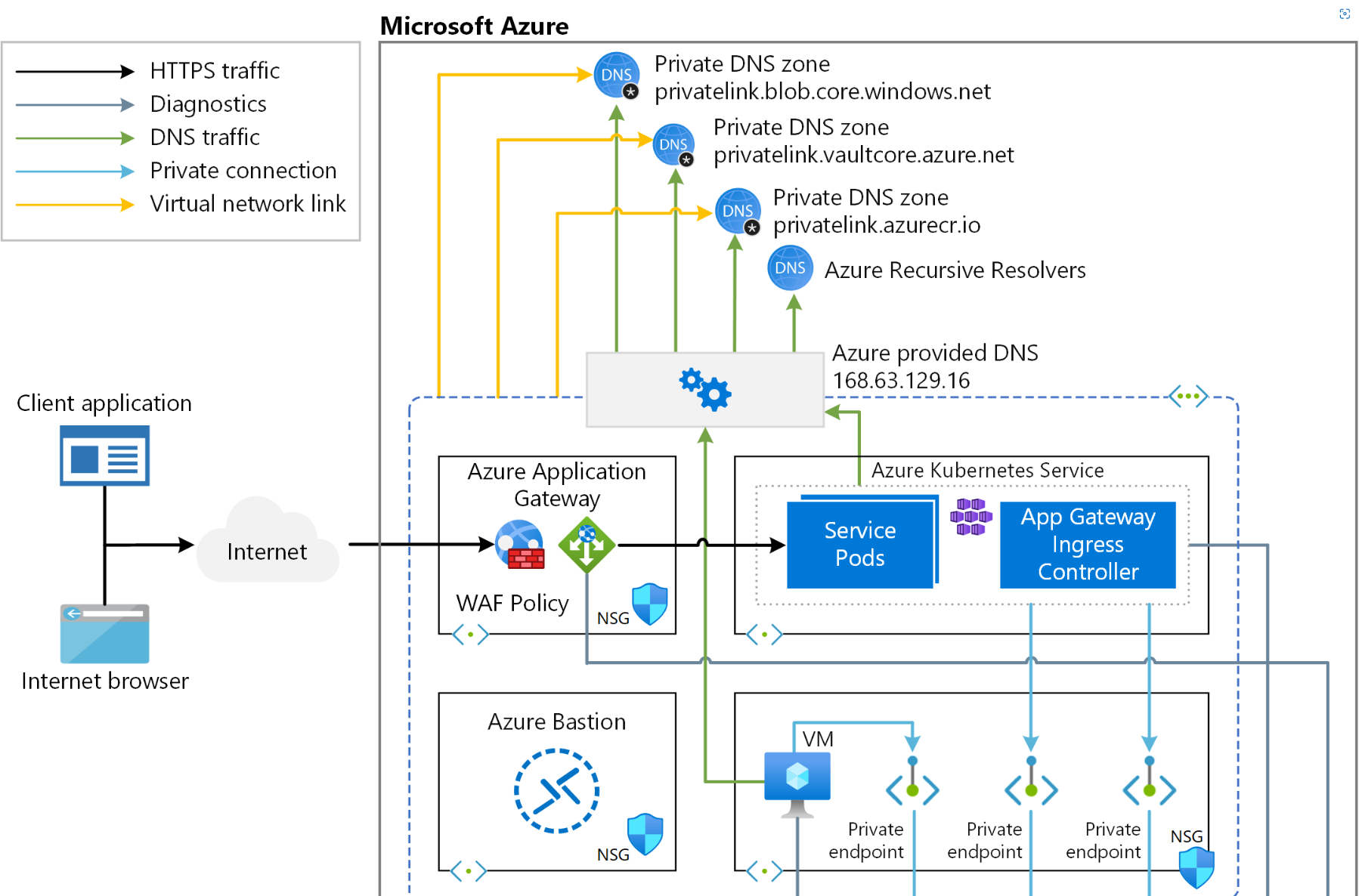
In this solution, [Azure Web Application Firewall (WAF)](https://docs.microsoft.com/en-us/azure/web-application-firewall/ag/ag-overview) on Azure Application Gateway provides centralized protection for web applications deployed on a multi-tenant Azure Kubernetes Service (AKS) cluster from common exploits and vulnerabilities. Web applications running on [Azure Kubernetes Service (AKS) cluster](https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes) and exposed via the [Application Gateway Ingress Controller (AGIC)](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview) can be protected from malicious attacks, such as SQL injection and cross-site scripting, by the [WAF Policy](https://docs.microsoft.com/en-us/azure/web-application-firewall/ag/create-waf-policy-ag) in Azure Application Gateway. WAF Policy on Azure Application Gateway comes pre-configured with OWASP core rule sets 3.1 version and can be changed to other supported OWASP CRS versions.

## Architecture

****

Download a [*Visio file*](https://arch-center.azureedge.net/aks-agic.vsdx) of this architecture.

