

# AKS DevOps CaseStudy2

Last updated by | Tony Wecker | Mar 17, 2021 at 4:36 PM GMT+1

---

## AKS DevOps Case Study 2

### Contents

- [AKS DevOps Case Study 2](#)
  - [Challenge](#)
  - [Microsoft Technologies](#)
  - [Target App-Infrastructure](#)
  - [Secrets Management](#)
  - [Identity and Access Management](#)
  - [Deployment of Azure resources](#)
  - [Container Build and Deployment Process](#)
  - [Staging](#)
  - [Conclusion](#)


### Challenge

Our customer wants to bring its services, which it previously operated in its own data center infrastructure, up to the current state of the art and fully automate them along the process. Due to strict governance and compliance requirements, these services are to be operated in a hybrid cloud infrastructure. The objective is to leverage the advantages of Azure Cloud and Azure Kubernetes Services in terms of automation and manageability while keeping the scalability options open.

### Microsoft Technologies

- Azure Kubernetes Services
- SQL Database Server
- Active Directory
- Key Vault
- Storage Accounts
- Virtual Network
- LogAnalytics

### Target App-Infrastructure

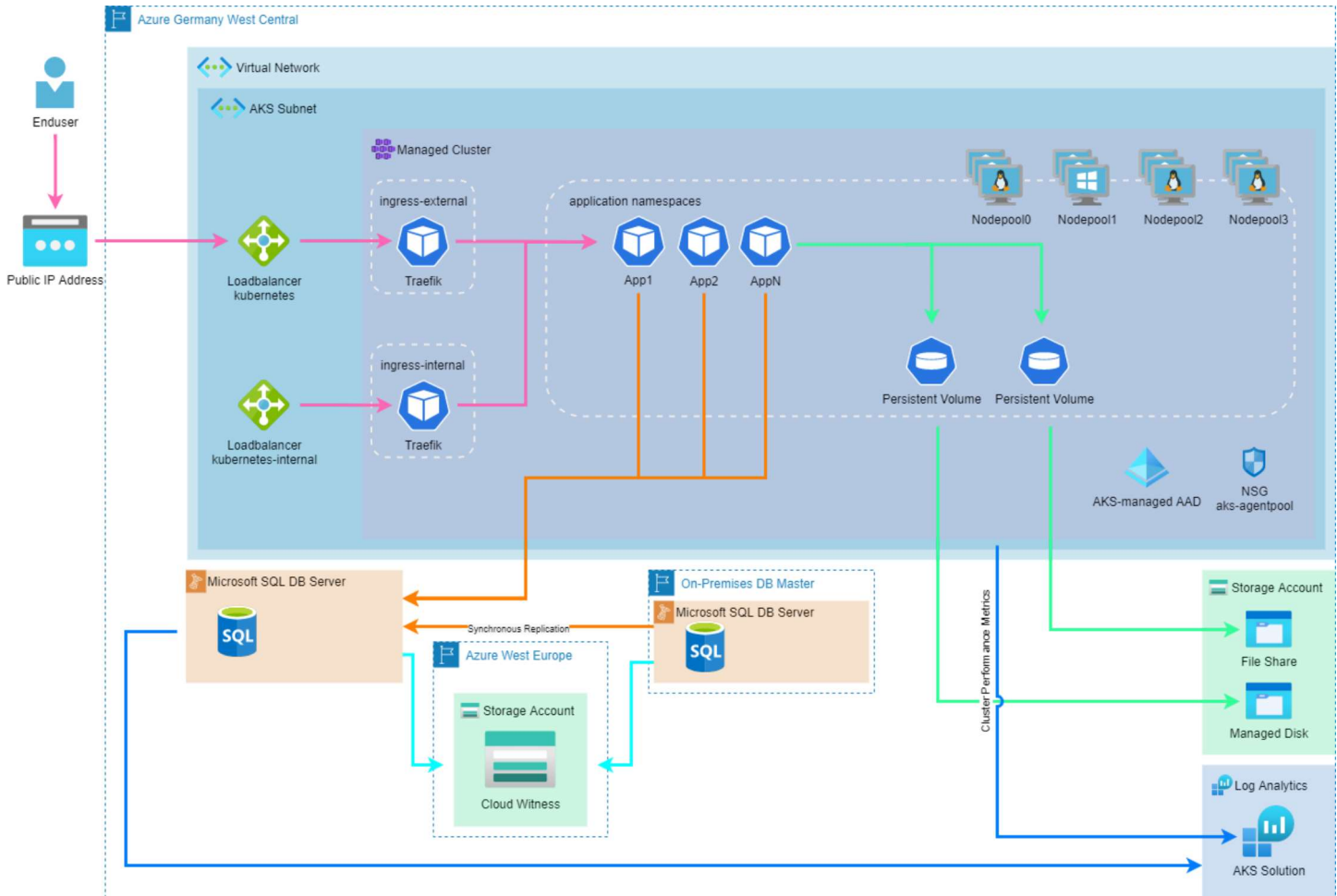
The fully containerized application is operated in a Kubernetes Services Cluster (AKS). An Azure Layer-4 load balancer forms the entry gate, which distributes incoming traffic to the customer's Kubernetes IngressController. At the customer's request, the cloud-native IngressController [Traefik](#)  is used. The traffic is secured via network security groups, to which the nodes of the cluster have been assigned.

