AKS Complex Solutions CaseStudy2

Last updated by | Fabian Flanhardt | Apr 11, 2021 at 4:34 PM GMT+2

AKS Complex Solutions Case Study 2

Contents

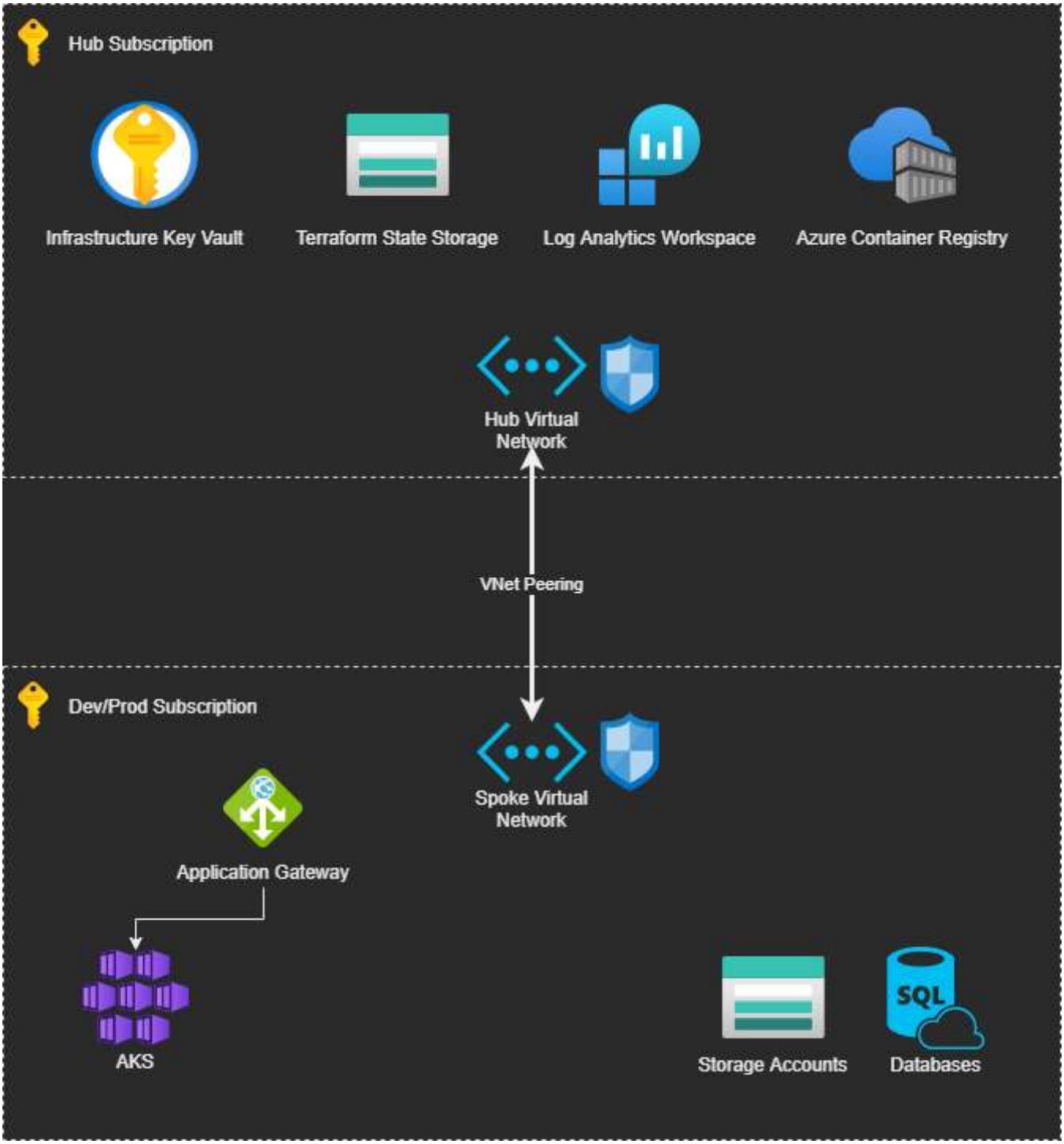
Challenge

Our customer has an existing self-developed software and requires it to be run on Azure Kubernetes Services. The software was designed according to microservice architecture and consists of a web-based frontend, a backend and an adapter service. It additionally depends on multiple Azure PaaS services for storage and databases. The services need be containerized, stored in an Azure Container Registry and deployed to AKS via Helm-Charts. There is no existing Kubernetes infrastructure which means that it has to be implemented in context of this migration as well. All infrastructure deployments are to be implemented via Terraform configuration files and automated via CI/CD pipelines. The deployments of the application services are to be automated via CI/CD pipelines as well.