**Workflow**

The companion ARM template deploys a new virtual network with four subnets:

* AksSubnet: Hosts the AKS cluster
* VmSubnet: Hosts a jump-box virtual machine and private endpoints
* AppGatewaySubnet: Hosts Azure Application Gateway in WAF v2 tier
* AzureBastionSubnet: Hosts Azure Bastion

The Azure Kubernetes Service (AKS) cluster uses a user-defined managed identity to create additional resources, such as load balancers and managed disks. The ARM template deploys an AKS cluster with the following features:

* [Container Storage Interface (CSI) drivers for Azure disks and Azure Files](https://docs.microsoft.com/en-us/azure/aks/csi-storage-drivers)
* [AKS-managed AAD integration](https://docs.microsoft.com/en-us/azure/aks/managed-aad)
* [Azure RBAC for Kubernetes Authorization](https://docs.microsoft.com/en-us/azure/aks/manage-azure-rbac)
* [Managed identity in place of a service principal](https://docs.microsoft.com/en-us/azure/aks/use-managed-identity)
* [Azure Active Directory pod-managed identities](https://docs.microsoft.com/en-us/azure/aks/use-azure-ad-pod-identity)
* [Azure Network Policies](https://docs.microsoft.com/en-us/azure/aks/use-network-policies)
* [Azure Monitor container insights add-on](https://docs.microsoft.com/en-us/azure/azure-monitor/containers/container-insights-enable-new-cluster)
* [Application Gateway Ingress Controller add-on](https://azure.github.io/application-gateway-kubernetes-ingress/)
* [Dynamic allocation of IPs and enhanced subnet support](https://docs.microsoft.com/en-us/azure/aks/configure-azure-cni#dynamic-allocation-of-ips-and-enhanced-subnet-support-preview)

The AKS cluster is composed of the following:

* System node pool that hosts only critical system pods and services. The nodes in system node pool have the taint CriticalAddonsOnly=true:NoSchedule that prevents application pods from beings scheduled on this node pool.
* User node pool that hosts user workloads and artifacts.

A virtual machine (VM) is deployed in the same virtual network that is hosting the AKS cluster. When you deploy Azure Kubernetes Service as a private cluster, this VM can be used by system administrators to manage the cluster via the [Kubernetes command-line tool](https://kubernetes.io/docs/tasks/tools/). The boot diagnostics logs of the virtual machine are stored in an Azure Storage account.

An Azure Bastion host provides secure and seamless SSH connectivity to the jump-box VM, directly in the Azure portal over HTTPS.

An Azure Container Registry (ACR) is used to build, store, and manage container images and artifacts (such as Helm charts).

The architecture includes an Application Gateway that is used by the ingress controller. The Application Gateway is deployed to a dedicated subnet and exposed to the public internet via a public IP address that is shared by all the tenant workloads. A Web Access Firewall (WAF) Policy is associated to the Application Gateway at the root level, at individual listeners, or at path-based routing rules to protect tenant workloads from malicious attacks. The policy is configured in Prevention mode and uses [OWASP 3.1](https://owasp.org/www-project-application-security-verification-standard) to block intrusions and attacks that are detected by rules. The attacker receives a "403 unauthorized access" exception, and the connection is closed. Prevention mode records these attacks in the WAF logs.

Application Gateway Ingress Controller (AGIC) is deployed to the Kubernetes cluster. The Application Gateway Ingress Controller (AGIC) is a Kubernetes application, which makes it possible for [Azure Kubernetes Service (AKS)](https://azure.microsoft.com/services/kubernetes-service/) customers to leverage Azure's native [Application Gateway](https://azure.microsoft.com/services/application-gateway/) L7 load-balancer to expose cloud software to the Internet. AGIC monitors the Kubernetes cluster it is hosted on and continuously updates an Application Gateway, so that selected services are exposed to the Internet.

A Key Vault is used as a secret store by workloads that run on Azure Kubernetes Service (AKS) to retrieve keys, certificates, and secrets via a client library, [Secrets Store CSI Driver](https://docs.microsoft.com/en-us/azure/aks/csi-secrets-store-driver), or [Dapr](https://docs.dapr.io/developing-applications/building-blocks/secrets/secrets-overview). [Azure Private Link](https://docs.microsoft.com/en-us/azure/private-link/private-link-overview) enables AKS workloads to access Azure PaaS Services, such as Key Vault, over a private endpoint in the virtual network.

The sample topology includes the following private endpoints:

* A private endpoint to the Blob Storage account
* A private endpoint to Azure Container Registry (ACR)
* A private endpoint to Key Vault
* If you opt for a private AKS cluster, a private endpoint to the API server of the Kubernetes cluster

The architecture also includes the following Private DNS Zones for the name resolution of the fully qualified domain name (FQDN) of a PaaS service to the private IP address of the associated private endpoint:

* A Private DNS Zone for the name resolution of the private endpoint to the Azure Blob Storage account
* A Private DNS Zone for the name resolution of the private endpoint to Azure Container Registry (ACR)
* A Private DNS Zone for the name resolution of the private endpoint to Azure Key Vault
* If you deploy the cluster as private, a Private DNS Zone for the name resolution of the private endpoint to the Kubernetes Server API

A Virtual Network Link exists between the virtual network hosting the AKS cluster and the above Private DNS Zones. A Log Analytics workspace is used to collect the diagnostics logs and metrics from the following sources:

* Azure Kubernetes Service cluster
* Jump-box virtual machine
* Azure Application Gateway
* Azure Key Vault
* Azure network security groups (NSG)

