AKS DevOps CaseStudy1

Last updated by | Fabian Flanhardt | Mar 18, 2021 at 3:06 PM GMT+1

AKS DevOps Case Study 1

Contents

- AKS DevOps Case Study 1
  - Challenge
  - Microsoft Technologies
  - Target App-Infrastructure
  - Secrets Management
  - Deployment of Azure resources
  - Configuration of Kubernetes Cluster
  - Container Build and Deployment Process
  - Staging
  - Conclusion

Challenge

Our customer has a containerizes application and is looking for a reliable and scalable solution to deploy it. To keep administration tasks to a minimum the customer wants to make use of PaaS Resources in a public cloud environment, and an automated CI/CD process.

The main system consists of multiple containerized services including a web application which stores data in a MySQL database as well as assets in a volume. A Redis container is used to minimize the database load by caching requests. The database is frequently backed-up by dumping all schemas and data into a file. Several other containers perform processes like database migrations and other maintenance tasks.

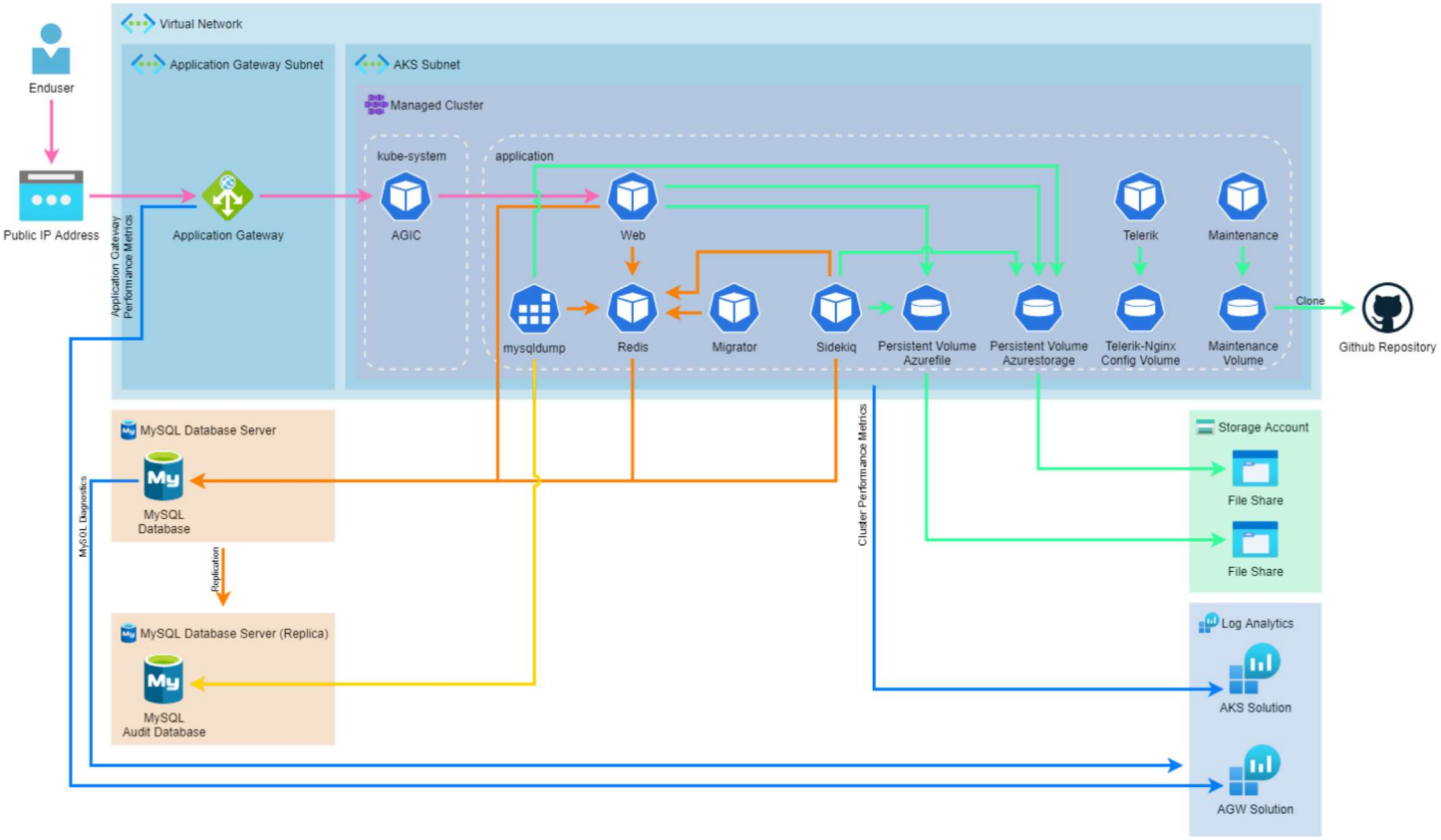
Microsoft Technologies

- Azure Kubernetes Services
- Azure Database for MySQL
- Storage Accounts
- Application Gateway
- Virtual Network
- LogAnalytics
- DevOps Services