The applications addressed with it are distributed on different NodePools, since there are different requirements regarding the performance and size of the associated nodes. In addition, the application is based on .NET and therefore uses a Windows node pool.

A group of SQL database servers is used as the database, part of which is operated on-premise in a data center in Frankfurt am Main. Due to low latencies to the Azure Region Germany West Central, synchronous replication of the database is possible. In order to be able to make correct failover decisions in the event of a data center failure, a cloud witness file is used as a quorum, which is stored within a storage account in a third Azure region.

Other data that needs to be stored persistently is stored in persistent volumes either as a file share or as a managed disk.

To extend the existing monitoring, cluster metrics are written into a log-analytics workspace.



## Secrets Management

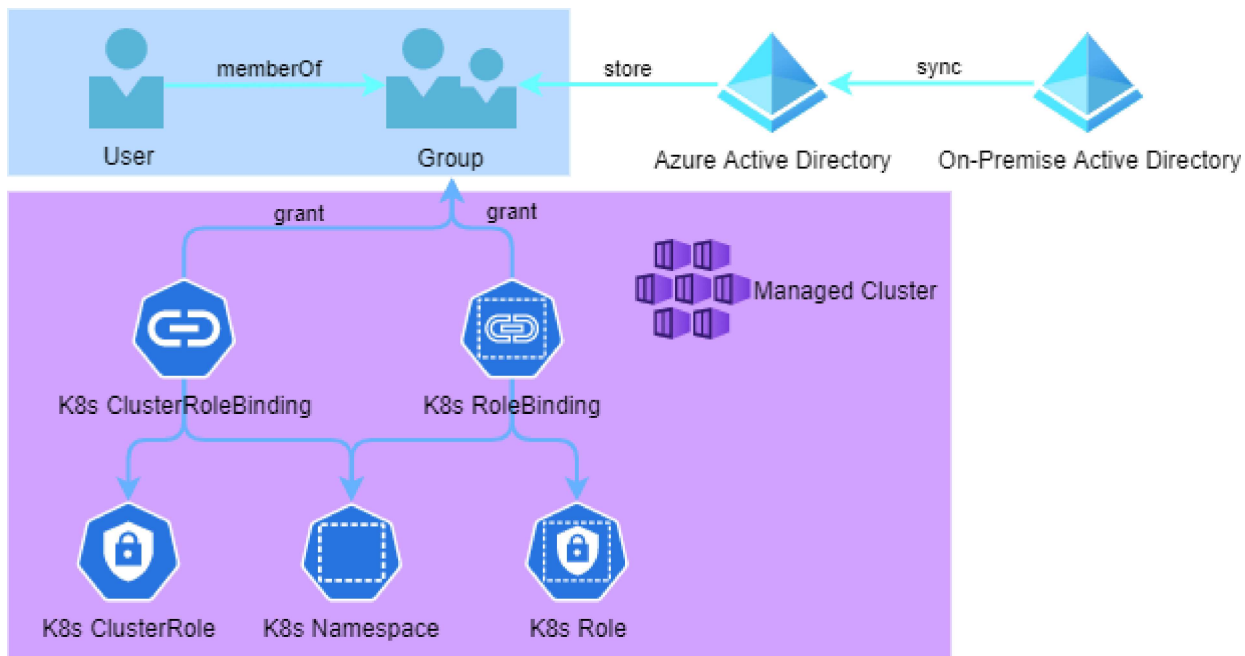
Azure Key Vault is used to manage secrets and certificates, which can be accessed from AKS as well as from the customer's build and deployment environment.