### Components

* [Azure Container Registry](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-intro) is a managed, private Docker registry service based on the open-source Docker Registry 2.0. You can use Azure container registries with your existing container development and deployment pipelines, or use Azure Container Registry Tasks to build container images in Azure. Build on-demand, or or automate builds triggered by source code updates, updates to a container's base image, or based on timers.
* [Azure Kubernetes Services](https://docs.microsoft.com/en-us/azure/aks/) simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance. Since Kubernetes masters are managed by Azure, you only manage and maintain the agent nodes.
* [Azure Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/general/overview/) securely stores and controls access to secrets like API keys, passwords, certificates, and cryptographic keys. Azure Key Vault also lets you easily provision, manage, and deploy public and private Transport Layer Security/Secure Sockets Layer (TLS/SSL) certificates, for use with Azure and your internal connected resources.
* [Azure Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview)  Azure Application Gateway is a web traffic load balancer that enables you to manage traffic to your web applications. Traditional load balancers operate at the transport layer (OSI layer 4 - TCP and UDP) and route traffic based on source IP address and port, to a destination IP address and port. Application Gateway can make routing decisions based on additional attributes of an HTTP request, for example URI path or host headers. This type of routing is known as application layer (OSI layer 7) load balancing. Azure Application Gateway can do URL-based routing and more.
* [Web Application Firewall](https://docs.microsoft.com/en-us/azure/application-gateway/waf-overview) or WAF is a service that provides centralized protection of web applications from common exploits and vulnerabilities. WAF can be deployed with Azure Application Gateway, Azure Front Door, and Azure Content Delivery Network (CDN) service from Microsoft. WAF is based on rules from the [OWASP (Open Web Application Security Project) core rule sets](https://owasp.org/www-project-modsecurity-core-rule-set). Azure WAF allows you to create custom rules that are evaluated for each request that passes through a policy. You can create multiple policies, and they can be associated with an Application Gateway, to individual listeners, or to path-based routing rules on an Application Gateway.
* [Azure Bastion](https://docs.microsoft.com/en-us/azure/bastion/bastion-overview) is a fully managed platform as a service (PaaS) that you provision inside your virtual network. Azure Bastion provides secure and seamless Remote Desktop Protocol (RDP) and secure shell (SSH) connectivity to the VMs in your virtual network, directly from the Azure portal over TLS.
* [Azure Virtual Machines](https://azure.microsoft.com/services/virtual-machines) provides on-demand, scalable computing resources that give you the flexibility of virtualization, without having to buy and maintain the physical hardware.
* [Azure Virtual Network](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview) is the fundamental building block for Azure private networks. With Virtual Network, Azure resources (like VMs) can securely communicate with each other, the internet, and on-premises networks. An Azure Virtual Network is similar to a traditional network that's on premises, but it includes Azure infrastructure benefits, such as scalability, availability, and isolation.
* [Virtual Network Interfaces](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-network-interface) let Azure virtual machines communicate with the internet, Azure, and on-premises resources. You can add several network interface cards to one Azure VM, so that child VMs can have their own dedicated network interface devices and IP addresses.
* [Azure Managed Disks](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/managed-disks-overview) provides block-level storage volumes that Azure manages on Azure VMs. The available types of disks are Ultra disks, Premium solid-state drives (SSDs), Standard SSDs, and Standard hard disk drives (HDDs).
* [Azure Blob Storage](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction) is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.
* [Azure Private Link](https://docs.microsoft.com/en-us/azure/private-link/private-link-overview) enables you to access Azure PaaS services (for example, Azure Blob Storage and Key Vault) and Azure hosted customer-owned/partner services, over a private endpoint in your virtual network.

**Alternatives**

In this architecture, the [Application Gateway Ingress Controller (AGIC)](https://azure.github.io/application-gateway-kubernetes-ingress) was installed using the [AGIC add-on for Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/application-gateway/tutorial-ingress-controller-add-on-new). You can also [install the Application Gateway Ingress Controller via a Helm chart](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-install-existing#multi-cluster--shared-application-gateway). For a new setup, by using one line in Azure CLI, you can deploy a new Application Gateway and a new AKS cluster (with AGIC enabled as an add-on). The add-on is also a fully managed service, which provides added benefits such as automatic updates and increased support. Both ways of deploying AGIC (Helm and the AKS add-on) are fully supported by Microsoft. Additionally, the add-on allows for better integration with AKS, as a first class add-on.

The Application Gateway Ingress Controller (AGIC) add-on is still deployed as a pod in your AKS cluster. There are a few differences between the Helm deployment version and the add-on version of AGIC. The following list includes the differences between the two versions:

* Helm deployment values cannot be modified on the AKS add-on:
  + verbosityLevel will be set to 5 by default
  + usePrivateIp will be set to be false by default; this can be overwritten by the use-private-ip annotation
  + shared is not supported by the add-on
  + reconcilePeriodSeconds is not supported by the add-on
  + armAuth.type is not supported by the add-on
* AGIC deployed via Helm supports ProhibitedTargets, which means AGIC can configure the Application Gateway specifically for AKS clusters, without affecting other existing backends.
* Since the AGIC add-on is a managed service, it is automatically updated to the latest version of the AGIC add-on, unlike AGIC deployed through Helm (where you must manually update AGIC).
* You can only deploy one AGIC add-on per AKS cluster, and each AGIC add-on currently can only target one Application Gateway instance. For deployments that require more than one AGIC per cluster, or multiple AGICs targeting one Application Gateway, you can continue to use AGIC deployed via Helm.

A single instance of the Azure Application Gateway Kubernetes Ingress Controller (AGIC) can ingest events from multiple Kubernetes namespaces. Should the AKS administrator decide to use the Application Gateway as an ingress, all namespaces will use the same instance of Application Gateway. A single installation of Ingress Controller will monitor accessible namespaces and will configure the Application Gateway that it is associated with. For more information, see [Enable multiple Namespace support in an AKS cluster with Application Gateway Ingress Controller](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-multiple-namespace-support).

