Handover document (User manual)



Table of Contents

[Overview 3](#_Toc156550499)

[Introduction 3](#_Toc156550500)

[Prerequisites 3](#_Toc156550501)

[Azure Prerequisites: 3](#_Toc156550502)

[Minikube Prerequisites: 3](#_Toc156550503)

[Setup and Deployment 3](#_Toc156550504)

[Azure Kubernetes Service (AKS) 3](#_Toc156550505)

[Minikube 5](#_Toc156550506)

[Configuration 6](#_Toc156550507)

[Azure Configuration 6](#_Toc156550508)

[Kubernetes Configuration 9](#_Toc156550509)

[Usage 15](#_Toc156550510)

[Accessing the Cluster 15](#_Toc156550511)

[For Azure Kubernetes Service (AKS): 15](#_Toc156550512)

[For Minikube: 15](#_Toc156550513)

[Deploying Applications 16](#_Toc156550514)

[Deployment: 16](#_Toc156550515)

[Best Practices: 17](#_Toc156550516)

[Scaling Resources 18](#_Toc156550517)

[Maintenance 21](#_Toc156550518)

[Upgrading Kubernetes 21](#_Toc156550519)

[Upgrading Kubernetes in Azure (AKS): 21](#_Toc156550520)

[Upgrading Kubernetes in Minikube: 21](#_Toc156550521)

[Additional Notes: 22](#_Toc156550522)

[Monitoring and Logging 22](#_Toc156550523)

[Backup and Restore 22](#_Toc156550524)

[Troubleshooting 22](#_Toc156550525)

[Contact Information 22](#_Toc156550526)

# Overview

This document serves as a comprehensive guide for understanding and managing the Kubernetes cluster hosted on Azure and Minikube. All necessary information and resources are available on the GitHub repository for easy redeployment.

GitHub Repository: [Your GitHub Repository URL]

# Introduction

This document serves as a handover guide for the Kubernetes cluster deployed for the website application. The cluster is hosted on Azure and Minikube, ensuring scalability, reliability, and cost-effectiveness. It has either 1 node and 4 pods or 4 nodes with 2 pods in each of them, depending on where it is hosted, because minikube cannot simulate multiple nodes.

# Prerequisites

## Azure Prerequisites:

Azure account with necessary permissions.

Azure Kubernetes Service (AKS) configured.

Kubectl configured.

Azure CLI: Install the Azure Command-Line Interface (CLI) on your local machine. You can download it from here.

Azure Subscription: You need an active Azure subscription. If you don't have one, create an account on Azure Portal.

Minikube Prerequisites:

Minikube installed on the local machine.

Kubectl configured.

Hypervisor: Minikube requires a hypervisor to create virtual machines. Install a hypervisor of your choice, such as VirtualBox, VMware, or Hyper-V. Or use docker directly.

# Setup and Deployment

## Azure Kubernetes Service (AKS)

Step 1: Login to Azure CLI

Open a terminal or command prompt and log in to your Azure account:

az login

Follow the instructions on the screen to authenticate and authorize the Azure CLI.

Step 2: Create a Resource Group

Create an Azure Resource Group to logically group your resources:

az group create --name <ResourceGroupName> --location <AzureRegion>

Replace <ResourceGroupName> with a name for your resource group, and <AzureRegion> with the Azure region you want to use (e.g., eastus, westeurope, etc.).

Step 3: Create AKS Cluster

Run the following command to create an AKS cluster:

az aks create --resource-group <ResourceGroupName> --name <AKSClusterName> --node-count <NodeCount> --enable-addons monitoring --generate-ssh-keys

Replace <AKSClusterName> with a unique name for your AKS cluster.

Adjust <NodeCount> based on the number of nodes you want in your cluster.

Step 4: Get AKS Credentials

To configure kubectl to connect to your AKS cluster, run:

az aks get-credentials --resource-group <ResourceGroupName> --name <AKSClusterName>

Step 6: Verify Cluster Connection

Ensure you can connect to your AKS cluster by running:

kubectl get nodes

You should see a list of nodes in your AKS cluster.

