Assessment - Azure Cloud DevOps

1. CI/CD with GitLab

1.1 GitLab Project Setup:

- Create a new project in GitLab
- Ensure the project repository contains your .NET application code.
- In this demo, the Dev branch will be used to deploy in the development environment and the Main branch will be used to deploy in the production environment.

1.2 GitLab Runner Setup:

- Install and configure GitLab Runner if it's not already set up.
- Ensure the runner is configured to work with your GitLab project.
- In this demo, Self-managed runner will be used to execute a job. Please follow this documentation to install Git Lab Self-managed runner
  - [https://docs.gitlab.com/runner/]
- Required software packages have to be installed such as dot NET SDK, Docker & Kubectl in self-managed runner.

1.3 Use Variable for reusability and security

Navigate to Your GitLab Project:

- Open your GitLab project in a browser.

Go to CI/CD Settings:

- In the left sidebar, go to Settings > CI/CD.

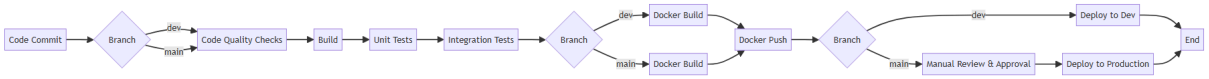
Add Variables:

- Expand the Variables section.
- Click Add Variable.
- Add the variables (SONAR_PROJECT_KEY, SONAR_TOKEN, DOCKER_REGISTRY, DOCKER_IMAGE_NAME, DOCKER_USERNAME, DOCKER_PASSWORD, KUBE_CONFIG).
- Set SONAR_TOKEN, DOCKER_PASSWORD to be masked for security.
- External Key Vaults can also be used to securely store and manage these secrets.

1.4 gitlab-ci.yml Configuration:

Create a gitlab-ci.yml file in the root of your project repository. This file will define the stages and jobs of your CI/CD pipeline.

1.5 Pipeline Architecture



GitLab Pipeline Design

- Auto deployment to Development Env.
- Need Approval and review before deployment to Production Env.

1.6 Pipeline Stages:

- Define the stages in your pipeline: stages: [code_quality, build, test, deploy]

1.7 Code Quality Checks:

- Add a job for code quality checks using tools like SonarQube.

1.8 Building the Application:

- Add a job for building the .NET application.

1.9 Unit Testing:

- Add a job for running unit tests.

1.10 Integration Testing:

- Add a job for running integration tests.

1.11 Docker Build & Push

- Add a job to build and push docker image to your container registry
- Please check the following link to see the Docker file.
  - [https://github.com/hellboyhha/donet-cicd-on-gitlab/blob/main/Dockerfile]
- Docker file configuration can be changed based on application requirements (e.g. expose additional port in swagger enabled application)

1.12 Deployment to Kubernetes:

- Add a job for deploying the application to a Kubernetes cluster.
- Use kubectl to manage Kubernetes resources.
- Please check the following link to see deployment.yaml and service.yaml. In this demo, the default namespace is used to deploy applications and services.
  - [https://github.com/hellboyhha/donet-cicd-on-gitlab/blob/main/deployment.yaml]
  - [https://github.com/hellboyhha/donet-cicd-on-gitlab/blob/main/service.yaml]
- Those Kubernetes configurations can be changed based on requirements (e.g. auto scaling rule for high availability, network restriction between pods according to security)

Final:

Please check the following link for full code:
[https://github.com/hellboyhha/donet-cicd-on-gitlab/blob/main/.gitlab-ci.yml].

