistakes is to control the flow of container images into your development environment. For example, you might sanction a single Linux distribution as a base image, preferably one that is lean (Alpine or CoreOS rather than Ubuntu) to minimize the surface for potential attacks.

Image signing or fingerprinting can provide a chain of custody that enables you to verify the integrity of the containers.

Permit only approved registries

An extension of ensuring that your environment uses only approved images is to permit only the use of approved container registries. Requiring the use of approved container registries reduces your exposure to risk by limiting the potential for the introduction of unknown vulnerabilities or security issues.

Ensure the integrity of images throughout the lifecycle

Part of managing security throughout the container lifecycle is to ensure the integrity of the container images in the registry and as they are altered or deployed into production.

Enforce least privileges in runtime

The concept of least privileges is a basic security best practice that also applies to containers. When a vulnerability is exploited, it generally gives the attacker access and privileges equal to those of the compromised application or process. Ensuring that containers operate with the lowest privileges and access required to get the job done reduces your exposure to risk.

Reduce the container attack surface by removing unneeded privileges

You can also minimize the potential attack surface by removing any unused or unnecessary processes or privileges from the container runtime.

Privileged containers run as root. If a malicious user or workload escapes in a privileged container, the container will then run as root on that system.

Whitelist files and executables that the container is allowed to access or run

Reducing the number of variables or unknowns helps you maintain a stable, reliable environment. Limiting containers so they can access or run only preapproved or whitelisted files and executables is a proven method of limiting exposure to risk.

It's a lot easier to manage a whitelist when it's implemented from the beginning. A whitelist provides a measure of control and manageability as you learn what files and executables are required for the application to function correctly.

A whitelist not only reduces the attack surface but can also provide a baseline for anomalies and prevent the use cases of the "noisy neighbor" and container breakout scenarios.

Enforce network segmentation on running containers

To help protect containers in one segment from security risks in another segment, maintain network segmentation (or nano-segmentation) or segregation between running containers. Maintaining network segmentation may also be necessary for using containers in industries that are required to meet compliance mandates.

For example, the partner tool Aqua provides an automated approach for nano-segmentation. Aqua monitors container network activities in runtime. It identifies all inbound and outbound network connections to/from other containers, services, IP addresses, and the public internet. Nano-segmentation is automatically created based on monitored traffic.

Monitor container activity and user access

As with any IT environment, you should consistently monitor activity and user access to your container ecosystem to quickly identify any suspicious or malicious activity. The container monitoring solution in Log Analytics can help you view and manage your Docker and Windows container hosts in a single location.

By using Log Analytics, you can:

- View detailed audit information that shows commands used with containers.
- Troubleshoot containers by viewing and searching centralized logs without having to remotely view Docker or Windows hosts.

- Find containers that may be noisy and consuming excess resources on a host.
- View centralized CPU, memory, storage, and network usage and performance information for containers.

On computers running Windows, you can centralize and compare logs from Windows Server, Hyper-V, and Docker containers. The solution supports container orchestrators such as Docker Swarm, DC/OS, Kubernetes, Service Fabric, and Red Hat OpenShift.

Monitor container resource activity

Monitor your resource activity, like files, network, and other resources that your containers access. Monitoring resource activity and consumption is useful both for performance monitoring and as a security measure.

Azure Monitor enables core monitoring for Azure services by allowing the collection of metrics, activity logs, and diagnostic logs. For example, the activity log tells you when new resources are created or modified.

Metrics are available that provide performance statistics for different resources and even the operating system inside a virtual machine. You can view this data with one of the explorers in the Azure portal and create alerts based on these metrics. Azure Monitor provides the fastest metrics pipeline (5 minutes down to 1 minute), so you should use it for time-critical alerts and notifications.

Log all container administrative user access for auditing

Maintain an accurate audit trail of administrative access to your container ecosystem, container registry, and container images. These logs might be necessary for auditing purposes and will be useful as forensic evidence after any security incident. You can use the Container Monitoring management solution in Azure Log Analytics to achieve this purpose.