Microsoft Technologies

- Azure Key Vault
- Azure SQL
- Azure Storage Account
- Azure Kubernetes Services
- Azure Virtual Networks
- Azure Log Analytics Workspace
- Azure DNS
- Azure Public IP
- Azure Application Gateway
- Azure Container Registry

Target Infrastructure



Secrets Management

Secrets are managed in Azure Key Vaults where they can be directly accessed by AKS and other resources. Secrets used in the CI/CD process are stored protected and (in some cases) masked as repository variables in GitLab.

Deployment of Azure resources

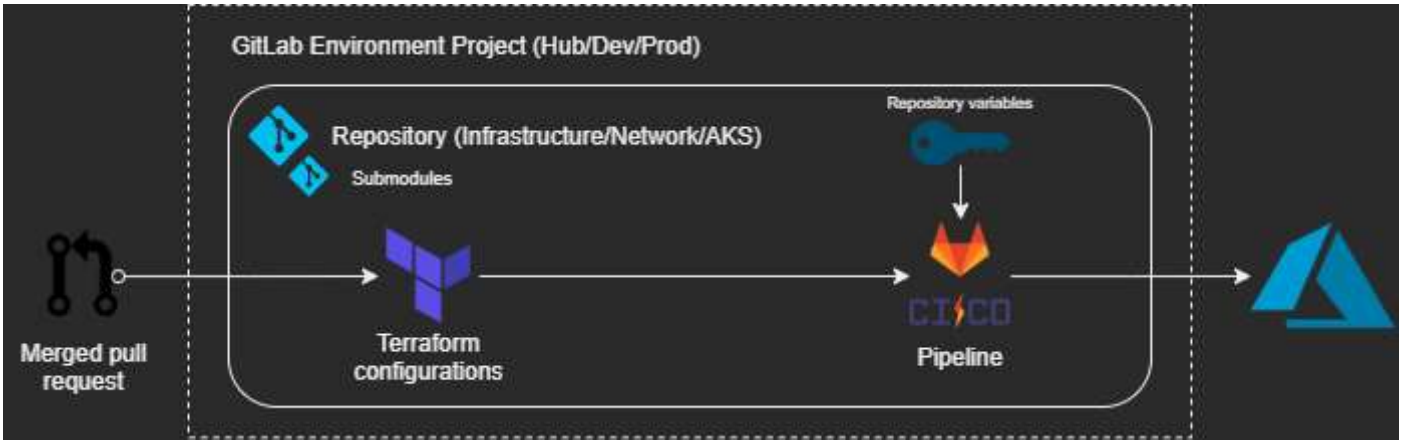
All Azure resources that need to be deployed in context of this migration project are required to be automatically deployed based on infrastructure as code (IaC) and CI/CD pipelines. The customer requested that all IaC is written using Terraform configuration files. For the automation of the deployment the customer requested that it has to be done using his existing DevOps lifecycle tool GitLab. GitLab offers CI/CD pipelines which can be used for the automatization.

The actual implementation relies heavily on Terraform Modules, reusable Terraform configurations that can be called and configured by other configurations. Modules often consist of multiple resources that are

logically bound together which allows to describe infrastructure in terms of architecture. Modules are stored in a dedicated repository and are integrated into other repositories as a git submodule.

The official Terraform provider for Azure (azurerm) is used for deployment. Terraform state files are stored in a central storage account in the Hub subscription. Authentication to Azure is done via individual service principals with owner assignments for each subscription. This is managed in a dedicated "Identity" repository. It is also used to define custom roles, applications, policies, role assignments and additionally needed service principals.

An "Infrastructure" repository is used to deploy essential resources into the subscription. It includes an "initial_deployment" module, which deploys resource groups, Virtual Network, Network Watcher, Log Analytics Workspace, Storage Account, Key Vault and Key Vault Access Policies. Additionally, Subnets, Route Tables, Network Security Groups and Network Security Rules are deployed.