2. Azure Kubernetes Service (AKS)

2.1 Microservices Deployment on an Azure Kubernetes Service (AKS) cluster

2.1.1 Before Deployment

2.1.1.1 HA Pillar: Redundancy

- **Controller Type (Deployment):** Kubernetes has various controllers for managing pod lifecycles. The most common is Deployment, which supports features like rollbacks. Others, like StatefulSet, are useful for maintaining pod identity post-recovery, while ReplicaSets lack some Deployment features.
- **Number of Replicas:** Setting replicas to one is a cold standby model, where a new instance starts from scratch if a failure occurs. This model is suitable for low-volume workloads but may affect availability for high-volume components.
- **Resource Requests and Limits:** By specifying resource limits, you can enable Horizontal Pod Autoscaling (HPA). HPA adjusts the number of replicas based on resource usage, helping to handle load surges and maintain application performance.
- Example Controller Type & Replicas yaml configuration:
  - [https://github.com/hellboyhha/dotnet-mysql-sample-app/blob/main/kubernetes%20deployment/api-deployment.yml]
- Example Horizontal Pod Scaler yaml configuration:
  - [https://github.com/hellboyhha/dotnet-mysql-sample-app/blob/main/kubernetes%20deployment/api-autoscaler.yaml]

2.1.1.2 HA Pillar: Monitoring

- **Liveness probes**, configuration spec.containers.livenessProbe, monitor the health of your pods. If a container crashes or exits, Kubernetes can detect it. When liveness fails, Kubernetes restarts the container.
- **Readiness probes**, configuration spec.containers.readinessProbe, determine whether to send traffic to the pod. If any pods of a deployment aren't ready, they won't be part of the endpoints of the Kubernetes service abstracting the deployment, and therefore won't be useful. It's important to carefully set the readiness probes, because they don't trigger a restart, but are used to isolate the pods from receiving traffic until they're ready.

- **Startup probes**, configuration spec.containers.startupProbe, mainly prevent false positives for readiness and liveness in slow-starting applications. Once the startup probe has succeeded, the liveness probe kicks in.

Note:

In my sample integration: I didn't use those types of probes because my sample application doesn't have health check endpoints.

2.1.1.3 HA Pillar: Recovery

- The main purpose of monitoring is to trigger recovery when it detects a failure. A recovery process involves three phases:
  - **Isolate and redirect:** Make sure the faulty replica isn't receiving traffic and direct its workload to healthy replicas.
  - **Repair:** Restart the faulty replica, which can repair transient errors.
  - **Rejoin**: After repair, if monitoring deems the replica healthy, rejoin the replica to other replicas in handling the workload.
- Service Type and Configuration
  - Exposing pods through a service benefits both redundancy and recovery, even if you have a single-replica deployment. Services automatically update DNS entries with Kubernetes service endpoints. Pods with failing readiness probes won't receive traffic, and while Kubernetes Cluster IP services have basic load balancing, combining them with ingress or service mesh solutions can improve load distribution.
- Restart Policy
  - The restartPolicy setting applies to all containers in a pod. Setting it to Never should be justified, such as avoiding excessive restart costs for containers that contact a license server on startup.

Please check example yaml configuration:

[https://github.com/hellboyhha/dotnet-mysql-sample-app/blob/main/kubernetes%20deployment/database-service.yaml]

2.1.1.4 HA Pillar: Checkpointing

While many modern applications are stateless, fully stateless apps are uncommon. Most manage state in the data layer. Kubernetes does not handle application state directly. Instead, it uses volumes, persistent volumes, and persistent volume claims to persist data at the file system or disk level, with the common approach being to store state as data records.

Please check example yaml configuration:
[https://github.com/hellboyhha/dotnet-mysql-sample-app/blob/main/kubernetes%20deployment/datafiles-persistentvolumeclaim.yaml]

2.1.1.5 Demo Project:

- [https://github.com/hellboyhha/dotnet-mysql-sample-app]

2.1.2 Deployment on Azure AKS

2.1.2.1 Available Methods to deploy:

- **Manual:** Use Azure CLI or kubectl to deploy directly.
- **Automation:** Utilize DevOps Tools (e.g. Azure DevOps or GitLab) for automated deployments.
- **Third-Party Tools:** Deploy with Helm for package management or Terraform for infrastructure as code.
- **GitOps:** Implement Flux CD or Argo CD for Git-based deployment management.