Step 7: Configuring Network Policies (Optional)

If you want to use network policies in your cluster, you can enable it(you should probably do it):

az aks enable-addons --resource-group <ResourceGroupName> --name <AKSClusterName> --addons azure-policy --workspace-resource-id <LogAnalyticsWorkspaceResourceId>

Replace <LogAnalyticsWorkspaceResourceId> with the resource ID of your Azure Log Analytics workspace.

Step 8: Clean Up (Optional)

If you need to delete your AKS cluster, run:

az aks delete --resource-group <ResourceGroupName> --name <AKSClusterName> --yes --no-wait

Conclusion

Your AKS cluster is now deployed and ready for use. You can deploy applications, configure networking, and scale resources using kubectl or Azure Portal.

## Minikube

Step 1: Start Minikube Cluster

Open a terminal or command prompt and run the following command to start Minikube:

minikube start

Minikube will create a virtual machine and set up a Kubernetes cluster locally.

Step 2: Check Cluster Status

After Minikube has started, verify the status of your cluster:

minikube status

You should see a message indicating that the cluster is running.

Step 4: Interact with Minikube Cluster

Minikube automatically configures kubectl to use the newly created cluster. To verify, run:

kubectl get nodes

You should see a single node listed, indicating that Minikube is running a single-node Kubernetes cluster.

Step 5: Accessing the Kubernetes Dashboard (Optional)

Minikube comes with a built-in Kubernetes dashboard. To access it, run:

minikube dashboard

This will open the Kubernetes dashboard in your default web browser.

Step 6: Deploying Applications

You can now deploy your applications to the Minikube cluster using standard Kubernetes manifests. For example:

kubectl apply -f deployment.yaml

Step 7: Stopping and Deleting Minikube Cluster

When you're done, you can stop and delete the Minikube cluster:

minikube stop

minikube delete

Additional Considerations:

Minikube Configurations: You can customize Minikube's behavior using various configurations. For example, you can specify a different VM driver, allocate more resources, or use a specific Kubernetes version. Refer to the Minikube documentation for more details.

minikube start --vm-driver=<driver> --cpus=<num> --memory=<size> --kubernetes-version=<version>

Proxy Configuration (if needed): If your environment requires a proxy, you may need to configure Minikube to use it. Refer to the Minikube documentation for proxy-related configurations.

Conclusion:

You now have a local Kubernetes cluster running on Minikube. You can use it for development, testing, and learning Kubernetes concepts without the need for a cloud environment. Refer to the Minikube documentation for more advanced configurations and options.

# Configuration

## Azure Configuration

1. Networking Configuration:

Virtual Network (VNet):

Create a dedicated virtual network for your AKS cluster:

az network vnet create \

--resource-group <ResourceGroupName> \

--name <VNetName> \

--address-prefixes 10.0.0.0/8 \

--subnet-name <SubnetName> \

--subnet-prefix 10.0.0.0/16

AKS Subnet:

Create a subnet within the VNet for AKS:

az network vnet subnet create \

--resource-group <ResourceGroupName> \

--name <SubnetName> \

--vnet-name <VNetName> \

--address-prefixes 10.0.1.0/24

AKS Cluster:

Create the AKS cluster, associating it with the specified VNet and Subnet:

az aks create \

--resource-group <ResourceGroupName> \

--name <AKSClusterName> \

--node-count 4 \

--network-plugin azure \

--vnet-subnet-id $(az network vnet subnet show --resource-group <ResourceGroupName> --vnet-name <VNetName> --name <SubnetName> --query id -o tsv) \

--service-principal <ServicePrincipalID> \

--client-secret <ServicePrincipalSecret> \

--node-vm-size <VMSize> \

--enable-addons monitoring \

--generate-ssh-keys

2. Security Configuration:

Azure AD Integration:

Integrate AKS with Azure Active Directory for role-based access control (RBAC):

az aks update \

--resource-group <ResourceGroupName> \

--name <AKSClusterName> \

--enable-aad \

--aad-admin-group-object-ids <AdminGroupObjectIDs>

<AdminGroupObjectIDs>: Azure AD group object IDs for cluster administrators.