Target App-Infrastructure

As solution for the described requirements a managed Azure Kubernetes Services (=AKS) cluster comes to use which can run all containers with minimum changes required to the application. A secured ingress is handled by an Azure Application Gateway. As for a database solution the MySQL database is deployed as a managed PaaS database in Microsoft Azure. To reduce load on the primary database, a replica database is deployed for all reporting and backup purposes. Azure Fileshares are mounted in the AKS cluster as

persistent volumes to provide necessary storage for assets and database-dumps. The AKS cluster, MySQL database and Application Gateway is monitored using an Azure LogAnalytics Workspace.



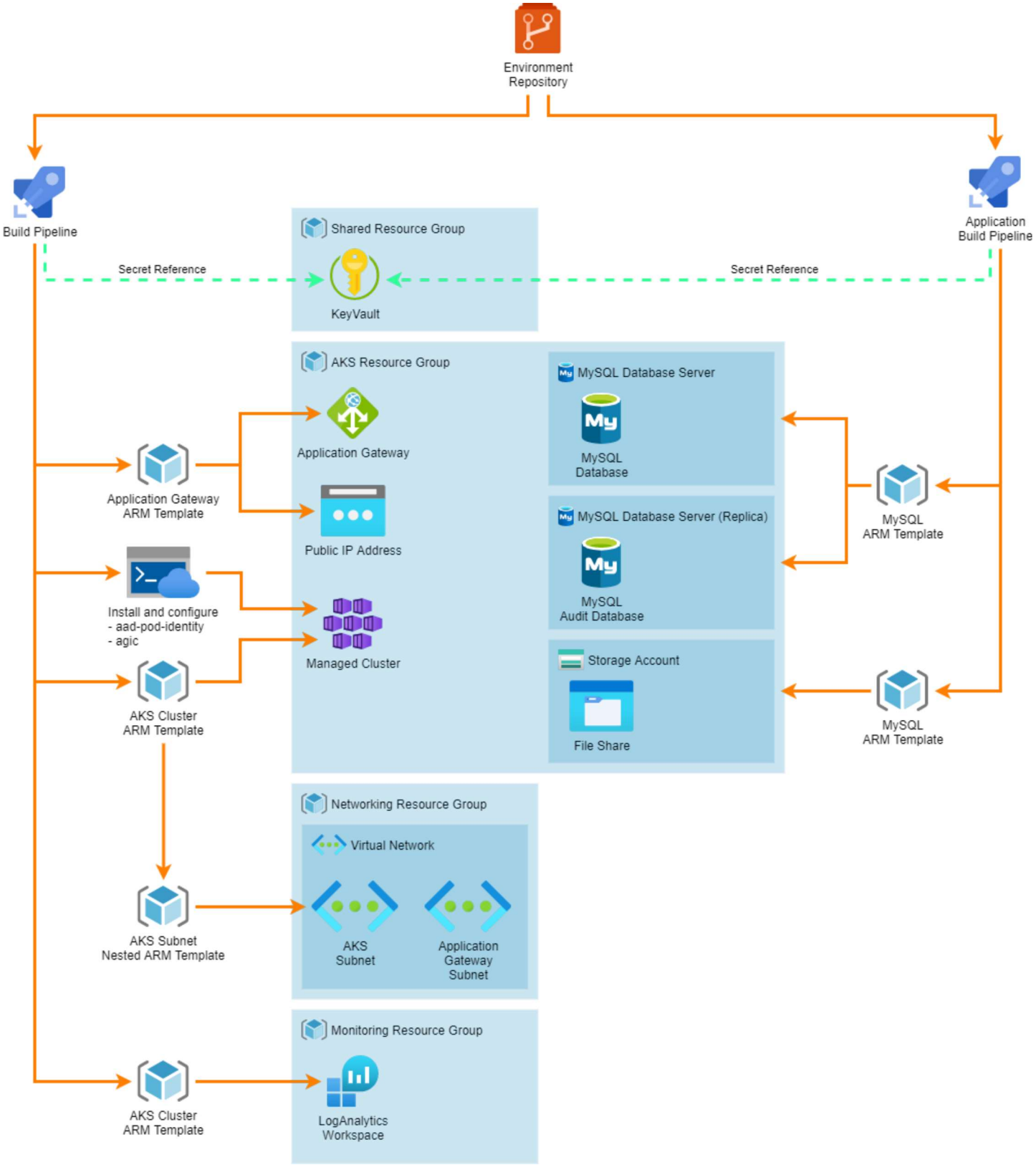
Secrets Management

All Secrets are stored in a manually configured Azure KeyVault. In order to make use of those secrets in Azure DevOps Pipelines a variable group is created which is linked to the preexisting KeyVault. The SPN used for the DevOps Pipelines has list and get access to this KeyVault, so it can retrieve stored secrets during runtime.

In a few cases the KeyVault is directly referenced in an Azure Resource Manager template (= ARM Template).

Deployment of Azure resources

