제조업 MES 공장자동화 시스템

1. 하이브리드 클라우드 아키텍처 설계 요구사항

아래는 하이브리드 클라우드 아키텍처 설계입니다. 각 요구 사항을 기반으로 구현 방안을 제시하고, 이를 통해 전체 아키텍처를 구성합니다.

**1. 하이브리드 클라우드 인프라**

* **구현 방안**:
  + 온프레미스 인프라와 Azure 클라우드를 통합.
  + Azure Arc를 사용하여 온프레미스 및 클라우드 리소스를 일관되게 관리.
  + 온프레미스 데이터 센터와 Azure 클라우드 간에 안전한 연결 설정.
* **아키텍처 구성**:
  + 온프레미스 데이터 센터: 기존 MES 시스템, 로컬 데이터베이스.
  + Azure 클라우드: Azure Arc로 온프레미스 리소스 통합, Azure Resource Manager로 리소스 관리.

**2. 네트워크 연결성**

* **구현 방안**:
  + Azure ExpressRoute를 사용하여 온프레미스 데이터 센터와 Azure 클라우드 간 고속 연결 구축.
  + VPN 게이트웨이를 사용하여 각 공장과 클라우드 간 보안 네트워크 연결 설정.
* **아키텍처 구성**:
  + 온프레미스 네트워크와 Azure VNet 간의 ExpressRoute.
  + 각 공장에 VPN 게이트웨이 설치.

**3. 데이터 통합 및 동기화**

* **구현 방안**:
  + Azure Data Factory를 사용하여 데이터 이동 및 변환.
  + 여러 지역 간 데이터 동기화 및 통합 관리.
* **아키텍처 구성**:
  + 각 공장에서 수집된 데이터는 Azure Data Factory를 통해 Azure Data Lake Storage로 이동.
  + Azure SQL Database에서 통합 데이터 관리.

**4. 데이터 보호 및 보안**

* **구현 방안**:
  + Azure Security Center를 통해 통합 보안 관리.
  + Azure Policy 및 Azure Key Vault를 사용하여 각 지역의 규제 준수 및 데이터 보호.
* **아키텍처 구성**:
  + 각 데이터 소스에 대해 Azure Policy 적용.
  + 중요한 데이터는 Azure Key Vault에 저장 및 관리.

**5. 실시간 데이터 처리**

* **구현 방안**:
  + Azure IoT Hub를 통해 공장에서 실시간으로 데이터 수집.
  + Azure Stream Analytics를 사용하여 실시간 데이터 분석.
* **아키텍처 구성**:
  + 각 공장에 IoT 센서 설치, IoT Hub로 데이터 전송.
  + 실시간 분석을 위해 Azure Stream Analytics 사용.

**6. 확장성**

* **구현 방안**:
  + Azure Virtual Machines 및 Azure Kubernetes Service (AKS)를 사용하여 유연한 자원 확장.
* **아키텍처 구성**:
  + 필요에 따라 Azure VM 및 AKS 클러스터 확장.

**7. 재해 복구 및 고가용성**

* **구현 방안**:
  + Azure Site Recovery를 사용하여 재해 복구 계획 수립.
  + Azure Availability Zones를 통해 고가용성 아키텍처 구축.
* **아키텍처 구성**:
  + 재해 발생 시 Azure Site Recovery를 통한 빠른 복구.
  + 여러 Azure 지역에 걸친 고가용성 구성.

**8. 비용 관리**

* **구현 방안**:
  + Azure Cost Management를 통해 비용 모니터링 및 관리.
  + 예약 인스턴스 및 Azure Hybrid Benefit 활용.
* **아키텍처 구성**:
  + Azure Portal에서 비용 관리 및 예약 인스턴스 사용.

**9. 모니터링 및 관리**

* **구현 방안**:
  + Azure Monitor 및 Log Analytics를 통해 시스템 상태 모니터링.
  + Azure Automation을 통해 운영 효율성 향상.
* **아키텍처 구성**:
  + 각 서비스에 대한 모니터링 설정.
  + 운영 자동화를 위한 Azure Automation 사용.

**10. 개발 및 배포 파이프라인**

* **구현 방안**:
  + Azure DevOps를 통해 CI/CD 파이프라인 구축.
  + 코드 배포 자동화 및 인프라 구성 관리.
* **아키텍처 구성**:
  + 개발 환경에서 Azure DevOps 사용.
  + 배포 자동화를 위한 파이프라인 설정.

**11. AI 및 머신러닝**

* **구현 방안**:
  + Azure Machine Learning을 사용하여 생산 최적화 및 예측 분석 구현.
* **아키텍처 구성**:
  + Azure Machine Learning 모델을 사용한 데이터 분석.
  + 예측 모델을 통한 생산 효율화.

**12. API 및 서비스 통합**

* **구현 방안**:
  + Azure API Management를 통해 서비스 및 API 통합 관리.
* **아키텍처 구성**:
  + 각 서비스 API를 Azure API Management로 통합.
  + API 게이트웨이 설정 및 관리.