Configuration of Kubernetes Cluster

A separate repository is used to deploy AKS. It makes use of an AKS Terraform Module and relies on Terraform Remote States of Identity and Infrastructure deployments. All essential AKS configuration is done via Terraform. This includes basic scaling (e.g. VM size and node count), networking properties like AKS Subnet configuration, Application Gateway configuration and Application Gateway Ingress Controller configuration, Service Principal configuration, as well as Kubernetes and Orchestrator version configuration.

AKS is deployed in each environment inside of a Spoke that is peered to the Hub Virtual Network. This follows Microsoft best practices by utilizing the Hub and Spoke Topology. The 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".

The customers preferred monitoring solution for the cluster itself as well as for workloads running on the cluster is Azure Monitor. Container insights is used to monitor cluster performance, container metrics and container health. Custom workbooks are created by the customers Ops-Teams for individual monitoring requirements.

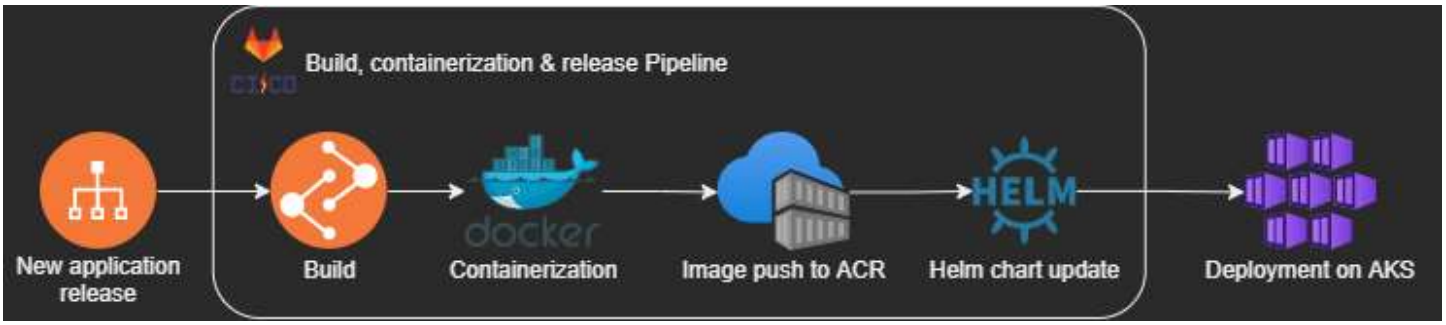
Container Build and Deployment Process

New application releases (based on git tags) in the application repository trigger a build, containerization and release pipeline. When the creation of an application image is completed, it gets pushed into an Azure Container Registry, to which AKS has access. Kubernetes objects are managed, packaged and deployed via Helm. The respective updated to the Helm chart are also done by the pipeline.

Application deployment on AKS happens similarly to the already described infrastructure deployment process. Each application that is to be deployed on AKS has its own repository. It includes Terraform configuration files for deployment of necessary application-dependent infrastructure like storage accounts

or databases. The created Helm charts for all of the applications microservices are also inside of the repository. In case of a merge of a pull request the CI/CD pipeline will deploy infrastructure changes as described in the Terraform configuration files as well as redeploy the microservices as necessary by using the Helm charts.

```
AKS application repository
└── Application
    ├── .gitlab-ci.yaml
    ├── Helm charts
    │   ├── Adapter
    │   │   ├── Chart.yaml
    │   │   ├── templates
    │   │   │   ├── deployment.yaml
    │   │   │   └── service.yaml
    │   │   └── values.yaml
    │   ├── Backend
    │   │   ├── Chart.yaml
    │   │   ├── templates
    │   │   │   ├── deployment.yaml
    │   │   │   └── service.yaml
    │   │   └── values.yaml
    │   └── Frontend
    │       ├── Chart.yaml
    │       ├── templates
    │       │   ├── deployment.yaml
    │       │   └── service.yaml
    │       └── values.yaml
    └── main.tf
```



Staging

Each stage defined by the customer has it's own environment. This enables development, test and production in separation. The customer requested a development and production environment for the application deployment on AKS.

Conclusion

The solution fits the customers requirements. The application is running reliably on AKS. Changes to the cluster and the surrounding infrastructure are easily made by changing the Terraform configuration files that were created. Infrastructure changes are transparent and comprehensible due to version control. The implemented CI/CD flows work as expected and are therefore enabling the customers application development teams to seamlessly integrate new releases into the existing system landscape. This decreases the amount of time needed for a new release to go into production substantially.