The AKS Checklist

www.the-aks-checklist.com

December 3, 2022

- Application
  - Implement a proper Liveness probe
  - Implement a proper Startup probe
  - Implement a proper Readiness probe
  - Implement a proper prestop hook
  - Run more than one replica for your Deployment
  - Apply tags/labels to all resources
  - Implement autoscaling of your applications
  - Store your secrets in Azure Key Vault, don't inject passwords in Docker Images
  - Implement Azure Workload Identity
  - Use Kubernetes namespaces to properly isolate your Kubernetes resources
  - Set up requests and limits on your containers
  - Specify the security context of your pod/container
  - Ensure your manifests respect good practices
  - Conduct Dockerfile scanning to ensure Docker Image Security Best Practices
  - Static Analysis of Docker Images on Build
  - Threshold enforcement of Docker Image Builds that contain vulnerabilities
  - Compliance enforcement of Docker Image Builds
  - Use Azure Migrate to quick containerize your applications
  - Apply the right deployment type to your application
  - Don't use naked pods
  - Control the usage of imagePullPolicy
  - Use automation through ARM/TF to create your Azure resources
  - Use canary or blue/green deployments
  - Use Dapr to ease microservice development
- Bc Dr
  - Define non-functional requirements such as SLAs, RTO (Recovery Time Objective) and RPO (Recovery Point Objective)
  - Schedule and perform DR tests regularly (whitespace deplyment)
  - Use Availability Zones if supported in your Azure region
  - Plan for multiregion deployment
  - Use Azure Traffic Manager or Azure Front Door as a global load balancer for region failover
  - Create a storage migration plan
  - Use the SLA-backed AKS offering
  - Avoid Pods being placed into a single node
  - If using a private registry, configure region replication to store images in multiple regions
- Cluster Multi

  - Logically isolate cluster
  - Physically isolate cluster
  - Use Azure tags in Azure Kubernetes Service (AKS)

- Cluster Security
  - Use Mariner COS as host OS
  - Configure your cluster for regulated industries
  - Check if you need the Kubernetes dashboard
  - Encrypt ETCD at rest with your own key
  - Maintain kubernetes version up to date
  - Policy blocking the deployment of vulnerable image
  - Use Azure Key Vault
  - Monitor the security of your cluster with Azure Security Center
  - Remove vulnerable images from your cluster
  - Enable Microsoft Defender for Containers
  - Use Azure Policy for Kubernetes to ensure cluster compliance
  - Separate applications from the control plane with user/system nodepools
  - If user Service Principals for the cluster, refresh credentials periodically (like quarterly)
  - Use a private registry for your images, such as ACR
  - Use Azure Security Center to detect security posture vulnerabiities
  - If required consider using Confidential Compute for AKS
  - Define app separation requirements (namespace/nodepool/cluster)
  - If required consider using Azure Dedicated Hosts for AKS nodes

- Container
  - Scan the container image against vulnerabilities
  - Allow deploying containers only from known registries
  - Runtime Security of Applications
  - Quarantine of Docker Images in Docker Registries that have discovered issues
  - Role-Based Access Contol (RBAC) to Docker Registries
  - Network Segmentation of Docker Registries
  - Prefer distroless images
  - Refresh container when base image is updated

- Identity
  - Integrate authentication with AAD (using the managed integration)
  - Integrate authorization with AAD RBAC
  - Use AKS and ACR integration without password
  - Use managed identities instead of Service Principals
  - Limit access to admin kubeconfig (get-credentials –admin)

  - For AKS non-interactive logins use kubelogin
  - Disable AKS local accounts
  - Configure if required Just-in-time cluster access
  - For finer control consider using a managed Kubelet Identity

- Networking
  - Choose the best CNI network plugin for your requirements (Azure CNI recommended)
  - If using Azure CNI, size your subnet accordingly considering the maximum number of pods per node
  - Use an ingress controller to expose web-based apps instead of exposing them with LoadBalancer-type services
  - Secure your exposed applications with a web application firewall (WAF)
  - Apply control on ingress hostnames
  - Don't expose your load-balancer on Internet if not necessary
  - Control traffic flow with network policies
  - Configure default network policies in each namespace
  - Filter egress traffic with AzFW/NVA if your security requirements mandate it (egress lockdown)
  - Don't expose your container registry on Internet
  - Bloc Pod access to VMSS IMDS
  - Use private clusters if your requirements mandate it
  - Enable traffic management
  - If using Azure CNI, check the maximum pods/node (default 30)
  - If using a public API endpoint, restrict the IP addresses that can access it
  - Use Azue NAT Gateway as outboundType for scaling egress traffic
  - Use Dynamic allocations of IPs in order to avoid Azure CNI IP exhaustion
  - If using AGIC, do not share an AppGW across clusters
  - Use Private Endpoints (preferred) or Virtual Network Service Endpoints to access PaaS services from the cluster
  - If required add company HTTP Proxy
  - If required add company HTTP Proxy
  - Consider using a service mesh for advanced microservice communication management
  - If required add your own CNI plugin
  - If using Azure CNI, consider using different Subnets for NodePools

- Operations
  - Have a regular process to upgrade the cluster node images periodically (weekly, for example), or use the AKS autoupgrade feature
  - Use placement proximity group to improve performance

  - Customize your clusters with extensions
  - Customize the name of the MC_ resource group
  - Securely connect to nodes through a bastion host
  - Regularly check for cluster issues
  - Provision a log aggregation tool
  - Monitor your cluster metrics with Container Insights (or other tools like Prometheus)
  - Configure distributed tracing
  - Set Upgrade Channel
  - Enable cluster autoscaling
  - Use GitOps to deploy workloads in your cluster
  - Implement CI/CD to deploy workloads in your cluster
  - Improve efficiency with K8S Tools
  - Don't use the default namespace
  - Apply different types of labels to all resources
  - Fine tune your node configuration
  - Consider using AKS command invoke on private clusters
  - Monitor CPU and memory utilization of the nodes
  - Send master logs (aka API logs) to Azure Monitor or your preferred log management solution
  - Consider an appropriate node size, not too large or too small
  - Regularly check Azure Advisor for recommendations on your cluster
  - Write manifests following good practices using dedicated tools
  - Enable AKS auto-certificate rotation
  - Store and analyze your cluster logs with Container Insights (or other tools like Telegraf/ElasticSearch)
  - Consider AKS virtual node for quick bursting
  - Consider spot node pools for non time-sensitive workloads
  - Configure alerts on the most critical metrics (see Container Insights for recommendations)
  - If not using egress filtering with AzFW/NVA, monitor standard ALB allocated SNAT ports
  - Subscribe to resource health notifications for your AKS cluster
  - Consider subscribing to EventGrid Events for AKS automation

- Resource Management
  - Burst serverless with Azure Container Instances and Virtual Nodes
  - Sizing of the nodes
  - Consider spot node pools for non time-sensitive workloads
  - Use AMD64 nodes when possible