For the deployment of the Infrastructure and the application itself Azure DevOps CI/CD pipelines are used to have a streamlined experience and centralized manageability.



First, all Azure resource for the base infrastructure are deployed in the following order:

1. Log Analytics workspace
2. Application Gateway
3. Managed Azure Kubernetes Cluster

Application specific resources are being deployed in a separate pipeline. This allows for better maintenance and flexibility for deployment of other applications in the future.

One Pipeline deploys the MySQL Database with all necessary resources as well as the Azure Fileshares by executing ARM Templates. The main database is being replicated to another database. The SQL Admin

password is set by a reference to the manually managed KeyVault in the ARM Template.

Configuration of Kubernetes Cluster

After deploying the Azure resources, the Kubernetes Cluster is being configured.

The helm package aad-pod-identity gets installed. It will be used so pods can access other Azure resources and configure them (e.g. Application Gateway).

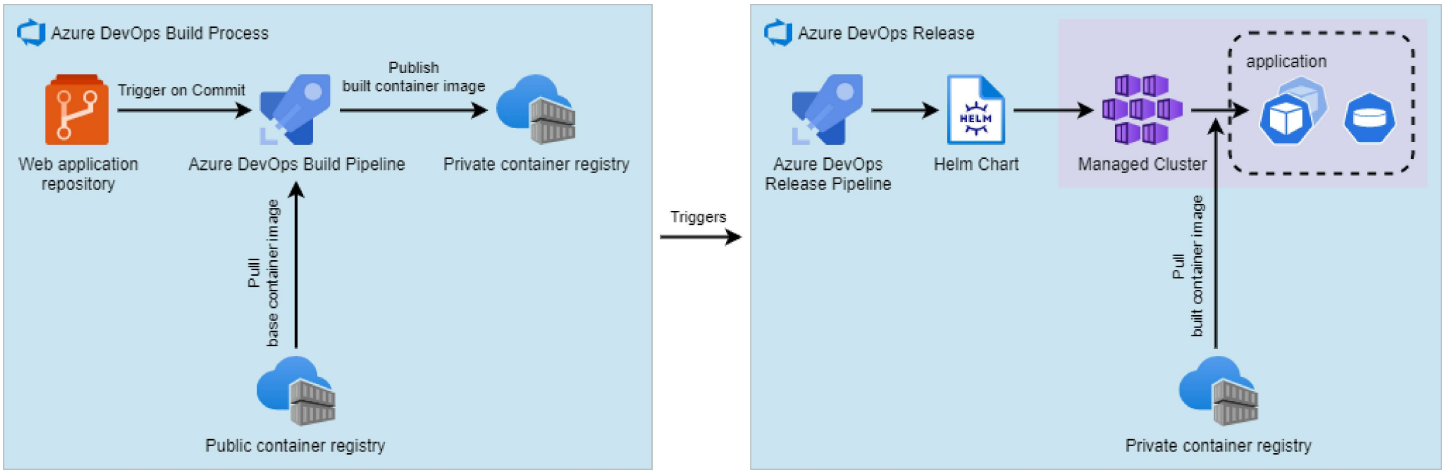
```
helm repo add aad-pod-identity https://raw.githubusercontent.com/Azure/aad-pod-identity/master/charts
helm repo update
helm upgrade aad-pod-identity --install aad-pod-identity/aad-pod-identity
```