To enable multi-namespace support, do the following:

* Modify the helm-config.yaml file in one of the following ways:
  + Delete the watchNamespace key entirely from the helm-config.yaml file. AGIC will observe all the namespaces.
  + Set watchNamespace to an empty string. AGIC will observe all namespaces.
  + Add multiple namespaces, separated by a comma (watchNamespace: default,secondNamespace). AGIC will observe these namespaces exclusively.
* Apply Helm template changes with this script: helm install -f helm-config.yaml application-gateway-kubernetes-ingress/ingress-azure

Once deployed with the ability to observe multiple namespaces, AGIC will do the following:

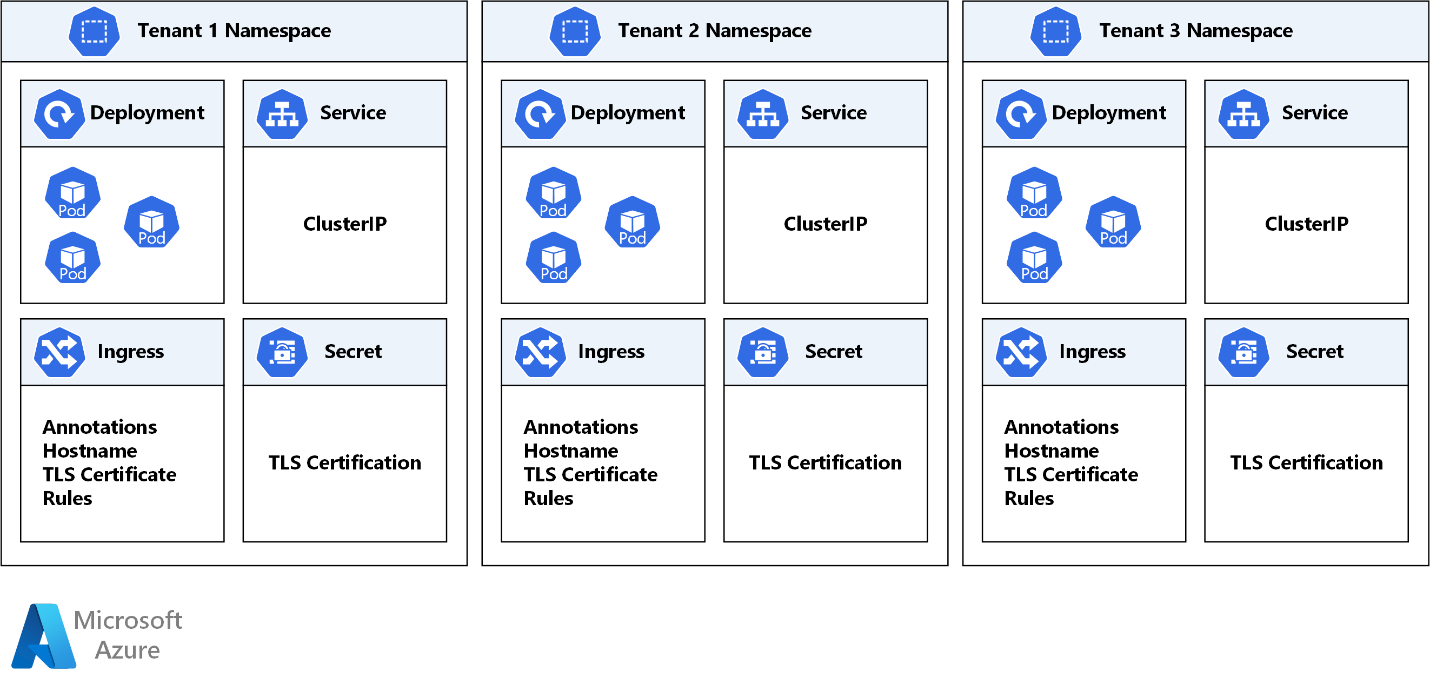
* List ingress resources from all the accessible namespaces
* Filter to ingress resources that are annotated with kubernetes.io/ingress.class: azure/application-gateway
* Compose combined [Application Gateway config](https://github.com/Azure/azure-sdk-for-go/blob/37f3f4162dfce955ef5225ead57216cf8c1b2c70/services/network/mgmt/2016-06-01/network/models.go#L1710-L1744)
* Apply the config to the associated Application Gateway via [ARM](https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview)

## Scenario details

A multitenant Kubernetes cluster is shared by multiple users and workloads that are commonly referred to as "tenants." This definition includes Kubernetes clusters that are shared by different teams or divisions within an organization. It also includes clusters that are shared by per-customer instances of a software-as-a-service (SaaS) application. Cluster multitenancy is an alternative to managing many single-tenant dedicated clusters. The operators of a multitenant Kubernetes cluster must isolate tenants from each other. Isolation minimizes the damage that a compromised or malicious tenant can do to the cluster and to other tenants. When several users or teams share the same cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources. [Resource Quotas](https://kubernetes.io/docs/concepts/policy/resource-quotas) is a tool for administrators to address this concern.

When you plan to build a multitenant Azure Kubernetes Service (AKS) cluster, you should consider the layers of resource isolation that are provided by Kubernetes: cluster, namespace, node, pod, and container. You should also consider the security implications of sharing different types of resources among multiple tenants. For example, scheduling pods from different tenants on the same node could reduce the number of machines needed in the cluster. On the other hand, you might need to prevent certain workloads from being collocated. For example, you might not allow untrusted code from outside of your organization to run on the same node as containers that process sensitive information. [Azure Policy](https://docs.microsoft.com/en-us/azure/aks/policy-reference) can be used to limit the deployment to AKS from only trusted registries.

Although Kubernetes cannot guarantee perfectly secure isolation between tenants, it does offer features that may be sufficient for specific use cases. As a best practice, you should separate each tenant and its Kubernetes resources into their own namespaces. You can then use [Kubernetes RBAC](https://kubernetes.io/docs/reference/access-authn-authz/rbac) and [Network Policies](https://kubernetes.io/docs/concepts/services-networking/network-policies/) to enforce tenant isolation. (RBAC stands for role-based access control.) For example, the following picture shows the typical SaaS Provider Model that hosts multiple instances of the same application on the same cluster, one for each tenant. Each application lives in a separate namespace. When tenants need a higher level of physical isolation and guaranteed performance, their workloads can be deployed to a dedicated set of nodes, dedicated pool, or even a dedicated cluster.



Download a [*Visio file*](https://arch-center.azureedge.net/aks-agic.vsdx) of this architecture.