  - Enforce resource quotas for namespaces
  - Namespaces should have LimitRange
  - Set memory limits and requests for all containers
  - Use Disruption Budgets in your pod and deployment definitions
  - Set up cluster auto-scaling
  - Use an external application such as kubecost to allocate costs to different users
  - When required use multi-instance partioning GPU on AKS Clusters
  - If running a Dev/Test cluster use NodePool Start/Stop
  - Ensure your subscription has enough quota to scale out your nodepools

- Storage

  - Choose the right storage type
  - Size the nodes for storage needs
  - Dynamically provision volumes
  - Secure and back up your data
  - Make your storage resilient
  - Use ephemeral OS disks
  - For hyper performance storage option use Ultra Disks on AKS
  - Avoid keeping state in the cluster, and store data outside (AzStorage, AzSQL, Cosmos, etc)
  - If using Azure Disks and AZs, consider having nodepools within a zone for LRS disk

- Windows

  - Map the base image to node OS
  - Prepare your application for an abrupt kill
  - Don't use privileged containers
  - Watch for memory usage
  - Implement CNI network mode
  - Patch your nodes yourself
  - Secure the traffic of your containers
  - Enable Group Managed Service Accounts (GMSA) for your Windows Server nodes
  - If required for AKS Windows workloads, HostProcess containers can be used
  - Taint Windows nodes

Application

Implement a proper Liveness probe

Many applications running for long periods of time eventually transition to broken states, and cannot recover except by being restarted. Kubernetes provides liveness probes to detect and remedy such situations. The probe is here to tell Kubernetes to restart your pod when it is not responding anymore

Documentation

- Configure Liveness, Readiness and Startup Probes

Implement a proper Startup probe

Protect slow starting containers with startup probes. Startup probe allow to delay the initial check by liveness which could cause deadlock or wrong result

Documentation

- Configure Liveness, Readiness and Startup Probes

Implement a proper Readiness probe

Sometimes, applications are temporarily unable to serve traffic. For example, an application might need to load large data or configuration files during startup, or depend on external services after startup. In such cases, you don't want to kill the application, but you don't want to send it requests either. Kubernetes provides readiness probes to detect and mitigate these situations. A pod with containers reporting that they are not ready does not receive traffic through Kubernetes Services.

Documentation

- Configure Liveness, Readiness and Startup Probes

Implement a proper prestop hook

This hook is called immediately before a container is terminated due to an API request or management event such as liveness probe failure, preemption, resource contention and others. It can be used when you have critical process you want to finish or save when your pod is destroyed for any reason

Documentation

- Container Lifecycle Hooks

Run more than one replica for your Deployment

Ensure that your application always configure proper replicas to ensure resiliency in the event of a pod crashing or being evicted.

Documentation

- Pod replicas

Apply tags/labels to all resources

Ensure that your components are tagged, it could be business, security or technical tags and these tags will help to assess or apply relevant policies.

Documentation

- Recommended Labels

Implement autoscaling of your applications

Automatically scale your application to the number of pods required to handle the current load. This can be achieved by using Horizontal Pod Autoscaler for CPU & Memory or by using KEDA for scaling based on other sources

Documentation

- Horizontal Pod Autoscaler

Tools

- Kubernetes Event-driven Autoscaling (KEDA)
- KEDA as AKS addon

Store your secrets in Azure Key Vault, don't inject passwords in Docker Images

Secrets are not encrypted in etcd, prefer to store your secrets in a proper HSM like Azure Key Vault. You can then inject secrets using CSI provider.

Documentation

- Azure Key Vault Provider for Secret Store CSI Driver
- AKV2K8S

Implement Azure Workload Identity

Don't use fixed credentials within pods or container images, as they are at risk of exposure or abuse. Instead, use workload identity federation to access Azure Active Directory (Azure AD) protected resources without needing to manage secrets When pods need access to other Azure services, such as Cosmos DB, Key Vault, or Blob Storage, the pod needs access credentials. These access credentials could be defined with the container image or injected as a Kubernetes secret, but need to be manually created and assigned. Often, the credentials are reused across pods, and aren't regularly rotated. Managed identities for Azure resources (currently implemented as an associated AKS open source project) let you automatically request access to services through Azure AD. You don't manually define credentials for pods, instead they request an access token in real time, and can use it to access only their assigned services.

Documentation

- Use Azure Workload Identity
- Azure Active Directory Pod Identity (deprecated)

Use Kubernetes namespaces to properly isolate your Kubernetes resources

Namespaces give you the ability to create logical partitions and enforce separation of your resources as well as limit the scope of user permissions. Don't forget not to use the Default namespace

Documentation

- Namespaces

Set up requests and limits on your containers

When Containers have resource requests specified, the scheduler can make better decisions about which nodes to place Pods on. And when Containers have their limits specified, contention for resources on a node can be handled in a specified manner.

Documentation

- Managing Compute Resources for Containers
- Take benefit of the Quality of Service

Specify the security context of your pod/container

A security context defines privilege and access control settings for a Pod or Container. Control the capabilities and the rights your container can have. If you don't specify the security context, the pod get the "default" one which may have more rights that it should You should also disable mounting credentials by default (automountServiceAccountToken)

Documentation

- Configure a Security Context for a Pod or Container

Ensure your manifests respect good practices

Best practices inside the cluster start with the configuration of your manifests. Ensure that they respect good practices

Documentation

- Kubernetes YAML: Enforcing best practices and security policies
- 13 Best Practices for Using Helm

Tools

- kube-score
- Checkov
- kubelinter

Conduct Dockerfile scanning to ensure Docker Image Security Best Practices

Define a Image build security baseline for your developers to follow. You should also use scanning tools to detect Dockerfile issue

Documentation

- SNYK 10 Docker Image Security Best Practices
- 21 Best Practises in 2021 for Dockerfile

Tools

- Dockle
- Hadolint

Static Analysis of Docker Images on Build

Introduction of DevSecOps into the environment to promote a proactive security model that starts to shift the responsibility left

Documentation

- Introduction to Microsoft Defender for container registries
- Palo Alto CI/CD Integration (twistcli)
- Qualys CI/CD Integration
- Clair CI/CD Integration

Threshold enforcement of Docker Image Builds that contain vulnerabilities

Restrict builds with identified issues. Use a tool that allows for the restriction of builds with enough granularity to not break development. All Critical CVE's are not the same, so being able to restrict builds based on Critical or High vulnerabilities with a Vendor fix, but allowing builds to continue if that Critical vulnerability is "Open"

Documentation

- Prisma Threshold enforcement

Compliance enforcement of Docker Image Builds

Being able to assess and restrict the Compliance state of an image on build. Identifying an image running as "root" before it get deployed, or opening up port 80 or 22

Documentation

- Azure Built-In Policy
- Prisma Managing Compliance

Use Azure Migrate to quick containerize your applications

The App Containerization tool offers a point-and-containerize approach to repackage applications as containers with minimal to no code changes by using the running state of the application. The tool currently supports containerizing ASP.NET applications and Java web applications running on Apache Tomcat.