Securing Docker containers in Azure Kubernetes Service

Microsoft Azure provides integrated security throughout the container development lifecycle. There are three integration points for this security:

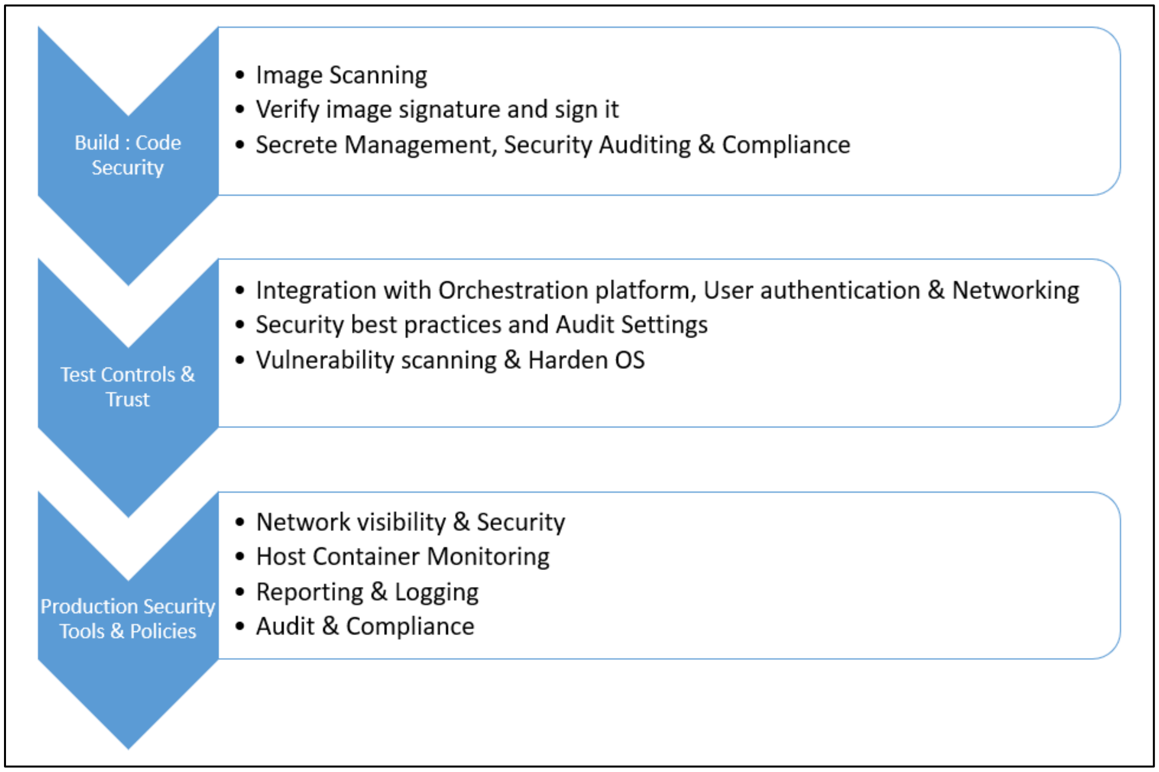

Build : Code Security
Image Scanning
Verify image signature and sign it
Secrete Management, Security Auditing & Compliance
Test Controls & Trust
Integration with Orchestration platform, User authentication & Networking
Security best practices and Audit Settings
Vulnerability scanning & Harden OS
Production Security Tools & Policies
Network visibility & Security
Host Container Monitoring
Reporting & Logging
Audit & Compliance

Image security

Containers are built from images that are stored in one or more repositories. These repositories can belong to public or private container registries like Docker Hub.

An example of a private registry is the Docker Trusted Registry, which can be installed on-premises or in a virtual private cloud. There are also cloud-based private container registry services, including Azure Container Registry. A publicly available container image does not guarantee security. Container images consist of multiple software layers, and each software layer might have vulnerabilities.

