**Microservices architecture on Azure Kubernetes Service (AKS)**

Azure

Azure DevOps

Files

Monitor

Pipelines

Azure role-based access control

Sentinel

Storage

GitHub

Visual Studio

Windows

This reference architecture shows a microservices application deployed to Azure Kubernetes Service (AKS). It describes a basic AKS configuration that can be the starting point for most deployments. This article assumes basic knowledge of Kubernetes. The article focuses mainly on the infrastructure and DevOps considerations of running a microservices architecture on AKS. For guidance on how to design microservices, see [Building microservices on Azure](https://docs.microsoft.com/en-us/azure/architecture/microservices/).

 A reference implementation of this architecture is available on [GitHub](https://github.com/mspnp/microservices-reference-implementation).

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

**Architecture**

The architecture consists of the following components.

**Azure Kubernetes Service** (AKS). AKS is an Azure service that deploys a managed Kubernetes cluster.

**Kubernetes cluster**. AKS is responsible for deploying the Kubernetes cluster and for managing the Kubernetes API server. You only manage the agent nodes.

**Virtual network**. By default, AKS creates a virtual network to deploy the agent nodes into. For more advanced scenarios, you can create the virtual network first, which lets you control things like how the subnets are configured, on-premises connectivity, and IP addressing. For more information, see [Configure advanced networking in Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/aks/configure-advanced-networking).

**Ingress**. An ingress exposes HTTP(S) routes to services inside the cluster. For more information, see the section [API Gateway](https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/containers/aks-microservices/aks-microservices#api-gateway) below.

**Azure Load Balancer**. An Azure Load Balancer is created when the NGINX ingress controller is deployed. The load balancer routes internet traffic to the ingress.

**External data stores**. Microservices are typically stateless and write state to external data stores, such as Azure SQL Database or Cosmos DB.

**Azure Active Directory**. AKS uses an Azure Active Directory (Azure AD) identity to create and manage other Azure resources such as Azure load balancers. Azure AD is also recommended for user authentication in client applications.

**Azure Container Registry**. Use Container Registry to store private Docker images, which are deployed to the cluster. AKS can authenticate with Container Registry using its Azure AD identity. Note that AKS does not require Azure Container Registry. You can use other container registries, such as Docker Hub.

**Azure Pipelines**. Pipelines is part of Azure DevOps Services and runs automated builds, tests, and deployments. You can also use third-party CI/CD solutions such as Jenkins.

**Helm**. Helm is as a package manager for Kubernetes — a way to bundle Kubernetes objects into a single unit that you can publish, deploy, version, and update.

**Azure Monitor**. Azure Monitor collects and stores metrics and logs, including platform metrics for the Azure services in the solution and application telemetry. Use this data to monitor the application, set up alerts and dashboards, and perform root cause analysis of failures. Azure Monitor integrates with AKS to collect metrics from controllers, nodes, and containers, as well as container and node logs.

**Design considerations**

This reference architecture is focused on microservices architectures, although many of the recommended practices will apply to other workloads running on AKS.

**Microservices**

A microservice is a loosely coupled, independently deployable unit of code. Microservices typically communicate through well-defined APIs and are discoverable through some form of service discovery. The service should always be reachable even when the pods move around. The Kubernetes **Service** object is a natural way to model microservices in Kubernetes.

**API gateway**

API gateways are a general [microservices design pattern](https://microservices.io/patterns/apigateway.html). An *API gateway* sits between external clients and the microservices. It acts as a reverse proxy, routing requests from clients to microservices. It may also perform various cross-cutting tasks such as authentication, SSL termination, and rate limiting. For more information, see

* [Using API gateways in microservices](https://docs.microsoft.com/en-us/azure/architecture/microservices/design/gateway)
* [Choosing a gateway technology](https://docs.microsoft.com/en-us/azure/architecture/microservices/design/gateway#choosing-a-gateway-technology)

In Kubernetes, the functionality of an API gateway is mostly handled by the **Ingress** resource and the **Ingress controller**. The considerations are described in the [Ingress](https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/containers/aks-microservices/aks-microservices#ingress) section.

**Data storage**

In a microservices architecture, services should not share data storage. Each service should own its own private data in a separate logical storage, to avoid hidden dependencies among services. The reason is to avoid unintentional coupling between services, which can happen when services share the same underlying data schemas. Also, when services manage their own data stores, they can use the right data store for their particular requirements. For more information, see [Designing microservices: Data considerations](https://docs.microsoft.com/en-us/azure/architecture/microservices/design/data-considerations).

Avoid storing persistent data in local cluster storage, because that ties the data to the node. Instead,

* Use an external service such as Azure SQL Database or Cosmos DB, *or*
* Mount a persistent volume using Azure Disks or Azure Files. Use Azure Files if the same volume needs to be shared by multiple pods.

**Service object**

The Kubernetes **Service** object provides a set of capabilities that match the microservices requirements for service discoverability:

* IP address. The Service object provides a static internal IP address for a group of pods (ReplicaSet). As pods are created or moved around, the service is always reachable at this internal IP address.
* Load balancing. Traffic sent to the service's IP address is load balanced to the pods.
* Service discovery. Services are assigned internal DNS entries by the Kubernetes DNS service. That means the API gateway can call a backend service using the DNS name. The same mechanism can be used for service-to-service communication. The DNS entries are organized by namespace, so if your namespaces correspond to bounded contexts, then the DNS name for a service will map naturally to the application domain.

The following diagram shows the conceptual relation between services and pods. The actual mapping to endpoint IP addresses and ports is done by kube-proxy, the Kubernetes network proxy.