Documentation

- Accelerate application modernization with Azure Migrate: App Containerization

Apply the right deployment type to your application

There are a variety of techniques to deploy new applications to production so choosing the right strategy is an important decision that needs to be made to leverage the impact of change on the consumer.

Documentation

- Kubernetes Deployment Strategies

Don't use naked pods

Naked pods are pods not linked to a Replicaset or a Deployment. Naked Pods will not be rescheduled in the event of a node failure.

Control the usage of imagePullPolicy

If this attributes is not properly define kubernetes downloads the container image for each new instance of the containers.

Documentation

- Updating images

Use automation through ARM/TF to create your Azure resources

If this attributes is not properly define kubernetes downloads the container image for each new instance of the containers.

Documentation

- Create a Kubernetes cluster with Azure Kubernetes Service using Terraform

Use canary or blue/green deployments

This will help ensure your workloads remain online during cluster upgrades.

Documentation

- Blue-green deployment for AKS

Use Dapr to ease microservice development

This will help ensure your workloads remain online during cluster upgrades.

Documentation

- Dapr project page

Bc Dr

Define non-functional requirements such as SLAs, RTO (Recovery Time Objective) and RPO (Recovery Point Objective)

Before going to production, it is very important to decide the target of availability and recovery time you are looking for. This will help you to decide the right backup strategy but also the architecture of your solution.

Documentation

- Define the BCDR strategy

Schedule and perform DR tests regularly (whitespace deplyment)

A whitespace (greenfield) deployment is the exercise to delete everything and to redeploy the whole platform in an automated way. In case of an emergency, a security flaw or datacenter failure, it's mandatory for you to be able to restore/create a new environment properly configured in a fully automated way

Documentation

- Schedule and perform DR tests regularly

Use Availability Zones if supported in your Azure region

An Azure Kubernetes Service (AKS) cluster distributes resources such as the nodes and storage across logical sections of the underlying Azure compute infrastructure. This deployment model makes sure that the nodes run across separate update and fault domains in a single Azure datacenter.

Documentation

- Create an AKS cluster across availability zones

Plan for multiregion deployment

When you deploy multiple AKS clusters, choose regions where AKS is available, and use paired regions.

Documentation

- Plan for multiregion deployment

Use Azure Traffic Manager or Azure Front Door as a global load balancer for region failover

Azure Traffic Manager can direct customers to their closest AKS cluster and application instance. For the best performance and redundancy, direct all application traffic through Traffic Manager before it goes to your AKS cluster.

Documentation

- Traffic Manager and AKS
- Cross-region load balancer (Preview)

Create a storage migration plan

Your applications might use Azure Storage for their data. Because your applications are spread across multiple AKS clusters in different regions, you need to keep the storage synchronized

Documentation

- Create a storage migration plan
- Backup, restore and migrate Kubernetes resources including state to another AKS cluster with Velero

Use the SLA-backed AKS offering

Uptime SLA is an optional feature to enable a financially backed, higher SLA for a cluster. It provides you a 99,95% SLA instead of the 99,5% SLO and is relevant for your production clusters Customers needing an SLA to meet compliance requirements or require extending an SLA to their end users should enable this feature. Customers with critical workloads that will benefit from a higher uptime SLA may also benefit. Using the Uptime SLA feature with Availability Zones enables a higher availability for the uptime of the Kubernetes API server.

Documentation

- Azure Kubernetes Service (AKS) Uptime SLA
- Business continuity / disaster recovery to protect and recover AKS

Avoid Pods being placed into a single node

Even if you run several copies of your Pods, there are no guarantees that losing a node won't take down your service. Customers needing an SLA to meet compliance requirements or require extending an SLA to their end users should enable this feature. Customers with critical workloads that will benefit from a higher uptime SLA may also benefit. Using the Uptime SLA feature with Availability Zones enables a higher availability for the uptime of the Kubernetes API server.

Documentation

- Inter-pod affinity and anti-affinity

If using a private registry, configure region replication to store images in multiple regions

Companies that want a local presence, or a hot backup, choose to run services from multiple Azure regions. As a best practice, placing a container registry in each region where images are run allows network-close operations, enabling fast, reliable image layer transfers. Geo-replication enables an Azure container registry to function as a single registry, serving multiple regions with multi-master regional registries.

Documentation

- Enable geo-replication for container images

Cluster Multi

Logically isolate cluster

Use logical isolation to separate teams and projects. Try to minimize the number of physical AKS clusters you deploy to isolate teams or applications

Documentation

- Isolating cluster

Physically isolate cluster

Minimize the use of physical isolation for each separate team or application deployment

Documentation

- Isolating cluster

Use Azure tags in Azure Kubernetes Service (AKS)

With Azure Kubernetes Service (AKS), you can set Azure tags on an AKS cluster and its related resources by using Azure Resource Manager, through the Azure CLI.

Documentation

- Use Azure tags in Azure Kubernetes Service
- Use Azure tags in Azure Kubernetes Service (AKS)

Cluster Security

Use Mariner COS as host OS

Mariner Container-Optimized OS is a first-party operating system for Azure Kubernetes Service (AKS) that is optimized for running Kubernetes workloads. Mariner is a secure, lightweight, and highly performant operating system that is based on the open source project CoreOS Container Linux. Mariner is designed to be a secure, lightweight, and highly performant operating system that is based on the open source project CoreOS Container Linux. Mariner is designed to be a secure, lightweight, and highly performant operating system that is based on the open source project CoreOS Container Linux.

Documentation

- Use the Mariner container host on Azure Kubernetes Service

Configure your cluster for regulated industries

Some industries require certified kubernetes or to implement specific configurations. AKS offers several features to meet this requirements

Documentation

- Use FIPS-enabled node pool

- AKS CIS benchmark
- AKS architecture reference for PCI-DSS 3.2.1

Check if you need the Kubernetes dashboard

Starting with Kubernetes version 1.19, AKS will no longer allow the managed Kubernetes dashboard add-on to be installed for security reasons, and the add-on is scheduled to be deprecated. Ensure the Kubernetes dashboard is not installed on the cluster. It can be done with the following CLI: az aks disable-addons –addons kube-dashboard –resource-group RG_NAME –name CLUSTER_NAME

Encrypt ETCD at rest with your own key

By default, ETCD is encrypted at rest with keys managed by Microsoft. It is possible to encrypt the database using your own key using a KMS plugin and store the key in Azure Key Vault.

Documentation

- Add KMS etcd encryption to an Azure Kubernetes Service (AKS) cluster
- Kubernetes KMS

Maintain kubernetes version up to date

To stay current on new features and bug fixes, regularly upgrade to the Kubernetes version in your AKS cluster. Support for kubernetes is current and N-2 versions only

Documentation

- Regularly update to the latest version of Kubernetes
- Use the auto-upgrade feature

Policy blocking the deployment of vulnerable image

You can now protect your Kubernetes clusters and container workloads from potential threats by restricting deployment of container images with vulnerabilities in their software components.