AKS RBAC Configuration:

Enable RBAC in AKS for fine-grained access control:

az aks update \

--resource-group <ResourceGroupName> \

--name <AKSClusterName> \

--enable-rbac \

--service-principal <ServicePrincipalID> \

--client-secret <ServicePrincipalSecret>

3. Authentication Configuration:

Configure kubectl for AKS:

Run the following command to configure kubectl to connect to your AKS cluster:

az aks get-credentials --resource-group <ResourceGroupName> --name <AKSClusterName>

4. Storage Configuration:

Azure Disk Storage Class:

If you have a volume mounted in Azure Storage, create a storage class for Azure Disk:

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: azure-disk

provisioner: kubernetes.io/azure-disk

Persistent Volume (PV) and Persistent Volume Claim (PVC):

Create a Persistent Volume and Persistent Volume Claim for your Azure Disk:

apiVersion: v1

kind: PersistentVolume

metadata:

name: azure-disk-pv

spec:

capacity:

storage: 5Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

azureDisk:

kind: Managed

diskName: <AzureDiskName>

diskURI: <AzureDiskURI>

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: azure-disk-pvc

spec:

accessModes:

- ReadWriteOnce

storageClassName: azure-disk

resources:

requests:

storage: 5Gi

<AzureDiskName>: The name of your Azure Disk.

<AzureDiskURI>: The URI of your Azure Disk.

Conclusion:

These configurations provide a foundation for deploying a Kubernetes cluster on Azure with specific network, security, and storage settings. Adapt the parameters and YAML definitions based on your actual environment and application requirements. Review the Azure AKS documentation for more advanced configurations and options.

## Kubernetes Configuration

For the configuration of your Kubernetes cluster, especially considering the setup of two APIs (users and tasks), an authentication service, and a frontend interacting with the APIs, you'll need to define various Kubernetes resources. Below are examples of what your Kubernetes configuration might look like.

1. Namespace:

Create a Kubernetes namespace to isolate your resources:

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace

2. Deployment for API Services:

Create Deployments for your APIs (users and tasks):

apiVersion: apps/v1

kind: Deployment

metadata:

name: users-api

namespace: my-namespace

spec:

replicas: 2

selector:

matchLabels:

app: users-api

template:

metadata:

labels:

app: users-api

spec:

containers:

- name: users-api

image: your-registry/users-api:latest

ports:

- containerPort: 8080

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: tasks-api

namespace: my-namespace

spec:

replicas: 2

selector:

matchLabels:

app: tasks-api

template:

metadata:

labels:

app: tasks-api

spec:

containers:

- name: tasks-api

image: your-registry/tasks-api:latest

ports:

- containerPort: 8080

3. Service for API Services:

Create Services for your APIs:

apiVersion: v1

kind: Service

metadata:

name: users-api-svc

namespace: my-namespace

spec:

selector:

app: users-api

ports:

- protocol: TCP

port: 80

targetPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: tasks-api-svc

namespace: my-namespace

spec:

selector:

app: tasks-api

ports:

- protocol: TCP

port: 80

targetPort: 8080

4. Deployment for Authentication Service:

Create a Deployment for your authentication service:

apiVersion: apps/v1

kind: Deployment

metadata:

name: auth-service

namespace: my-namespace

spec:

replicas: 1

selector:

matchLabels:

app: auth-service

template:

metadata:

labels:

app: auth-service

spec:

containers:

- name: auth-service

image: your-registry/auth-service:latest

ports:

- containerPort: 8080

5. Service for Authentication Service:

Create a Service for your authentication service:

apiVersion: v1

kind: Service

metadata:

name: auth-service-svc

namespace: my-namespace

spec:

selector:

app: auth-service

ports:

- protocol: TCP

port: 80

targetPort: 8080

6. Deployment for Frontend:

Create a Deployment for your frontend application:

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend-app

namespace: my-namespace

spec:

replicas: 1

selector:

matchLabels:

app: frontend-app

template:

metadata:

labels:

app: frontend-app

spec:

containers:

- name: frontend-app

image: your-registry/frontend-app:latest

ports:

- containerPort: 80

7. Service for Frontend:

Create a Service for your frontend application:

apiVersion: v1

kind: Service

metadata:

name: frontend-app-svc

namespace: my-namespace

spec:

selector:

app: frontend-app

ports:

- protocol: TCP

port: 80

targetPort: 80

8. Ingress (Optional):

If you want to expose your frontend using Ingress:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: frontend-ingress

namespace: my-namespace

spec:

rules:

- host: frontend.yourdomain.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: frontend-app-svc

port:

number: 80

Remember to replace placeholders like your-registry, yourdomain.com, etc., with your actual configurations.