2.1.2.2 Deployment using AZ CLI

- **Login to Azure:** az login
- **Set the cluster subscription:** az account set --subscription <subscription-id>
- **Download cluster credentials:** az aks get-credentials --resource-group <resource-group-name> --name <cluster-name> --overwrite-existing
- **Apply Deployment:** kubectl apply -f sample-deployment.yaml

2.2 Azure Kubernetes Cluster (AKS) Deployment for High Availability and Scalability

2.2.1 Use Multiple Node Pools:

To create an AKS cluster with multiple node pools, adjust the following command:

- -

```
az aks create \
  --resource-group <ResourceGroupName> \
  --name <ClusterName> \
  --node-count 3 \
  --enable-addons monitoring \
  --generate-ssh-keys \
  --nodepool-name systempool \
  --node-vm-size Standard_DS2_v2 \
  --zones 1 2 3
```

- -

2.2.2 Enable Cluster Autoscaler:

To enable cluster autoscaler for the “userpool” node pool:

- -

```
az aks nodepool update \
  --resource-group <ResourceGroupName> \
  --cluster-name <ClusterName> \
  --name systempool \
  --enable-cluster-autoscaler \
  --min-count 3 \
  --max-count 10
```

- -

2.2.3 Managed Prometheus:

Managed Prometheus provides a metric collection for the AKS cluster.

- -

az aks create \

--resource-group <ResourceGroupName> \

--name <ClusterName> \

--enable-addons monitoring \

--generate-ssh-keys

- -

2.2.4 Container Insights:

Container Insights collects logs and performance data from the AKS cluster.

For Azure CLI, execute:

- -

az aks create/update --enable-azure-monitor-metrics --name <cluster-name> --resource-group <cluster-resource-group>

- -

2.2.5 Using Log Analytics Workspace:

Custom workspace:

- -

az aks enable-addons -a monitoring -n <ClusterName> -g <ResourceGroupName> --workspace-resource-id <LogAnalyticsWorkspaceResourceId>

- -

2.2.6 Setting Up Alerts and Dashboards:

To create a metric alert for high CPU usage:

- -

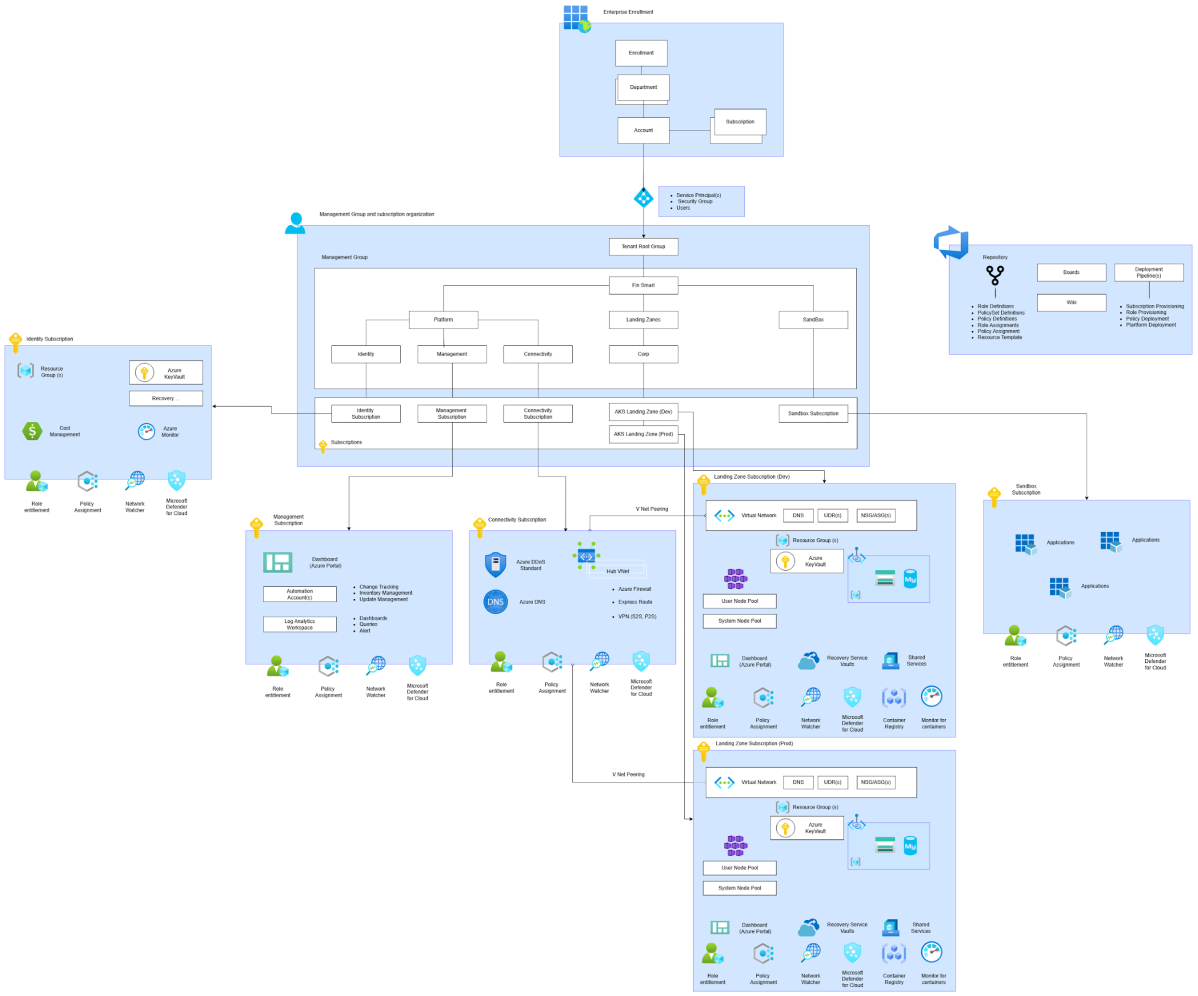
```
az monitor metrics alert create \
  --name "HighCPUAlert" \
  --resource-group <ResourceGroupName> \
  --scopes <ClusterResourceID> \
  --condition "avg Aggregation CPUPercentage > 80" \
  --description "Alert for high CPU usage"
```

- -

2.2.7 Setting Up Azure Backup:

- Create a Backup Vault in the same region as your AKS cluster.
- Install the Backup extension within your AKS cluster.
- Define backup policies (e.g., daily backups) and retain data for up to certain days.

3. Azure Landing Zone for AKS Deployment



Azure AKS Landing Zone Architecture

3.1 Identity and access management considerations for AKS

3.1.1 Design Recommendations

- **Cluster Identity:** Use a managed identity and define custom Azure RBAC roles.
- **Cluster Access:** Employ Kubernetes RBAC with Microsoft Entra ID for access control; use AKS-managed integration for authentication.
- **RBAC Configuration:** Define roles and bindings for SRE, SecOps, and developers in Kubernetes; consider Azure RBAC for unified management.
- **Access Management:** Grant SRE access just-in-time; use Privileged Identity Management and Microsoft Entra Workload ID for secure access.