Tools

- Policy blocking the deployment of vulnerable images

Use Azure Key Vault

Use Azure Key Vault to store Secrets and Certificates

Documentation

- Tutorial: Configure and run the Azure Key Vault provider for the Secrets Store CSI driver on Kubernetes

Monitor the security of your cluster with Azure Security Center

Security Center brings security benefits to your AKS clusters using data already gathered by the AKS master node.

Documentation

- Azure Kubernetes Services integration with Security Center

Remove vulnerable images from your cluster

With image cleaner, you can detect and automatically remove all unused and vulnerable images cached on AKS nodes keeping the nodes cleaner and safer. It's common to use pipelines to build and deploy images on Azure Kubernetes Service (AKS) clusters. While great for image creation, this process often doesn't account for the stale images left behind and can lead to image bloat on cluster nodes. These images can present security issues as they may contain vulnerabilities. By cleaning these unreferenced images, you can remove an area of risk in your clusters. When done manually, this process can be time intensive, which ImageCleaner can mitigate via automatic image identification and removal.

Documentation

- Use ImageCleaner to clean up stale images on your AKS cluster

Enable Microsoft Defender for Containers

Microsoft Defender for Containers provides protections for your Kubernetes clusters wherever they're running (AKS and on-premesis)

Documentation

- Introduction to Microsoft Defender for Containers

Use Azure Policy for Kubernetes to ensure cluster compliance

Azure Policy integrates with the Azure Kubernetes Service (AKS) to apply at-scale enforcements and safeguards on your clusters in a centralized, consistent manner.

Documentation

- Azure Policies for AKS
- Azure Kubernetes Services Governance with Azure Policy

Tools

- Gatekeeper

Separate applications from the control plane with user/system nodepools

Manage system node pools in Azure Kubernetes Service (AKS) and add taint to your system nodepool to make it dedicated

Documentation

- AKS System Pools

If user Service Principals for the cluster, refresh credentials periodically (like quarterly)

You may want to update, or rotate, the credentials as part of a defined security polic

Documentation

- Update or rotate the credentials for AKS

Use a private registry for your images, such as ACR

Azure Container Registry allows you to build, store, and manage container images and artifacts in a private registry

Documentation

- Azure Container Registry documentation

Use Azure Security Center to detect security posture vulnerabiities

Microsoft Defender for Containers is the cloud-native solution that is used to secure your containers

Documentation

- Overview of Microsoft Defender for Containers

If required consider using Confidential Compute for AKS

Containers run within a Trusted Execution Environment(TEE) brings isolation from other containers, the node kernel in a hardware protected, integrity protected attestable environment.

Documentation

- Application enclave support on AKS

Define app separation requirements (namespace/nodepool/cluster)

Plan for multi-tenant clusters and separation of resources and use logical or physical isolation in your AKS clusters

Documentation

- Best practices for cluster isolation in AKS

If required consider using Azure Dedicated Hosts for AKS nodes

Plan for multi-tenant clusters and separation of resources and use logical or physical isolation in your AKS clusters

Documentation

- Add Azure Dedicated Host to an AKS cluster

Container

Scan the container image against vulnerabilities

Scan your container images to ensure there are no vulnerabilities in it

Documentation

- Azure Security Center : scanning feature (Qualys)
- Identify vulnerable container images in your CI/CD workflows

Tools

- Prisma (ex Twistlock)
- Anchore
- Clair

Allow deploying containers only from known registries

One of the most common custom policies that you might want to consider is to restrict the images that can be deployed in your cluster. But it can also be addressed with a proper egress lockdown or using an admission controller

Documentation

- Use the Azure Policy : Ensure only allowed container images in AKS
- Using ImagePolicyWebhook
- Using egress lockdown and authorizing only the URL of your registry

Runtime Security of Applications

Integrate Runtime Security for your pods. To complete the defense in depth structure, ensure Runtime protection is in place to protect from process, network, storage and system call attacks.

Tools

- Prisma Runtime defense
- Falco

Quarantine of Docker Images in Docker Registries that have discovered issues

Use policy to protect images from drift while in the registry, on both push and pull. On build, the image is secured based on the threshold set, but now while in the registry a new issue is discovered. You need to ensure that the image can not be deployed until the issue is remediated.

Documentation

- ACR Quarantine

Role-Based Access Contol (RBAC) to Docker Registries

The Azure Container Registry service supports a set of built-in Azure roles that provide different levels of permissions to an Azure container registry. Use Azure role-based access control (RBAC) to assign specific permissions to users, service principals, or other identities that need to interact with a registry.

Documentation

- Azure Container Registry roles and permissions

Network Segmentation of Docker Registries

Limit access to a registry by assigning virtual network private IP addresses to the registry endpoints and using Azure Private Link. Network traffic between the clients on the virtual network and the registry's private endpoints traverses the virtual network and a private link on the Microsoft backbone network, eliminating exposure from the public internet. Private Link also enables enables private registry access from on-premises through Azure ExpressRoute private peering or a VPN gateway.

Documentation

- Azure Container Registry Private Link

Prefer distroless images

When building a docker image, try to use the distroless version of the base OS image, to reduce the risk of vulnerabilities with preinstalled but unused tools. From example, use base-debian10 instead of debian10 "Distroless" images are bare-bones versions of common base images. They have the bare-minimum needed to execute a binary.The

shell and other developer utilities have been removed so that if/when an attacker gains control of your container, they can't do much of anything

Documentation

- Google distroless images

Refresh container when base image is updated

As you use base images for application images, use automation to build new images when the base image is updated. As those base images typically include security fixes, update any downstream application container images.

Documentation

- Automatically build new images on base image update
- Azure DevOps - Trigger pipeline from Docker image update

Identity

Integrate authentication with AAD (using the managed integration)

Azure Kubernetes Service (AKS) can be configured to use Azure Active Directory (Azure AD) for user authentication. In this configuration, you can sign in to an AKS cluster by using your Azure AD authentication token.

Documentation

- AKS-managed Azure Active Directory integration
- Disable local accounts

Integrate authorization with AAD RBAC

Control access to cluster resources using Kubernetes role-based access control and Azure Active Directory identities in Azure Kubernetes Service

Documentation

- Limit cluster access via K8S RBAC for users & workloads

Use AKS and ACR integration without password

AKS can authenticate to ACR without using any password, but by using either Service Principal or Managed Identity. For AKS to download/pull images from Azure Container Registry (ACR), it needs the ACR credentials including the password. To avoid saving the password in the cluster, you can simply activate the ACR integration on new or existing AKS cluster using SPN or Managed Identity

Documentation

- Authenticate with Azure Container Registry from AKS

Use managed identities instead of Service Principals

Each AKS cluster needs either a Managed Identity or Service Principal. We recommend using Managed Identity in AKS

Documentation

- Use managed identities in Azure Kubernetes Service

Limit access to admin kubeconfig (get-credentials –admin)

To limit who can get that Kubernetes configuration (kubeconfig) information and to limit the permissions they then have, you can use Azure role-based access control (Azure RBAC).