## Identity and Access Management



Azure Active Directory (AAD) is used for Identity and Access Management. The customer already has a local Active Directory in place for which Azure AD Connect is set up. The required entities are filtered and synchronized to Azure AD.

The users and groups managed in AAD should be able to log in to the AKS cluster with their accounts. In addition, a granular permission concept has been defined, which is implemented with Azure RBAC and Kubernetes RBAC.

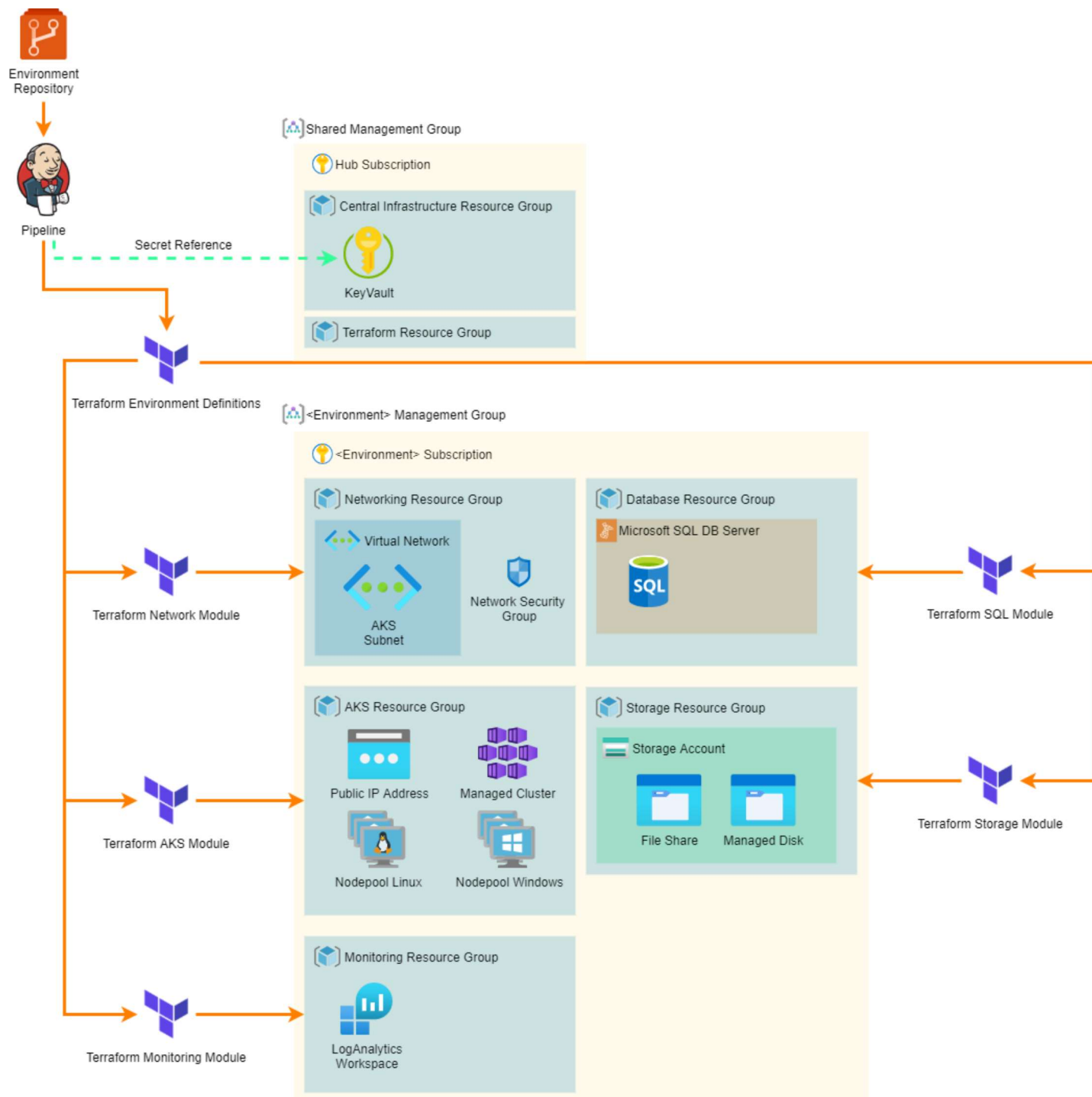
As a prerequisite for this, the AKS-managed Azure Active Directory integration feature is used in all AKS clusters and Kubernetes Roles and RoleBindings exist that reference the respective AAD groups.