3.2 Network topology and connectivity

3.2.1 Design Recommendations

- **Private AKS Cluster:** Deploy if your security policy requires the Kubernetes API to have a private IP.
- **DNS Configuration:** Use custom private DNS zones for private clusters rather than default system zones.
- **Networking:** Use Azure Container Networking Interface (CNI) unless IP address limitations exist. Follow IP address planning guidelines.
- **Node Pools:** Refer to the Windows AKS support FAQ for limitations on Windows Server node pools and virtual nodes.
- **DDoS Protection**: Use Azure DDoS Protection for the AKS cluster's virtual network or Azure Firewall/WAF in a centralized subscription.
- **DNS Setup:** Integrate with Azure Virtual WAN, hub-and-spoke architecture, Azure DNS zones, and your DNS infrastructure.
- **Private Link:** Secure connections to Azure services (e.g., Storage, Container Registry, SQL Database, Key Vault) with Private Link.
- **Ingress Controller:** Use for advanced HTTP routing and security; enforce TLS encryption for web applications.
- **Application Gateway:** Consider Azure Application Gateway Ingress Controller (AGIC) or in-cluster controllers like NGINX/Traefik.
- **Web Application Firewall:** Protect internet-facing and security-critical applications with Azure Web Application Firewall.
- **Egress Traffic:** Secure with Azure Firewall or a third-party NVA if your policy mandates inspecting all outbound traffic.
- **Load Balancer:** Prefer Standard tier over Basic tier for better performance.
- **Network Model:** Choose between Kubenet, Azure CNI, and Azure CNI Overlay based on your networking needs and capabilities.

3.3 Resource organization

3.3.1 Design Recommendations

- Naming and Tagging Standard:
  - **Workload Names:** Identify the workloads supported by each cluster.
  - **Cluster Resources:** Define resource alignment based on preceding considerations.
  - **Host Operator:** Identify the responsible team for host operations.
- OCI Artifact Registry Policy:
  - Implement a policy to use a specific registry according to your container registry topology.

3.4 Security

3.4.1 Design Recommendations

- **Cluster Access:** Use Azure RBAC to limit access to the Kubernetes configuration file. Secure pod access with minimal permissions.
- **Secrets Management:** Use Entra Workload ID and Azure Key Vault for secret protection.
- **Node Management**: Update node images or use kured for automated reboots.
- **Policy Enforcement:** Use Azure Policy add-on for Kubernetes and monitor with Microsoft Defender for Cloud.
- **Container Registry:** Deploy a private Azure Container Registry per landing zone and use Private Link.
- **Vulnerability Management:** Scan container images with Microsoft Defender Vulnerability Management or other solutions.