Documentation

- Use Azure RBAC to define access to the Kubernetes configuration file

For AKS non-interactive logins use kubelogin

You can use kubelogin to access the cluster with non-interactive service principal sign-in.

Documentation

- Non-interactive sign in with kubelogin

Disable AKS local accounts

AKS offers users the ability to disable local accounts via a flag, disable-local-accounts

Documentation

- Disable local accounts

Configure if required Just-in-time cluster access

AKS offers users the ability to enable access for a limited time

Documentation

- Configure just-in-time cluster access with Azure AD and AKS

For finer control consider using a managed Kubelet Identity

A Kubelet identity enables access granted to the existing identity prior to cluster creation.

Documentation

- Use a pre-created kubelet managed identity

Networking

Choose the best CNI network plugin for your requirements (Azure CNI recommended)

For integration with existing virtual networks or on-premises networks, use Azure CNI networking in AKS. This network model also allows greater separation of resources and controls in an enterprise environment but be aware of the impact on the network topology/IP ranges. In the future, you'll be able to use the CNI overlay model which gives more control and takes the benefits of both approachs. While Kubenet is the default Kubernetes network plugin, the Container Networking Interface (CNI) is a vendor-neutral protocol that lets the container runtime make requests to a network provider. The Azure CNI assigns IP addresses to pods and nodes, and provides IP address management (IPAM) features as you connect to existing Azure virtual networks. Each node and pod resource receives an IP address in the Azure virtual network, and no additional routing is needed to communicate with other resources or services.

Documentation

- Kubenet vs CNI
- Dynamic IP allocation
- Azure CNI Overlay
- Bring your own CNI
- Add a node pool with a unique subnet

If using Azure CNI, size your subnet accordingly considering the maximum number of pods per node

The size of your virtual network and its subnet must accommodate the number of pods you plan to run and the number of nodes for the cluster. As an example, using CNI, you need one IP for each node + one spare for a new node in case of cluster upgrade, and you need an IP for each pod which can represent hundred of IP addresses

Documentation

- Plan IP addressing for your cluster

Use an ingress controller to expose web-based apps instead of exposing them with LoadBalancer-type services

To distribute HTTP or HTTPS traffic to your applications, use ingress resources and controllers. Ingress controllers provide additional features over a regular Azure load balancer, and can be managed as native Kubernetes resources.

Documentation

- Distribute ingress traffic
- Network concepts: Ingress controllers

Secure your exposed applications with a web application firewall (WAF)

If you plan to host exposed applications, to scan incoming traffic for potential attacks, use a web application firewall (WAF) such as Barracuda WAF for Azure or Azure Application Gateway. These more advanced network resources can also route traffic beyond just HTTP and HTTPS connections or basic SSL termination.

Documentation

- Secure traffic with a web application firewall (WAF)

Apply control on ingress hostnames

When a user creates an Ingress manifest, they can use any hostname in it. You may want to control which hostnames are allowed to use, like your company's hostnames.

Documentation

- Tutorial: only allow approved domain names as ingress hostnames

Don't expose your load-balancer on Internet if not necessary

There is almost no reason to directly expose the ingress entry point to Internet but by default AKS create a public one. Tell him to create an internal one only.

Documentation

- Create an ingress controller to an internal virtual network

Control traffic flow with network policies

Use network policies to allow or deny traffic to pods. By default, all traffic is allowed between pods within a cluster. For improved security, define rules that limit pod communication. Network policy is a Kubernetes feature that lets you control the traffic flow between pods. You can choose to allow or deny traffic based on settings such as assigned labels, namespace, or traffic port. The use of network policies gives a cloud-native way to control the flow of traffic. As pods are dynamically created in an AKS cluster, the required network policies can be automatically applied. Don't use Azure network security groups to control pod-to-pod traffic, use network policies.

Documentation

- Enable a Kubernetes Network Policy option (Calico/Azure)
- Secure traffic between pods using network policies

Tools

- Calico
- Cillium

Configure default network policies in each namespace

Start by creating a deny all policy in each namespace and then add specific policies.

Documentation

- Recipes of best default network policies
- Callico global network policy

Filter egress traffic with AzFW/NVA if your security requirements mandate it (egress lockdown)

Use Azure Firewall to secure and control all egress traffic going outside of the cluster.

Documentation

- Egress traffic requirements

Don't expose your container registry on Internet

When possible, use private link to only allow private network to reach your registry.

Documentation

- Azure Container Registry Private Link

Bloc Pod access to VMSS IMDS

By default, Pods have access to VMSS IMDS and can request access token from the attached Managed Identity. This access should be restriced by using Network Policy.

Documentation

- Pods requesting access to get a token

Use private clusters if your requirements mandate it

In a private cluster, the control plane or API server has internal IP addresses and is not exposed to Internet By using a private cluster, you can ensure that network traffic between your API server and your node pools remains on the private network only. Because the API server has a private address, it means that to access it for administration or for deployment, you need to set up private connection, like using a "jumpbox" (i.e.: Azure Bastion)

Documentation

- Create a private cluster (using private link)
- Create an Azure Kubernetes Service cluster with API Server VNet Integration (preview)

- Use azure CLI to run command on a private cluster
- Use public DNS with a private cluster

Enable traffic management

Azure Traffic Manager can direct customers to their closest AKS cluster and application instance. For the best performance and redundancy, direct all application traffic through Traffic Manager before it goes to your AKS cluster. If you have multiple AKS clusters in different regions, use Traffic Manager to control how traffic flows to the applications that run in each cluster. Azure Traffic Manager is a DNS-based traffic load balancer that can distribute network traffic across regions. Use Traffic Manager to route users based on cluster response time or based on geography. It can be used to improve app availability with automatic failover

Documentation

- Use Azure Traffic Manager to route traffic

If using Azure CNI, check the maximum pods/node (default 30)

With Azure Container Networking Interface (CNI), every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be unique across your network space, and must be planned in advance. Each node has a configuration parameter for the maximum number of pods that it supports.

Documentation

- Configure Azure CNI networking on AKS

If using a public API endpoint, restrict the IP addresses that can access it

To improve the security of your clusters and minimize the risk of attacks, we recommend limiting the IP address ranges that can access the API server.

Documentation

- Secure access to the API server using authorized IP address ranges

Use Azue NAT Gateway as outboundType for scaling egress traffic

Whilst AKS customers are able to route egress traffic through an Azure Load Balancer, there are limitations on the amount of outbound flows of traffic that is possible.