# Usage

## Accessing the Cluster

### For Azure Kubernetes Service (AKS):

If you're using Azure AKS, you can obtain the cluster credentials using the Azure CLI:

az aks get-credentials --resource-group <resource-group-name> --name <aks-cluster-name>

Replace <resource-group-name> and <aks-cluster-name> with your actual values.

### For Minikube:

If you're using Minikube, the credentials are automatically configured. You can simply start Minikube:

minikube start

3. Verify Connection:

To ensure that kubectl is configured correctly and is pointing to the right cluster, you can use the following command to get the cluster information:

kubectl cluster-info

This command should display the Kubernetes master and other services running on the cluster.

4. Interact with the Cluster:

Now that you have configured kubectl with the cluster credentials, you can start interacting with the Kubernetes cluster. Here are some common kubectl commands:

To view nodes in the cluster:

kubectl get nodes

To get a list of all pods in the default namespace:

kubectl get pods

To access a shell inside a pod:

kubectl exec -it <pod-name> -- /bin/bash

5. Configuration Files:

The configuration for kubectl is stored in a file called kubeconfig. By default, it is located at ~/.kube/config. This file includes information about clusters, users, and contexts.

6. Switching Contexts:

If you're working with multiple Kubernetes clusters, you can switch between them using contexts. To list available contexts:

kubectl config get-contexts

To switch to a specific context:

kubectl config use-context <context-name>

Replace <context-name> with the desired context.

## Deploying Applications

### Deployment:

Define a Deployment YAML file, e.g., my-app-deployment.yaml:

apiVersion: apps/v1 kind: Deployment metadata: name: my-app spec: replicas: 3 selector: matchLabels: app: my-app template: metadata: labels: app: my-app spec: containers: - name: my-app-container image: your-registry/your-app-image:tag

Apply the Deployment:

kubectl apply -f my-app-deployment.yaml

2. Create a Service:

A Service provides network access to a set of pods. Define a Service YAML file, e.g., my-app-service.yaml:

apiVersion: v1 kind: Service metadata: name: my-app-service spec: selector: app: my-app ports: - protocol: TCP port: 80 targetPort: 8080 type: LoadBalancer

Apply the Service:

kubectl apply -f my-app-service.yaml

3. Verify Deployment:

Check the status of the Deployment:

kubectl get deployments

Check the status of the Pods:

kubectl get pods

4. Access the Application:

If using a LoadBalancer service type, get the external IP:

kubectl get service my-app-service

Access the application using the external IP.

### Best Practices:

Use Configurations:

Externalize configurations using ConfigMaps and Secrets.

Avoid hardcoding configuration values in your application code.

Health Checks:

Define readiness and liveness probes to improve reliability.

Adjust probe parameters based on your application's characteristics.

Resource Requests and Limits:

Set resource requests and limits for CPU and memory in your Deployment.

Helps with resource allocation and scaling decisions.

Namespace Isolation:

Use Kubernetes namespaces to isolate applications and resources.

Helps organize and manage different environments or projects.

Labels and Annotations:

Use labels to organize and categorize resources.

Leverage annotations for additional metadata.

Rolling Updates:

Use Deployment strategies for rolling updates.

Minimize downtime and ensure zero-downtime deployments.

Secrets Management:

Store sensitive information in Kubernetes Secrets.

Mount secrets as volumes or use them as environment variables.

Monitoring and Logging:

Implement monitoring with tools like Prometheus and Grafana.

Use centralized logging to capture application logs.

CI/CD Pipelines:

Implement a CI/CD pipeline for automated deployments.

Integrate with tools like Jenkins, GitLab CI, or GitHub Actions.

Documentation:

Maintain clear and up-to-date documentation for your applications and deployments.

Helps with onboarding and troubleshooting.

## Scaling Resources

Scaling resources within a Kubernetes cluster involves adjusting the number of pods, replicas, or other resources to meet the changing demands of your application. Below are explanations on how to scale various resources within the cluster, including pods, deployments, and services:

1. Scaling Pods:

Scaling pods typically involves adjusting the number of replicas in a Deployment or ReplicationController. Use the kubectl scale command:

kubectl scale deployment <deployment-name> --replicas=<desired-replica-count>

Replace <deployment-name> with the name of your Deployment and <desired-replica-count> with the desired number of replicas.