The Kubernetes cluster is configured to use the Application Gateway as ingress using Application Gateway Ingress Controller (=agic). This controller is installed in a separate namespace called agic.

```
kubectl create namespace agic --output yaml --dry-run | kubectl apply --filename -
helm repo add application-gateway-kubernetes-ingress https://appgwingress.blob.core.windows.net/ingress-azu
helm repo update
helm upgrade agic --install --namespace agic --version $(agicVersion) \
  --set appgw.subscriptionId=$(subscriptionId) \
  --set appgw.resourceGroup=$(ResourceGroupName) \
  --set appgw.name=$(applicationGateWayName) \
  --set appgw.subnetName=$(applicationGatewaySubnetName) \
  --set appgw.subnetID=$(applicationGatewaySubnetId) \
  --set appgw.shared=false \
  --set appgw.usePrivateIP=false \
  --set armAuth.type=aadPodIdentity \
  --set armAuth.identityResourceID=$(appGwOutput.identityResourceId) \
  --set armAuth.identityClientID=$(appGwOutput.identityClientId) \
  --set rbac.enabled=true \
  application-gateway-kubernetes-ingress/ingress-azure
```

Container Build and Deployment Process

All changes code changes to the web application in the corresponding Azure DevOps repository trigger a build pipeline. This pipeline compiles the application for a release, pulls a base container image from a public container registry and builds a predeployed container. If the build succeeds a release pipeline is then started. The release pushes a helm chart to the AKS cluster which describes the container and all other necessary kubernetes resources and configuration (e.g. volumes, services, agic configuration, ...). The cluster then pulls the image from the private container registry and starts all defined components.

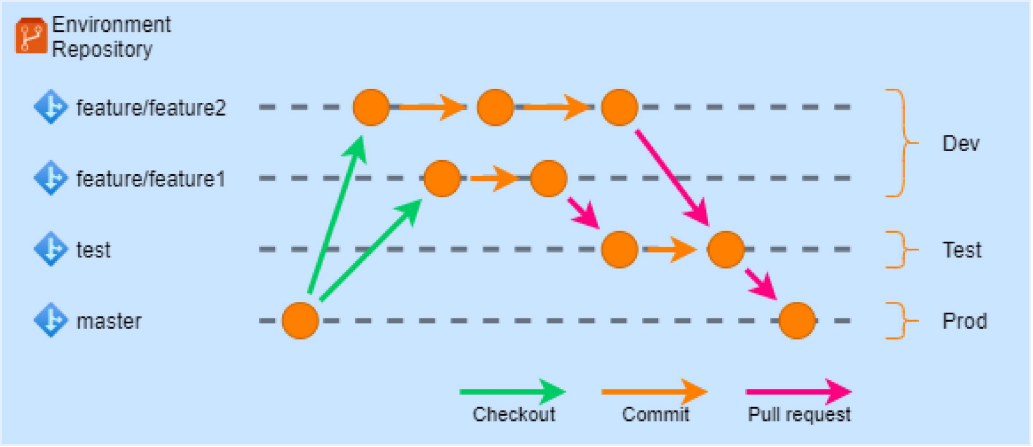


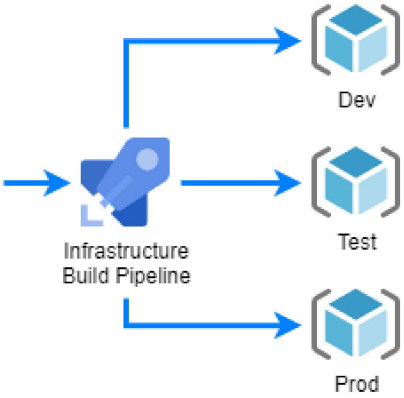
Staging

For each stage a separate AKS cluster including all Azure Resources is deployed. This ensures that all changes are thoroughly tested prior deployment to production in order to keep service availability and quality as high as possible.

The environment is deployed in three stages:

- Development
- Test
- Prod





An Azure DevOps build pipeline deploys all stages based on the branch of the last git commit.
It is mapped as follows:

Stage	Branch
Dev	refs/feature/*
Test	refs/test
Prod	refs/master

Conclusion

The implemented solution allows our customer to extend the application with other components, deliver features faster to endusers and keep the environment scalable with high availability. Also, the AKS cluster in its configuration allows hosting multiple applications in different namespaces.