## Deployment of Azure resources

The deployment and configuration of all Azure resources is automated via [Terraform](#)  using the official HashiCorp Azure Terraform provider. The Terraform definitions are managed in a Git repository of the customer and the Terraform state file, which stores the state of the managed infrastructure, is stored using a Storage Account within the hub subscription. Each environment has its own Git repository, and reusable modules are used for central components. The execution of Terraform is also automated via a [Jenkins](#)  pipeline within the customer's Jenkins.

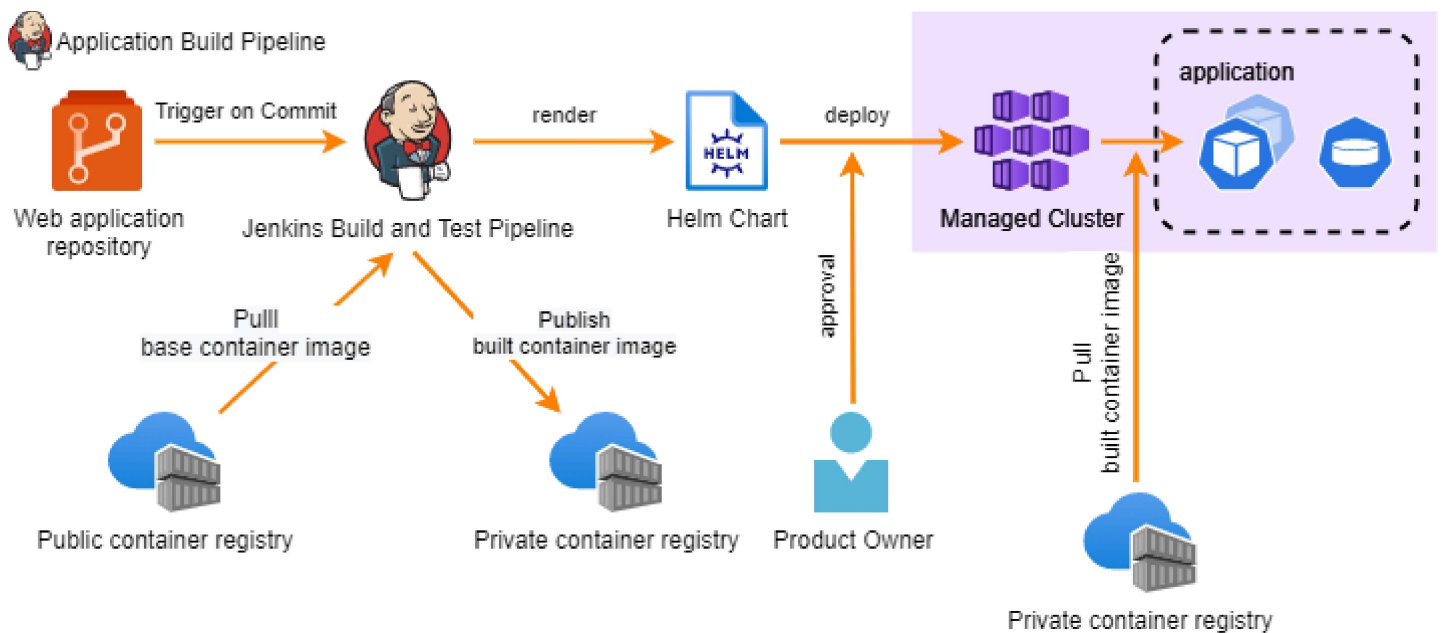
The load balancer configuration is done automatically by the Kubernetes Cloud Controller through an AKS managed identity, which uses the Kubernetes objects as a reference.



## Container Build and Deployment Process

Deployment of all Kubernetes Cluster internal resources is done from the customer's CI/CD environment. Separate pipelines are used for applications and infrastructure. New commits in the customer's repositories automatically initiate the pipeline processes and the associated customer-specific build, test and acceptance process.

The customer is also responsible for building the application containers. The resulting artifacts are stored as container images in an Azure Container Registry, to which the AKS clusters also have access. Kubernetes objects are managed and packaged using Helm. Helm is also used as a deployment tool for Kubernetes resources.



## Staging

The customer applications go through different development cycles, for which different clusters are provided. There is a development cluster for development purposes, a stage cluster for acceptance tests, and finally the production system.

## Conclusion

The environment outlined here covers all the customer's requirements in terms of compliance, scalability and modern system architecture. In addition, the high level of automation in the provisioning of resources enables a reduction in the potential for errors caused by manual configuration.