The [Application Gateway Ingress Controller (AGIC)](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview) is a Kubernetes application, which makes it possible for [Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes) customers to use an [Azure Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) to expose their containerized applications to the Internet. AGIC monitors the Kubernetes cluster that it is hosted on and continuously updates an Application Gateway, so that the selected services are exposed to the Internet. The Ingress Controller runs in its own pod on the customer's AKS instance. AGIC monitors a subset of Kubernetes Resources for changes. The state of the AKS cluster is translated to Application Gateway-specific configuration and applied to the [Azure Resource Manager (ARM)](https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview). This architecture sample shows proven practices to deploy a public or private [Azure Kubernetes Service (AKS) cluster](https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes), with an [Azure Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) and an [Application Gateway Ingress Controller](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview) add-on.

A single instance of the [Azure Application Gateway Kubernetes Ingress Controller (AGIC)](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-multiple-namespace-support) can ingest events from and observe multiple namespaces. All namespaces in the AKS cluster will use the same instance of Application Gateway. A single installation of Ingress Controller will monitor accessible namespaces and will configure the Application Gateway that it is associated with.

### Potential use cases

Use [Application Gateway Ingress Controller (AGIC)](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview) to expose and protect internet-facing workloads that are running on an [Azure Kubernetes Service (AKS) cluster](https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes) in a multitenant environment.

## Considerations

Although the security considerations are not fully pertaining to multitenancy in AKS, we believe they are essential requirements when deploying this solution.

**Multitenancy considerations**

* Design AKS clusters for multitenancy. Kubernetes provides features that let you logically isolate teams and workloads in the same cluster. The goal should be to provide the least number of privileges, scoped to the resources that each team needs. A [Namespace](https://docs.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#namespaces) in Kubernetes creates a logical isolation boundary.
* Use logical isolation to separate teams and projects. The logical separation of clusters usually provides a higher pod density than physically isolated clusters.
* Use dedicated node pools, or dedicated AKS clusters, whenever you need to implement a strict physical isolation. For example, you can dedicate a pool of worker nodes or an entire cluster, to a team or a tenant in a multitenant environment. You can use a combination of [taints and tolerations](https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/) to control the deployment of pods to a specific node pool. Please note that a node in AKS can be tainted only at the time of node pool creation. Alternately, [labels and nodePool selectors](https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/) can be used to control the deployment of workload to specific node pools.

#### Network security

* Create a [private endpoint](https://azure.microsoft.com/services/private-link) for any PaaS service that is used by AKS workloads, such as Key Vault, Service Bus, or Azure SQL Database. This is so that the traffic between the applications and these services isn't exposed to the public internet. For more information, see [What is Azure Private Link](https://docs.microsoft.com/en-us/azure/private-link/private-link-overview).
* Configure your Kubernetes Ingress resource to expose workloads via HTTPS, and use a separate subdomain and digital certificate for each tenant. The [Application Gateway Ingress Controller (AGIC)](https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview) will automatically configure the [Azure Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) listener for secure socket layer (SSL) termination.
* Configure [Azure Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) to use a [Web Application Firewall Policy](https://docs.microsoft.com/en-us/azure/application-gateway/waf-overview) to protect public-facing workloads (that are running on AKS) from malicious attacks.
* 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.
* Use network policies to segregate and secure intra-service communications by controlling which components can communicate with each other. By default, all pods in a Kubernetes cluster can send and receive traffic without limitations. To improve security, you can use Azure Network Policies or Calico Network Policies to define rules that control the traffic flow between different microservices. For more information, see [Network Policy](https://docs.microsoft.com/en-us/azure/aks/use-network-policies).
* 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.
* Consider using a [private AKS cluster](https://docs.microsoft.com/en-us/azure/aks/private-clusters) in your production environment, or at least secure access to the API server, by using [authorized IP address ranges](https://docs.microsoft.com/en-us/azure/aks/api-server-authorized-ip-ranges) in Azure Kubernetes Service.

#### Authentication and authorization

* Deploy AKS clusters with Azure AD integration. For more information, see [AKS-managed Azure Active Directory integration](https://docs.microsoft.com/en-us/azure/aks/managed-aad). Using Azure AD centralizes the identity management component. Any change in user account or group status is automatically updated in access to the AKS cluster. Use [Roles](https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole) or [ClusterRoles](https://kubernetes.io/docs/reference/access-authn-authz/rbac/" \l "role-and-clusterrole) and [Bindings](https://kubernetes.io/docs/reference/access-authn-authz/rbac/#rolebinding-and-clusterrolebinding) to scope users or groups to the least number of permissions needed.
* Use Kubernetes RBAC to define the permissions that users or groups have to resources in the cluster. Create roles and bindings that assign the least number of permissions required. [Integrate Kubernetes RBAC with Azure AD](https://docs.microsoft.com/en-us/azure/aks/azure-ad-rbac) so any change in user status or group membership is automatically updated and access to cluster resources is current.
* Use Azure RBAC to define the minimum required permissions that users or groups have to AKS resources in one or more subscriptions. For more information, see [Kubernetes RBAC](https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-identity#use-kubernetes-role-based-access-control-kubernetes-rbac) and [Use Azure RBAC for Kubernetes authorization](https://docs.microsoft.com/en-us/azure/aks/manage-azure-rbac).
* Consider using [AAD Pod Identity](https://docs.microsoft.com/en-us/azure/aks/use-azure-ad-pod-identity) to assign a managed identity for an Azure resource to individual microservices, which they can then use to access managed resources (such as Azure Key Vault, SQL Database, Service Bus, and Cosmos DB), without the need to store and retrieve use connection strings or credentials from Kubernetes secrets.
* Consider using the [Secret Store CSI Driver for Azure Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/general/key-vault-integrate-kubernetes) to access secrets, such as credentials and connections strings from Key Vault, rather than from Kubernetes secrets.
* Consider using the [Dapr Secrets Stores](https://docs.dapr.io/developing-applications/building-blocks/secrets/secrets-overview/) building block, with the [Azure Key Vault store with Managed Identities on Kubernetes](https://docs.dapr.io/developing-applications/integrations/azure/authenticating-azure/), to retrieve secrets (such as credentials and connection strings) from Key Vault.