To help prevent attacks, users should store and retrieve images from a private registry, such as Azure Container Registry or Docker Trusted Registry. In addition to providing a managed private registry, Azure Container Registry supports service principal-based authentication through Azure Active Directory for basic authentication flows. This authentication includes role-based access for read-only, write, and owner permissions.

Security monitoring and scanning solutions such as Twistlock and Aqua are available through the Azure Marketplace. You can use them to scan container images in a private registry and identify potential vulnerabilities. It's important to understand the depth of scanning that the different solutions provide.

Containers can spread across several clusters and Azure regions. So you must secure credentials required for logins or API access, such as passwords or tokens. Ensure that only privileged users can access those containers in transit and at rest. Inventory all credential secrets, and then require developers to use emerging secrets-management tools that are designed for container platforms.

Make sure that your solution includes encrypted databases, TLS encryption for secrets data in transit, and least-privilege RBAC role access controls. Azure Key Vault is a cloud service that safeguards encryption keys and secrets (such as certificates, connection strings, and passwords) for containerized applications. Because this data is sensitive and business critical, secure access to your key vaults so that only authorized applications and users can access them.

Test: network controls

You must secure the runtime environment (host, kernel, Docker, orchestration tools) before running application containers live in production.

Azure virtual networks are secure, logical networks that provide network isolation. They provide security controls to configure network segmentation and create security layers between containers. Customers create their own structure by using subnets. They use their own private IP address range and configure route tables, network security groups, access control lists (ACLs), gateways, and virtual appliances to run containers on Microsoft Azure.

You can set up the following network controls with AKS deployment:

- Agent pools in new and custom virtual networks:

  AKS contains agent nodes in two pools, a public pool and a private pool. An application can be deployed to either pool, affecting accessibility between machines in your container service. The machines can be exposed to the internet (public) or kept internal (private).

  The **private pool** contains the number of agent nodes that you specify when you deploy the DC/OS cluster. The **public pool** initially contains a predetermined number of agent nodes. This pool is added automatically when the DC/OS cluster is provisioned.

- Configurable addressing with PodCIDR and ServiceCIDR.
- Container Networking Interface (CNI) support:

  You can use the CNI plug-in to deploy and manage your own Kubernetes cluster, with native Azure networking capability. The plug-in is enabled, by default, when you deploy Kubernetes clusters with the Azure Container Service Engine (or acs-engine).A pod can connect to other pods and virtual machines in peered virtual networks and to on-premises networks, over ExpressRoute and site-to-site VPN connections. On-premises resources can communicate with pods. Pods in a subnet that has service endpoints enabled can securely connect to Azure services (Azure Storage and SQL Database, for example) by using the CNI plug-in.

- Azure Application Gateway Ingress Controller.

When a customer deploys container applications on Azure resources, they're deployed at a subscription level in resource groups and are not multi-tenant. If that customer shares a subscription with others, no boundaries can be built between two deployments in the same subscription. Container-level security is not guaranteed.

Containers share the kernel and the resources of the host. Containers in production must be run in non-privileged user mode. Running a container with root privileges can compromise the entire environment.

It's important to run containers with read-only file systems. This prevents someone who has access to the compromised container from writing malicious scripts to the file system and gaining access to other files. Similarly, it's important to limit the resources (such as memory) allocated to a container. This helps prevent hackers from hogging resources and pursuing illegal activities, which might prevent other containers from running on the host or cluster.

Production: security tools and polices

After an application is deployed in production, set a few rules to ensure that images used in production environments are secure and contain no vulnerabilities:

- Images with vulnerabilities, even minor, should not be allowed to run in a production environment. Ideally, all images deployed in production should be saved in a private registry accessible to a select few. Keep the number of production images small to ensure that they can be managed effectively.
- Because it's hard to pinpoint the origin of software from a publicly available container image, build images from the source to ensure knowledge of the origin of the layer. When a vulnerability surfaces in a self-built container image, customers can find a quicker path to a resolution. With a public image, customers would need to find the root of a public image to fix it or get another secure image from the publisher.
- A thoroughly scanned image deployed in production is not guaranteed to be up-to-date for the lifetime of the application. Security vulnerabilities might be reported for layers of the image that were not previously known or were introduced after the production deployment. Periodically audit images deployed in production to identify images that are out of date or have not been updated in a while. You might use blue-green deployment methodologies and rolling upgrade mechanisms to update container images without downtime. You can scan images by using tools described in the preceding section.
- A continuous integration (CI) pipeline to build images and integrated security scanning can help maintain secure private registries with secure container images. The vulnerability scanning built into the CI solution ensures that images that pass all the tests are pushed to the private registry from which production workloads are deployed. A CI pipeline failure ensures that vulnerable images are not pushed to the private registry

that's used for production workload deployments. It also automates image security scanning if there's a significant number of images. Otherwise, manually auditing images for security vulnerabilities can be painstakingly lengthy and error prone.

The Azure Marketplace provides other partner tools for full-stack security, including vulnerability scanning, run-time protection, and compliance through Aqua, Sysdig Secure, or Twistlock.

Adding process restrictions

Restricting access and capabilities reduces the surface area that's potentially vulnerable to attack. Docker's default settings are designed to limit Linux capabilities. Although the traditional view of Linux considers OS security in terms of root privileges versus user privileges, modern Linux has evolved to support a more nuanced privilege model: capabilities.

Linux capabilities allow granular specification of user capabilities. Traditionally, the root user has access to every capability. Typical non-root users have a more restricted capability set but are usually given the option to elevate their access to root level by using sudo or setuid binaries. This ability might be a security risk.

The default bounding set of capabilities inside a Docker container is less than half the total capabilities assigned to a Linux process. This reduces the possibility of escalation to a fully privileged root user through application-level vulnerabilities. Docker uses an extra degree of detail, which dramatically expands on the traditional root/non-root dichotomy.

In most cases, the application containers don't need all the capabilities attributed to the root user, because most tasks that require this level of privilege are handled by the OS environment external to the container. Containers can run with a reduced capability set that does not negatively affect the application and yet improves the overall security system levels and makes running applications more secure by default. It's then difficult to provoke system-level damages during intrusion, even if the intruder manages to escalate to root within a container, because the container capabilities are fundamentally restricted.

Security frameworks for container management and orchestration

A container management framework is a solution for building, shipping, and deploying your app in containers. Docker, as a container, has become a standard.

Container management frameworks help you to build a Container as a Service (CaaS) solution. In the CaaS model, IT organizations and developers can work together to build, ship, and run their applications anywhere. CaaS enables an IT-secured and managed application environment that consists of content and infrastructure. From this environment, developers can build and deploy applications in a self-service manner.

Here are some popular container management frameworks:

- Docker Enterprise

- Red Hat OpenShift
- Mesosphere Enterprise DC/OS
- CoreOS Tectonic
- Rancher
- Apcera
- Apprenda

The choice of the orchestration layer often drives the choice of selecting the container management framework. Popular container orchestration tools include:

- Docker Swarm
- Kubernetes
- Apache Mesos

Conclusion

Enterprises should watch container technology to see how they can take advantage of it. As with all nascent technologies, there are caveats to watch out for—and best practices and policies to adhere to—in a container deployment. The Microsoft Azure platform has fully managed container services that are integrated with identity and access management (IAM) capabilities: authentication, authorization, and single sign-on or federation. These services also provide encrypted communications to secure data in transit. And they fully support network controls to lock down kernel resources.

A comprehensive security program uses a mixture of Microsoft Azure and partner solutions to fill in gaps, and it covers much more than just the container runtime environment. Container technology is maturing, but most of what you need to deliver and manage container security is available in the Azure platform today.

Resources

GitHub repositories

- Azure Container Registry
- Manage Azure Container Instances
- Deploy and connect to a Docker container in an Azure cluster

Samples

- Docker samples
- Service Fabric container samples

- Manage with Azure Container Registry - in .NET
- Manage with multiple container images - in .NET
- Manage Container Service with Kubernetes Orchestrator - in .NET