Example:

kubectl scale deployment my-app --replicas=5

This scales the number of pods managed by the Deployment named my-app to 5.

2. Autoscaling Pods:

Kubernetes also supports autoscaling based on resource usage. To enable Horizontal Pod Autoscaler (HPA), define an HPA resource for your Deployment and set the desired resource metrics:

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 1

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

targetAverageUtilization: 80

Apply the HPA:

kubectl apply -f my-app-hpa.yaml

3. Scaling Deployments:

If you want to scale a Deployment directly without creating an HPA, you can update the replicas in the Deployment manifest:

kubectl edit deployment <deployment-name>

This opens the Deployment manifest in the default text editor. Update the replicas field and save the file.

Example:

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 3 # Change this to the desired replica count

# ...

Save the file, and the Deployment will be automatically updated with the new replica count.

4. Scaling Services:

Kubernetes Services do not have a direct replica count, but you can scale the number of pods behind a service by adjusting the number of replicas in the associated Deployment or ReplicationController.

If your service is of type LoadBalancer, the cloud provider's load balancer will distribute traffic to the available pods.

5. Verify Scaling:

After scaling, you can verify the changes using commands like:

kubectl get pods

kubectl get deployment <deployment-name>

kubectl get hpa

Best Practices:

Monitor Resource Usage:

Regularly monitor the resource usage of your pods to make informed scaling decisions.

Set Resource Requests and Limits:

Configure resource requests and limits to guide the Kubernetes scheduler and ensure resource allocation.

Use Horizontal Pod Autoscaling:

Consider using HPA for dynamic scaling based on resource metrics.

Implement Readiness and Liveness Probes:

Use proper readiness and liveness probes to avoid scaling based on unhealthy pods.

Define Resource Quotas:

Implement resource quotas to control the total amount of resources consumed by a namespace.

Plan for Burst Scenarios:

Anticipate and plan for burst scenarios by setting appropriate scaling limits.

# Maintenance

## Upgrading Kubernetes

### Upgrading Kubernetes in Azure (AKS):

1. Check Compatibility:

Review the Kubernetes release notes to ensure compatibility with your applications.

Check Azure AKS documentation for any specific requirements or considerations.

2. Upgrade Control Plane:

Use Azure CLI or Azure Portal to initiate the control plane upgrade:

az aks upgrade --resource-group <resource-group-name> --name <aks-cluster-name> --kubernetes-version <new-version>

Replace <resource-group-name>, <aks-cluster-name>, and <new-version> with your actual values.

3. Upgrade Node Pools:

If you have multiple node pools, you may need to upgrade them individually:

az aks nodepool upgrade --resource-group <resource-group-name> --cluster-name <aks-cluster-name> --name <node-pool-name> --kubernetes-version <new-version>

Repeat this step for each node pool.

4. Verify Upgrade:

Monitor the upgrade progress using Azure CLI or Azure Portal.

Validate that your applications are running as expected.

### Upgrading Kubernetes in Minikube:

1. Update Minikube:

Ensure you have the latest version of Minikube installed:

minikube update-check minikube update

2. Delete the Existing Minikube Cluster:

Upgrading Minikube often involves creating a new cluster. Delete the existing cluster:

minikube delete

3. Start a New Minikube Cluster with the Desired Version:

Start a new Minikube cluster with the version you want:

minikube start --kubernetes-version=<new-version>

Replace <new-version> with the desired Kubernetes version.

4. Verify Upgrade:

Confirm the cluster is running the updated version:

kubectl version

Ensure your applications are working as expected.

### Additional Notes:

Always back up critical data and configurations before upgrading.

Consult the official documentation for any version-specific considerations.

Consider upgrading in a test environment first to identify potential issues.

## Monitoring and Logging

Refer to Design document or Deployment document.

## Backup and Restore

Refer to backup solution document.

## Troubleshooting

Refer to business continuity document.

## Contact Information

Stefan Dulgerov-CTO- 0612345678 - [fontys@fhict.nl](mailto:fontys@fhict.nl) (Monday-Friday 09:00 till 17:00)