Documentation

- Managed NAT Gateway

Use Dynamic allocations of IPs in order to avoid Azure CNI IP exhaustion

A drawback with the traditional CNI is the exhaustion of pod IP addresses as the AKS cluster grows, resulting in the need to rebuild the entire cluster in a bigger subnet. The new dynamic IP allocation capability in Azure CNI solves this problem by allocating pod IPs from a subnet separate from the subnet hosting the AKS cluster.

Documentation

- Dynamic allocation of IPs and enhanced subnet support

If using AGIC, do not share an AppGW across clusters

The App Gateway Ingress Controller (AGIC) is a pod within your Kubernetes cluster. AGIC monitors the Kubernetes Ingress resources, and creates and applies App Gateway config based on these.

Documentation

- Application gateway ingress controller

Use Private Endpoints (preferred) or Virtual Network Service Endpoints to access PaaS services from the cluster

This will help keep your cluster secure by limiting access from outside the virtual network to Azure PaaS services

Documentation

- What is Azure Private Link?

If required add company HTTP Proxy

Http Proxy exposes a straightforward interface that cluster operators can use to secure AKS-required network traffic in proxy-dependent environments.

Documentation

- HTTP proxy support in Azure Kubernetes Service

If required add company HTTP Proxy

Http Proxy exposes a straightforward interface that cluster operators can use to secure AKS-required network traffic in proxy-dependent environments.

Documentation

- HTTP proxy support in Azure Kubernetes Service

Consider using a service mesh for advanced microservice communication management

A service mesh provides capabilities like traffic management, resiliency, policy, security, strong identity, and observability to your workloads.

Documentation

- About service meshes

If required add your own CNI plugin

A service mesh provides capabilities like traffic management, resiliency, policy, security, strong identity, and observability to your workloads.

Documentation

- Bring your own Container Network Interface (CNI) plugin for AKS

If using Azure CNI, consider using different Subnets for NodePools

This can address requirements such as having non-contiguous virtual network address space to split across node pools and other security reasons.

Documentation

- Add a node pool with a unique subnet

Operations

Have a regular process to upgrade the cluster node images periodically (weekly, for example), or use the AKS autoupgrade feature

AKS supports upgrading the images on a node so you're up to date with the newest OS and runtime updates. AKS provides one new image per week with the latest updates, so it's beneficial to upgrade your node's images regularly for the latest features, including Linux or Windows patches Using automation and this method will ensure that all your nodes are consistently up to date with last features/fixes/patchs, without having to upgrade the kubernetes version. An alternative could be to use Kured to reboot nodes with pending reboots but it will only patch the Operating System, not the AKS layer

Documentation

- Azure Kubernetes Service (AKS) node image upgrades
- Process Linux node updates and reboots using Kured (not recommended because it can behave incorrectly in some cluster configurations like autoscaling)
- Use Event Grid to know when an upgrade is available

Tools

- Kured (KUbernetes REboot Daemon)

Use placement proximity group to improve performance

When deploying your application in Azure, spreading Virtual Machine (VM) instances across regions or availability zones creates network latency, which may impact the overall performance of your application. A proximity placement group is a logical grouping used to make sure Azure compute resources are physically located close to each other Be careful, by using PPG on a nodepool, you reduce the average SLA of your application since they don't rely on availability zones anymore

Documentation

- Reduce latency with proximity placement groups

Customize your clusters with extensions

Cluster extensions provides an Azure Resource Manager driven experience for installation and lifecycle management of services like Azure Machine Learning (ML) on an AKS cluster. This feature enables:Azure Resource Manager-based deployment of extensions, including at-scale deployments across AKS clusters but also lifecycle management of the extension (Update, Delete) from Azure Resource Manager.

Documentation

- Customize node configuration for Azure Kubernetes Service (AKS) node pools

Customize the name of the MC_ resource group

When creating a new AKS cluster, the MC_ resource group is created by default. You can customize the name of the MC_ resource group.

Documentation

- Custom resource group name

Securely connect to nodes through a bastion host

Don't expose remote connectivity to your AKS nodes. Create a bastion host, or jump box, in a management virtual network. Use the bastion host to securely route traffic into your AKS cluster to remote management tasks.

Documentation

- Securely connect to nodes through a bastion host

Regularly check for cluster issues

Regularly run the latest version of cluster scanning open source tool to detect issues in your cluster. For instance, if you apply resource quotas on an existing AKS cluster, run kubestriker first to find pods that don't have resource requests and limits defined.

Tools

- AKS Periscope
- kubestriker
- kubebench

Provision a log aggregation tool

Ensure that you are always aware of what happens in your cluster. Monitor the health of the cluster (nodes, server) but also the pods

Documentation

- Azure Monitor for AKS
- Azure Managed Grafana

Tools

- Elastic Cloud
- Datadog

Monitor your cluster metrics with Container Insights (or other tools like Prometheus)

If default integration can collect telemetry data and basic metrics (CPU/Memory), they don't collect custom metrics and more detailed information. It's often necessary to install a 3rd party software (prometheus is recommend within Kubernetes) and they store these metrics to exploit them. Typically, to use Prometheus, you need to set up and manage a Prometheus server with a store. By integrating with Azure Monitor, a Prometheus server is not required. You just need to expose the Prometheus metrics endpoint through your exporters or pods (application), and the containerized agent for Azure Monitor for containers can scrape the metrics for you.

Documentation

- Configure scraping of Prometheus metrics
- Deploying ELK

Configure distributed tracing

Distributed tracing, also called distributed request tracing, is a method used to profile and monitor applications, especially those built using a microservices architecture. Distributed tracing helps pinpoint where failures occur and what causes poor performance.

Documentation

- Solution for onboarding Kubernetes/AKS workloads onto Application Insights monitoring.
- Zero instrumentation application monitoring for Kubernetes hosted applications (deprecated)

Set Upgrade Channel

In addition to manually upgrading a cluster, you can set an auto-upgrade channel on your cluster

Documentation

- Set AKS auto-upgrade channel

Enable cluster autoscaling

To keep up with application demands in Azure Kubernetes Service (AKS), you may need to adjust the number of nodes that run your workloads. The cluster autoscaler component can watch for pods in your cluster that can't be scheduled because of resource constraints. You can enable autoscaling module per node pool but only create one mutual autoscale profile

Documentation

- AKS Autoscaler
- Cluster autoscaler

Use GitOps to deploy workloads in your cluster

GitOps works by using Git as a single source of truth for declarative infrastructure and applications On Azure you can for instance use Azure Arc for Kubernetes but also directly GitOps addon for AKS

Documentation

- Use Flux v2 addon for AKS
- What is Azure Arc enabled Kubernetes?
- Guide To GitOps

Implement CI/CD to deploy workloads in your cluster

Instead of GitOps (or in addition) you can use CI/CD to deploy workloads/tools in your cluster

Documentation

- Automated Deployments for Azure Kubernetes Service (Preview)
- Build and deploy to Azure Kubernetes Service with Azure Pipelines
- Tutorial: Deploy from GitHub to Azure Kubernetes Service using Jenkins

Improve efficiency with K8S Tools

The Kubernetes ecosystem is strengthened by many tools that make operating it easier. Here's a few

Tools

- Lens
- Helm
- kubectl aliases
- kubectx
- k9s

Don't use the default namespace

It's recommended to keep all applications in a namespace other than default

Apply different types of labels to all resources

A common set of labels allows tools to work interoperably, describing objects in a common manner that all tools can understand. For instance, resources should have technical, business and security labels.