#### Workload and cluster

* Securing access to the Kubernetes API-Server is one of the most important things you can do to secure your cluster. Integrate Kubernetes role-based access control (Kubernetes RBAC) with Azure Active Directory to control access to the API server. These controls let you secure AKS the same way that you secure access to your Azure subscriptions.
* Limit access to actions that containers can perform. Use [Pod Security Policy](https://kubernetes.io/docs/concepts/policy/pod-security-policy/) to enable the fine-grained authorization of pod creation and updates. Provide the least number of permissions, and avoid the use of root / privileged escalation. For more best practices, see [Secure pod access to resources](https://docs.microsoft.com/en-us/azure/aks/developer-best-practices-pod-security#secure-pod-access-to-resources).
* Whenever possible, avoid running containers as a root user.
* Use the [AppArmor](https://kubernetes.io/docs/tutorials/clusters/apparmor) Linux kernel security module to limit the actions that containers can do.
* Regularly upgrade your AKS clusters to the latest Kubernetes version to take advantage of new features and bug fixes.
* AKS automatically downloads and installs security fixes on each Linux node, but it doesn't automatically reboot the node if necessary. Use [kured](https://github.com/weaveworks/kured) to watch for pending reboots, cordon and drain nodes, and finally, apply your updates. For Windows Server nodes, regularly run an AKS upgrade operation to safely cordon and drain pods and to deploy any updated nodes.
* Consider using HTTPS and gRPC secure transport protocols for all intra-pod communications and to use a more advanced authentication mechanism that does not require you to send the plain credentials on every request, like OAuth or JWT. Secure intra-service communication can be achieved by leveraging a service mesh, like [Istio](https://istio.io/), [Linkerd](https://linkerd.io/), [Consul](https://www.consul.io/), or [Open Service Mesh](https://openservicemesh.io/), or by using [Dapr](https://docs.dapr.io/developing-applications/building-blocks/service-invocation/service-invocation-overview).

#### Azure Container Registry

* Scan your container images for vulnerabilities, and only deploy images that have passed validation. Regularly update the base images and application runtime, then redeploy workloads in the AKS cluster. Your deployment workflow should include a process to scan container images, by using tools such as [Prisma Cloud](https://docs.prismacloudcompute.com/docs) or [Aqua](https://www.aquasec.com/), and then you can only allow verified images to be deployed.
* As you use base images for application images, use automation to build new images when the base image is updated. Because those base images typically include security fixes, update any downstream application container images. For more information about base image updates, see [Automate image builds on base image update with Azure Container Registry Tasks](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-tutorial-base-image-update).

### Performance considerations

Although the performance considerations are not fully pertaining to multitenancy in Azure Kubernetes Service (AKS), we believe they are essential requirements when deploying this solution:

* For low latency workloads, consider deploying a dedicated node pool in a proximity placement group. 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 that's used to make sure Azure compute resources are physically located close to each other. Some use cases (such as gaming, engineering simulations, and high-frequency trading (HFT)) require low latency and tasks that complete quickly. For high-performance computing (HPC) scenarios such as these, consider using [proximity placement groups](https://docs.microsoft.com/en-us/azure/virtual-machines/co-location#proximity-placement-groups) (PPG) for your cluster's node pools.
* Always use smaller container images, as it helps you to create faster builds. Smaller images are also less vulnerable to attack vectors, because of a reduced attack surface.
* Use Kubernetes autoscaling to dynamically scale out the number of worker nodes of an AKS cluster when the traffic increases. With [Horizontal Pod Autoscaler](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale) and a cluster autoscaler, node and pod volumes get adjusted dynamically in real-time, to match the traffic condition and to avoid downtimes that are caused by capacity issues. For more information, see [Automatically scale a cluster to meet application demands on Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/aks/cluster-autoscaler).

### Availability and reliability considerations

Although the availability and reliability considerations are not fully pertaining to multitenancy in Azure Kubernetes Service (AKS), we believe they are essential requirements when deploying this solution. Consider the following ways to optimize availability for your AKS cluster and workloads.

#### Containers

* Use Kubernetes health probes to check that your containers are alive and healthy:
  + The [livenessProbe](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/" \l "define-a-liveness-command) indicates whether the container is running. If the liveness probe fails, the [kubelet](https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet) kills the container, and the container is subjected to its restart policy.
  + The [readinessProbe](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/" \l "define-readiness-probes) indicates whether the container is ready to respond to requests. If the readiness probe fails, the endpoints controller removes the pod's IP address from the endpoints of all services that match the pod. The default state of readiness before the initial delay is Failure.
  + The [startup probe](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/#define-startup-probes) indicates whether the application within the container is started. All other probes are disabled if a startup probe is provided, until it succeeds. If the startup probe fails, the [kubelet](https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet) kills the container, and the container is subjected to its restart policy.
* Resource contention can affect service availability. Define container resource constraints so that no single container can overwhelm the cluster memory and CPU resources. You can use AKS diagnostics to identify any issues in the cluster.
* Use the resource limit to restrict the total resources allowed for a container, so one particular container can't starve others.