3.5 Governance Policy

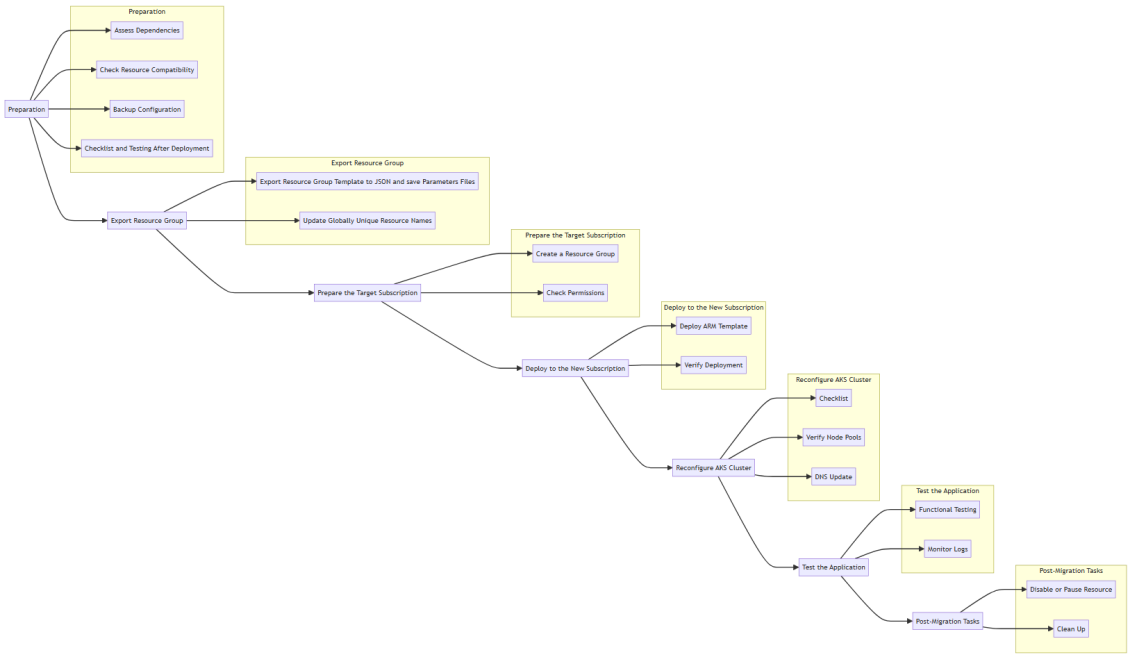
3.5.1 Design Recommendations

- **Policy Inheritance:** Familiarize yourself with Azure policies that AKS clusters will inherit based on enterprise-scale landing zone best practices.
- **Cluster Accessibility:** Decide if the AKS control plane should be exposed to the internet or restricted to a private network, adhering to organizational security policies.
- **Security Controls:** Use AppArmor and seccomp to enforce container security at the process level. Ensure Azure policies prevent privileged container creation and escalation.

- **Registry Access:** Evaluate whether your container registry should be accessible via the internet or restricted to a virtual network, considering the impact on CI/CD pipelines and other integrations.
- **Registry Deployment:** Choose between a shared container registry across multiple landing zones or dedicated registries per subscription.
- **Monitoring and Compliance:** Utilize Microsoft Defender for Kubernetes and Azure Policy to monitor and enforce security and compliance across clusters.

Note:
Using Azure Blueprints can help ensure consistent and standardized deployments by automating the setup of resources and policies across environments

4. Resource Group Export & Import



Resource Group Export & Import Process Diagram

4.1 Preparation

Assess Dependencies: Identify all the resources associated with the AKS cluster, such as virtual networks, storage accounts, and any other dependencies.

Check Resource Compatibility: Ensure that the resources can be moved to the new subscription and verify if any specific configurations or constraints need to be addressed.

Backup Configuration: Backup configurations and important data from the current resources.

Checklist and Testing After deployment: Make a list of test case scenario after deployment.

4.2 Export Resource Group

- Export resource group template, save the template as Json File and Parameters File.
  - AZ CLI: az group export --name <ResourceGroupName> --resource-group <ResourceGroupName> --output json > template.json
- Change resource name parameters of resources have globally unique naming requirements, meaning their names must be unique across all Azure regions and subscriptions.

4.3 Prepare the Target Subscription

- Create a Resource Group: In the new subscription, create a resource group where the resources will be deployed.
  - AZ CLI: az group create --name <NewResourceGroupName> --location <Location>
- Check Permissions: Ensure having the necessary permissions in the new subscription to deploy resources.

4.4 Deploy to the New Subscription

- Deploy ARM Template
  - AZ CLI: az deployment group create --resource-group <NewResourceGroupName> --template-file template.json --parameters @parameters.json
- Verify Deployment: Check that all resources have been created successfully and that there are no errors.

4.5 Reconfigure AKS Cluster

- **Checklist:** Check that any settings related to networking, monitoring, and integrations are correctly configured in the new subscription.
- **Verify Node Pools:** Check if the node pools are set up correctly.
- Update that any DNS records or networking configurations are updated to reflect the new subscription’s setup.

4.6 Test the Application (Testing After New Deployment)

- **Functional Testing:** Verify that the application is functional in the new subscription. This includes testing connectivity, application performance, and integration points.
- **Monitor Logs:** Check logs and monitoring tools to ensure there are no issues post-migration.

4.7 Post-Migration Tasks

- **Disable or Pause Resource:** Certain resources can be disabled or put in a paused state for a week to ensure the new deployment is working correctly before permanently decommissioning them.
- **Clean Up:** Remove or decommission the old resources if they are no longer needed.