Documentation

- Recommended labels

Fine tune your node configuration

Customizing your node configuration allows you to configure or tune your operating system (OS) settings or the kubelet parameters to match the needs of the workloads. When you create an AKS cluster or add a node pool to your cluster, you can customize a subset of commonly used OS and kubelet settings.

Documentation

- Customize node configuration for Azure Kubernetes Service (AKS) node pools

Consider using AKS command invoke on private clusters

You can use command invoke to access private clusters without having to configure a VPN or Express Route Using command invoke allows you to remotely invoke commands

like kubectl and helm on your private cluster through the Azure API without directly connecting to the cluster.

Documentation

- Use command invoke to access a private AKS

Monitor CPU and memory utilization of the nodes

With Container insights, you can use the performance charts and health status to monitor the workload of Kubernetes clusters hosted on AKS

Documentation

- Monitor your Kubernetes cluster performance

Send master logs (aka API logs) to Azure Monitor or your preferred log management solution

To help troubleshoot your application and services, you may need to view the logs generated by the master components. Be aware that if you don't enable these logs, there is no way for Microsoft to retrieve them for you

Documentation

- Enable and review Kubernetes master node logs
- Monitoring Azure Kubernetes Service (AKS) with Azure Monitor

Consider an appropriate node size, not too large or too small

Optimizing node size can lead to lots of benefits including cost optimization, optimized performance and proper resource allocation

Documentation

- Which VM size should I choose as AKS node?

Regularly check Azure Advisor for recommendations on your cluster

Azure advisor can enhance environment monitoring and can help improve the operations of your cluster by reducing costs, etc

Documentation

- Azure advisor get started

Write manifests following good practices using dedicated tools

There are various tools that can make creating k8s manifest files much easier and can help with operational efficiency

Documentation

- what is prettier

Enable AKS auto-certificate rotation

Periodically, you may need to rotate those certificates for security or policy reasons. For example, you may have a policy to rotate all your certificates every 90 days.

Documentation

- Certificate rotation in Azure Kubernetes Service (AKS)

Store and analyze your cluster logs with Container Insights (or other tools like Telegraf/ElasticSearch)

Monitoring the logs from your cluster can help prevent issues and identify areas of improvements

Documentation

- Container insights overview

Consider AKS virtual node for quick bursting

Virtual nodes with ACI can help speed up scaling

Documentation

- Create and configure an AKS cluster to use virtual nodes

Consider spot node pools for non time-sensitive workloads

Spot nodepools can help with cost management

Documentation

- Add an Azure Spot node pool to an AKS cluster

Configure alerts on the most critical metrics (see Container Insights for recommendations)

Alerts can provide early warnings of issues, etc in your cluster

Documentation

- Metric alert rules in Container insights

If not using egress filtering with AzFW/NVA, monitor standard ALB allocated SNAT ports

Monitoring these ports are important for security reasons

Documentation

- Use a public standard load balancer in Azure Kubernetes Service

Subscribe to resource health notifications for your AKS cluster

Learn how to troubleshoot resource health issues

Documentation

- Azure Kubernetes Services troubleshooting documentation

Consider subscribing to EventGrid Events for AKS automation

AKS emits certain events that enable it to subscribe to EventGrid which can be used for automation

Documentation

- AKS as an Event Grid source

Resource Management

Burst serverless with Azure Container Instances and Virtual Nodes

To rapidly scale your AKS cluster, you can integrate with Azure Container Instances (ACI) Kubernetes has built-in components to scale the replica and node count. However,

if your application needs to rapidly scale, the horizontal pod autoscaler may schedule more pods than can be provided by the existing compute resources in the node pool.

Documentation

- Burst to Azure Container Instances

Sizing of the nodes

what type of worker nodes should I use, and how many of them is a critical question which requires the analysis of the workloads deployed on your cluster to get the best values of it Choosing on one hand between easy management and big blast radius, and on the other end to focus on high replication, low impact but worse resources optimization

Documentation

- Choosing a worker node size
- Choose the right storage type

Tools

- Kubernetes instance calculator

Consider spot node pools for non time-sensitive workloads

Using Spot VMs for nodes with your AKS cluster allows you to take advantage of unutilized capacity in Azure at a significant cost savings

Documentation

- Add an Azure Spot node pool to an Azure Kubernetes Service (AKS) cluster

Use AMD64 nodes when possible

Benefit from ARM64's better price, compute performance, and lower power utilization in AKS node pools.

Documentation

- Add an ARM64 node pool

Enforce resource quotas for namespaces

Plan and apply resource quotas at the namespace level. If pods don't define resource requests and limits, reject the deployment. Monitor resource usage and adjust quotas as needed. Resource requests and limits are placed in the pod specification. These limits are used by the Kubernetes scheduler at deployment time to find an available node in the cluster. But developers can forget them and thus impact other applications by over-consuming resources of the cluster

Documentation

- Enforce resource quotas
- Resources quotas

Namespaces should have LimitRange

A LimitRange is a policy to constrain resource allocations (to Pods or Containers) in a namespace. It's useful to ensure that pods don't forget to declare request limits

Documentation

- LimitRange

Set memory limits and requests for all containers

Set CPU and memory limits and requests to all the containers. It prevents memory leaks and CPU over-usage and protects the whole platform When you specify limits for CPU and memory, each takes a different action when it reaches the specified limit. With CPU limits, the container is throttled from using more than its specified limit. With memory limits, the pod is restarted if it reaches its limit. The pod might be restarted on the same host or a different host within the cluster.

Documentation

- Define pod resource requests and limits
- Assign Memory Resources to container

Use Disruption Budgets in your pod and deployment definitions

To maintain the availability of applications, define Pod Disruption Budgets (PDBs) to make sure that a minimum number of pods are available in the cluster. At some point in time, Kubernetes might need to evict pods from a host. There are two types of

evictions: voluntary and involuntary disruptions. Involuntary disruptions can be caused by hardware failure, network partitions, kernel panics, or a node being out of resources. Voluntary evictions can be caused by performing maintenance on the cluster, the Cluster Autoscaler deallocating nodes, or updating pod templates. To minimize the impact to your application, you can set a PodDisruptionBudget to ensure uptime of the application when pods need to be evicted. A PodDisruptionBudget allows you to set a policy on the minimum available and maximum unavailable pods during voluntary eviction events. An example of a voluntary eviction would be when draining a node to perform maintenance on the node.