#### Container registry

* We suggest storing container images in Azure Container Registry, and then geo-replicate the registry to each AKS region using [Azure Container Registry geo-replication](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-geo-replication). Geo-replication is a feature of Premium SKU ACR registries.
* Scan your container images for vulnerabilities, and only deploy images that have passed validation. Regularly update the base images and application runtime, and then redeploy your workloads in the AKS cluster. [Microsoft Defender for Containers](https://docs.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-introduction) can scan container images in Azure Container Registry.
* Limit the image registries that pods and deployments can use. Only allow trusted registries, where you validate and control the images that are available.
* As you use base images for application images, use automation to build new images, when the base image is updated. Because those base images typically include security fixes, update any downstream application container images. We recommend that you scan the container images for vulnerabilities as part of CI/CD pipeline before you publish the images to container registry. [Microsoft Defender for Container can be integrated to CI/CD workflows.](https://docs.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-cicd)
* Leverage [ACR Tasks](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-tasks-overview) in Azure Container Registry to automate OS and framework patching for your Docker containers. ACR Tasks supports automated build execution, when a container's base image is updated, such as when you patch the OS or application framework in one of your base images.

#### Intra-region resiliency

* Consider deploying Azure Application Gateway with availability zones within a region.
* Consider deploying the node pools of your AKS cluster, across all the [Availability Zones](https://docs.microsoft.com/en-us/azure/aks/availability-zones) within a region, and use an [Azure Standard Load Balancer](https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview) or [Azure Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) in front of your node pools. This topology provides better resiliency, in case of an outage of a single datacenter. This way, cluster nodes are distributed across multiple datacenters, in three separate Availability Zones within a region.
* Enable [zone redundancy in Azure Container Registry](https://docs.microsoft.com/en-us/azure/container-registry/zone-redundancy), for intra-region resiliency and high availability.
* Use [Pod Topology Spread Constraints](https://kubernetes.io/docs/concepts/workloads/pods/pod-topology-spread-constraints/) to control how pods are spread across your AKS cluster among failure-domains, such as regions, availability zones, and nodes.
* Consider using Uptime SLA for AKS clusters that host mission-critical workloads. [Uptime SLA](https://docs.microsoft.com/en-us/azure/aks/uptime-sla) is an optional feature to enable a financially backed, higher SLA for a cluster. Uptime SLA guarantees 99.95% availability of the Kubernetes API server endpoint, for clusters that use Availability Zones. And it guarantees 99.9% availability for clusters that don't use Availability Zones. AKS uses master node replicas across update and fault domains, in order to ensure the SLA requirements are met.

#### Disaster recovery and business continuity

* Consider deploying your solution to at least [two paired Azure regions](https://docs.microsoft.com/en-us/azure/best-practices-availability-paired-regions) within a geography. You should also adopt a global load balancer, such as [Azure Traffic Manager](https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview) or [Azure Front Door](https://docs.microsoft.com/en-us/azure/frontdoor/front-door-overview), with an active/active or active/passive routing method, in order to guarantee business continuity and disaster recovery.
* Make sure to script, document, and periodically test any regional failover process in a QA environment, to avoid unpredictable issues if a core service is affected by an outage in the primary region.
* These tests are also meant to validate if the DR approach meets the RPO/RTO targets, in conjunction to eventual manual processes and interventions that are needed for a failover.
* Make sure you test fail-back procedures, to understand if they work as expected.
* Store your container images in [Azure Container Registry](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-intro), and geo-replicate the registry to each AKS region. For more information, see [Geo-replication in Azure Container Registry](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-geo-replication).
* Where possible, don't store service state inside the container. Instead, use an Azure platform as a service (PaaS) that supports multi-region replication.
* If you use Azure Storage, prepare and test how to migrate your storage from the primary region to the backup region.
* [Consider deploying the cluster configuration using GitOps](https://docs.microsoft.com/en-us/azure/architecture/example-scenario/gitops-aks/gitops-blueprint-aks). Using GitOps provides uniformity between primary and DR clusters and a quick way to rebuild new cluster in case of cluster loss.
* Consider backup / restore of the cluster configuration using tools such as [Velero](https://github.com/vmware-tanzu/velero).

### Storage considerations

Although the storage considerations are not fully pertaining to multitenancy in Azure Kubernetes Service (AKS), we believe they are essential requirements when deploying this solution:

* Consider deploying your AKS cluster with [ephemeral OS disks](https://docs.microsoft.com/en-us/azure/aks/cluster-configuration#ephemeral-os) that provide lower read/write latency, along with faster node scaling and cluster upgrades.
* Understand the needs of your application to pick the right storage. Use high performance, SSD-backed storage for production workloads. Plan for a network-based storage system, such as [Azure Files](https://docs.microsoft.com/en-us/azure/storage/files/storage-files-introduction), when multiple pods need to concurrently access the same files. For more information, see [Storage options for applications in Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/aks/concepts-storage).
* 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.
* 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.

**Scheduler considerations**

Although some of the scheduler considerations are not fully pertaining to multitenancy in Azure Kubernetes Service (AKS), we believe they are essential requirements when deploying this solution:

* Make sure you review and implement the [best practices](https://docs.microsoft.com/en-us/azure/aks/best-practices), for cluster operators and application developers to build and run applications successfully on Azure Kubernetes Service (AKS).
* Plan and apply [resource quotas](https://kubernetes.io/docs/concepts/policy/resource-quotas) at the namespace level, for all namespaces. If pods don't define resource requests and limits, then reject the deployment. Monitor resource usage, and then adjust quotas as needed. When several teams or tenants share an AKS cluster with a fixed number of nodes, resource quotas can be used to assign a fair share of resources to each team or tenant.
* Adopt [Limit Ranges](https://kubernetes.io/docs/concepts/policy/limit-range/) to constrain resource allocations (to pods or containers) in a namespace, in terms of CPU and memory.
* Explicitly define resource [requests and limits](https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/) for CPU and memory usage, for your pods in the YAML manifests or Helm charts that you use to deploy applications. When you specify the resource request for containers in a pod, the Kubernetes scheduler uses this information to decide which node to place the pod on. When you specify a resource limit (such as the CPU or memory) for a container, the kubelet enforces those limits so that the running container can't use more of that resource than the limit you set.
* To maintain the availability of applications, define [Pod Disruption Budgets](https://kubernetes.io/docs/tasks/run-application/configure-pdb), to make sure that a minimum number of pods are available in the cluster.
* [Priority classes](https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption) can be used to indicate the importance of a pod. If a pod cannot be scheduled, the scheduler tries to preempt (evict) lower priority pods, in order to make scheduling of the pending pod possible.
* Use Kubernetes [taints and tolerations](https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/) to place resource-intensive applications on specific nodes, and to avoid noisy neighbor issues. Keep node resources available for workloads that require them, and don't allow other workloads to be scheduled on the nodes.
* Control the scheduling of pods on nodes, by using node selectors, node affinity, or inter-pod affinity. Use inter-pod [affinity and anti-affinity](https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity) settings to colocate pods that have chatty communications, to place pods on different nodes, and to avoid running multiple instances of the same pod kind on the same node.

**Service mesh considerations**

Although the service mesh considerations are not fully pertaining to multitenancy in AKS, we believe they are essential requirements when deploying this solution:

* Consider using an open-source service mesh (like [Istio](https://istio.io/), [Linkerd](https://linkerd.io/), [Consul](https://www.consul.io/), or [Open Service Mesh](https://openservicemesh.io/)) in your AKS cluster, in order to improve the observability, reliability, and security for your microservices, via mutual TLS.
* You can also implement traffic-splitting strategies (such blue/green deployments and canary deployments). In short, a service mesh is a dedicated infrastructure layer for making service-to-service communication safe, fast, and reliable. For more information, see:
  + [Open Service Mesh AKS add-on](https://docs.microsoft.com/en-us/azure/aks/open-service-mesh-about)
* Consider adopting [Dapr](https://dapr.io/) to build resilient, microservice stateless and stateful applications. You can use any programming language and developer framework.

**DevOps considerations**

* Deploy your workloads to Azure Kubernetes Service (AKS), with a [Helm](https://helm.sh/) chart in a CI/CD pipeline, by using a DevOps system, such as [GitHub Actions](https://docs.github.com/en/actions) or [Azure DevOps](https://azure.microsoft.com/services/devops). For more information, see [Build and deploy to Azure Kubernetes Service](https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/kubernetes/aks-template?view=azure-devops).
* [Deploy the cluster configuration using GitOps](https://docs.microsoft.com/en-us/azure/azure-arc/kubernetes/tutorial-use-gitops-flux2), by using GitHub.
* Introduce A/B testing and canary deployments in your application lifecycle management, to properly test an application before making it available for all users. There are several techniques that you can use to split the traffic across different versions of the same service.
* As an alternative, you can use the traffic-splitting capabilities that are provided by a service mesh implementation. For more information, see:
  + [Traffic Splitting | Open Service Mesh](https://release-v1-0.docs.openservicemesh.io/docs/guides/traffic_management/traffic_split/)
* Use Azure Container Registry or another container registry (like Docker Hub), to store the private Docker images that are deployed to the cluster. AKS can authenticate with Azure Container Registry, by using its Azure AD identity.

**Monitoring considerations**

Although the monitoring considerations are not fully pertaining to multitenancy in Azure Kubernetes Service (AKS), we believe they are essential requirements when deploying this solution:

* Use [Container insights](https://docs.microsoft.com/en-us/azure/azure-monitor/containers/container-insights-overview) to monitor the health status of the AKS cluster and workloads.
* Configure all the PaaS services (such as Azure Container Registry and Key Vault) to collect diagnostics logs and metrics, to [Azure Monitor Log Analytics](https://docs.microsoft.com/en-us/azure/azure-monitor/logs/log-analytics-overview).

**Cost optimization**

The cost of this architecture depends on configuration aspects, like the following:

* Service tiers
* Scalability, meaning the number of instances that are dynamically allocated by services to support a given demand
* Automation scripts
* Your disaster recovery level

After you assess these aspects, go to the [Azure pricing calculator](https://azure.microsoft.com/pricing/calculator) to estimate your costs. Also, for more pricing optimization options, see the [Principles of cost optimization](https://docs.microsoft.com/en-us/azure/architecture/framework/cost/overview) in the Microsoft Azure Well-Architected Framework.

## Deploy this scenario

The source code for this scenario is available [on GitHub](https://github.com/Azure-Samples/aks-agic). This solution is open source and provided with an [MIT License](https://github.com/Azure-Samples/aks-agic/blob/main/LICENSE.md). You can also find a demo application, as shown in the following figure, in [this GitHub repository](https://github.com/Azure-Samples/aks-multi-tenant-agic).

Download a [*Visio file*](https://arch-center.azureedge.net/aks-agic.vsdx) of this architecture.

### Prerequisites

For online deployments, you must have an existing Azure account. If you need one, create a [free Azure account](https://azure.microsoft.com/free/?WT.mc_id=A261C142F) before you begin.

### Deployment to Azure

1. Make sure you have your Azure subscription information handy.
2. Start by cloning the [workbench GitHub repository](https://github.com/Azure-Samples/aks-agic):

gitCopy

git clone https://github.com/Azure-Samples/aks-agic.git

1. Follow the instructions provided in the [README.md file](https://github.com/Azure-Samples/aks-agic/blob/main/README.md).

## Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

* [Paolo Salvatori](https://it.linkedin.com/in/paolo-salvatori) | Principal Service Engineer