Documentation

- Plan for availability using pod disruption budgets
- Specifying a Disruption Budget for your Application

Set up cluster auto-scaling

To maintain the availability of applications and guarantee available resources, set up cluster auto-scaling

Documentation

- Use AKS cluster auto-scale

Use an external application such as kubecost to allocate costs to different users

These tools can be very valuable for cost allocation especially in the case of multitenancy

Documentation

- cost governance with kubecost

When required use multi-instance partioning GPU on AKS Clusters

Nvidia's A100 GPU can be divided in up to seven independent instances. Each instance has their own memory and Stream Multiprocessor (SM).

Documentation

- Multi-instance GPU Node pool

If running a Dev/Test cluster use NodePool Start/Stop

Your AKS workloads may not need to run continuously, for example a development cluster that has node pools running specific workloads.

Documentation

- Start and stop an Azure Kubernetes Service (AKS) node pool

Ensure your subscription has enough quota to scale out your nodepools

Azure VMs/nodes have quotas per subscription. Ensure that you have planned for these quota and proactively prepare to request for more if needed

Documentation

- Azure subscription and service limits, quotas, and constraints

Storage

Choose the right storage type

Understand the needs of your application to pick the right storage. Use high performance, SSD-backed storage for production workloads. Plan for network-based storage when there is a need for multiple concurrent connections.

Documentation

- Storage considerations for Azure Kubernetes Service
- Choose the right storage type

Size the nodes for storage needs

Each node size supports a maximum number of disks. Different node sizes also provide different amounts of local storage and network bandwidth. Plan for your application demands to deploy the appropriate size of nodes. Different types and sizes of nodes are available. Each node (underlying VM) size provides a different amount of core resources such as CPU and memory. These VM sizes have a maximum number of disks that can be attached. Storage performance also varies between VM sizes for the maximum local and attached disk IOPS (input/output operations per second). If your applications require

Azure Disks as their storage solution, plan for and choose an appropriate node VM size. The amount of CPU and memory isn't the only factor when you choose a VM size. The storage capabilities are also important.

Documentation

- Size the nodes for storage needs

Dynamically provision volumes

To reduce management overhead and let you scale, don't statically create and assign persistent volumes. Use dynamic provisioning. In your storage classes, define the appropriate reclaim policy to minimize unneeded storage costs once pods are deleted.

Documentation

- Dynamically provision volumes

Secure and back up your data

Back up your data using an appropriate tool for your storage type, such as Velero or Azure Site Recovery Understand the limitations of the different approaches to data backups and if you need to quiesce your data prior to snapshot. Data backups don't necessarily let you restore your application environment of cluster deployment.

Documentation

- Secure and back up your data

Make your storage resilient

Where possible, don't store service state inside the container. Instead, use an Azure platform as a service (PaaS) that supports multiregion replication. Service state refers to the in-memory or on-disk data that a service requires to function. State includes the data structures and member variables that the service reads and writes. Depending on how the service is architected, the state might also include files or other resources that are stored on the disk. For example, the state might include the files a database uses to store data and transaction logs.

Documentation

- Remove service state from inside containers

Use ephemeral OS disks

With ephemeral OS disks, you see lower read/write latency on the OS disk of AKS agent nodes since the disk is locally attached. You will also get faster cluster operations like scale or upgrade thanks to faster re-imaging and boot times

Documentation

- Everything you want to know about ephemeral OS disks and AKS

For hyper performance storage option use Ultra Disks on AKS

Azure ultra disks offer high throughput, high IOPS, and consistent low latency disk storage for your stateful applications. One major benefit of ultra disks is the ability to dynamically change the performance of the SSD along with your workloads without the need to restart your agent nodes. Ultra disks are suited for data-intensive workloads.

Documentation

- Use Azure ultra disks on AKS

Avoid keeping state in the cluster, and store data outside (AzStorage, AzSQL, Cosmos, etc)

Where possible, don't store service state inside the container. Instead, use an Azure platform as a service (PaaS) that supports multiregion replication. Service state refers to the in-memory or on-disk data that a service requires to function. State includes the data structures and member variables that the service reads and writes. Depending on how the service is architected, the state might also include files or other resources that are stored on the disk. For example, the state might include the files a database uses to store data and transaction logs.

Documentation

- Remove service state from inside containers

If using Azure Disks and AZs, consider having nodepools within a zone for LRS disk

If using Azure Disks and AZs, consider having nodepools within a zone for LRS disk with VolumeBindingMode:WaitForFirstConsumer for provisioning storage in right zone or use ZRS disk for nodepools spanning multiple zones

Documentation

- Azure disk availability zone support

Windows

Map the base image to node OS

Windows has strict compatibility rules, where the host OS version must match the container base image OS version. Only Windows containers with a container operating system of Windows Server 2019 are supported.

Documentation

- Windows container version compatibility
- Limitations of Windows containers

Prepare your application for an abrupt kill

TerminationGracePeriod is not implemented on Windows containers

Documentation

- Understand pod lifecycle
- Limitations of Windows containers

Don't use privileged containers

Windows containers do not support elevation of privilege

Documentation

- Limitations of Windows containers

Watch for memory usage

There are no OOM eviction actions taken by the kubelet. Windows always treats all user-mode memory allocations as virtual, and pagefiles are mandatory. If memory is over-provisioned and all physical memory is exhausted, then paging can slow down performance.

Documentation

- Limitations of Windows containers

Implement CNI network mode

AKS clusters with Windows node pools must use the Azure CNI (advanced) networking model. Kubenet (basic) networking is not supported.

Documentation

- What network plug-ins are supported?

Patch your nodes yourself

Windows Server nodes in AKS must be upgraded to get the latest patch fixes and updates. Windows Updates are not enabled on nodes in AKS. AKS releases new node pool images as soon as patches are available, it is the customers responsibility to upgrade node pools to stay current on patches and hotfix.

Documentation

- How do patch my Windows nodes?

Secure the traffic of your containers

Network policies are currently not supported, ensure that the containerized applications have a layer of protection like authentication

Documentation

- Limitations of Windows containers

Enable Group Managed Service Accounts (GMSA) for your Windows Server nodes

Group Managed Service Accounts (GMSA) is a managed domain account for multiple servers that provides automatic password management, simplified service principal name (SPN) management and the ability to delegate the management to other administrators.

Documentation

- Enable Group Managed Service Accounts (GMSA) for your Windows Server nodes

If required for AKS Windows workloads, HostProcess containers can be used

HostProcess / Privileged containers extend the Windows container model to enable a wider range of Kubernetes cluster management scenarios.

Documentation

- Use Windows HostProcess containers

Taint Windows nodes

Adding taints, tolerations and node selectors will help avoid having pods scheduled in the wrong OS node pool

Documentation

- Adapt apps for mixed-OS Kubernetes clusters using node selectors or taints and tolerations