1. 전체 아키텍처 다이어그램

[온프레미스 데이터 센터] ---[ExpressRoute/VPN]--- [Azure 클라우드]

| |

| |

+--> [MES 시스템] +--> [Azure Arc]

| +--> [Azure Data Factory]

| +--> [Azure Security Center]

| +--> [Azure IoT Hub]

| +--> [Azure Stream Analytics]

| +--> [Azure VMs / AKS]

| +--> [Azure Site Recovery]

| +--> [Azure Monitor / Log Analytics]

| +--> [Azure DevOps]

| +--> [Azure Machine Learning]

| +--> [Azure API Management]

|

+--> [각 공장 (한국, 유럽, 베트남)]

|

+--> [IoT 센서] ---[VPN]--- [Azure IoT Hub]

|

+--> [데이터 수집 및 동기화] ---[Azure Data Factory]

|

+--> [실시간 분석 및 처리] ---[Azure Stream Analytics]

이 아키텍처는 제조업 MES 공장자동화 시스템을 위한 하이브리드 클라우드 솔루션으로, 한국, 유럽, 베트남에 걸친 확장성과 유연성을 제공하며, 안정적인 운영과 보안을 보장합니다.

Azure Kubernetes Service (AKS)에서 확장성(scale out 및 scale in)을 구현하는 방법은 다음과 같습니다. AKS는 자동 확장(Auto-scaling) 기능을 제공하여 필요에 따라 클러스터의 노드 및 파드 수를 동적으로 조절할 수 있습니다.

**AKS에서 Scale Out 및 Scale In 구현**

**1. Cluster Autoscaler**

Cluster Autoscaler는 AKS 클러스터의 노드 풀을 자동으로 조절합니다. 사용자가 정의한 최소 및 최대 노드 수에 따라 노드 풀의 크기를 확장하거나 축소합니다.

* **설정 방법**:
  1. AKS 클러스터를 생성하거나 기존 클러스터에 노드 풀을 추가할 때 최소 및 최대 노드 수를 설정합니다.

bash

코드 복사

az aks nodepool add \

--resource-group <resource-group> \

--cluster-name <cluster-name> \

--name <nodepool-name> \

--min-count <min-nodes> \

--max-count <max-nodes> \

--enable-cluster-autoscaler

* **동작 방식**:
  1. 워크로드가 증가하여 클러스터의 자원이 부족하면 Cluster Autoscaler가 자동으로 새로운 노드를 추가하여 확장(Scale Out)합니다.
  2. 워크로드가 감소하여 자원이 유휴 상태가 되면 Cluster Autoscaler가 자동으로 노드를 제거하여 축소(Scale In)합니다.

**2. Horizontal Pod Autoscaler (HPA)**

Horizontal Pod Autoscaler (HPA)는 배포된 애플리케이션의 파드 수를 자동으로 조절합니다. CPU 사용률, 메모리 사용률 등과 같은 메트릭을 기반으로 파드 수를 동적으로 확장 또는 축소합니다.

* **설정 방법**:
  1. HPA를 설정하기 위해 먼저 Kubernetes Metrics Server를 설치해야 합니다.

bash

코드 복사

kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

* 1. HPA를 정의하는 YAML 파일을 작성합니다. 예를 들어, CPU 사용률을 기준으로 파드 수를 조절하는 HPA는 다음과 같이 정의할 수 있습니다.

yaml

코드 복사

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: myapp-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: myapp

minReplicas: 1

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

* 1. HPA 리소스를 클러스터에 적용합니다.

bash

코드 복사

kubectl apply -f myapp-hpa.yaml

* **동작 방식**:
  1. 애플리케이션의 CPU 사용률이 설정된 임계값(예: 50%)을 초과하면 HPA가 파드 수를 증가시켜 확장(Scale Out)합니다.
  2. CPU 사용률이 설정된 임계값 이하로 떨어지면 HPA가 파드 수를 줄여 축소(Scale In)합니다.

**3. Azure Monitor 및 KEDA 사용**

KEDA (Kubernetes-based Event Driven Autoscaling)는 이벤트 기반으로 파드를 자동 확장할 수 있도록 지원합니다. Azure Monitor를 통해 메트릭을 수집하고, KEDA를 사용하여 이러한 메트릭을 기반으로 자동 확장을 설정할 수 있습니다.

* **설정 방법**:
  1. KEDA를 AKS 클러스터에 설치합니다.

bash

코드 복사

kubectl apply -f https://github.com/kedacore/keda/releases/latest/download/keda-2.0.0.yaml

* 1. ScaledObject 리소스를 정의하여 특정 이벤트에 따라 파드를 자동 확장합니다. 예를 들어, Azure Monitor 메트릭을 사용하는 경우 다음과 같이 설정할 수 있습니다.

yaml

코드 복사

apiVersion: keda.sh/v1alpha1

kind: ScaledObject

metadata:

name: myapp-scaledobject

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: myapp

minReplicaCount: 1

maxReplicaCount: 10

triggers:

- type: azure-monitor

metadata:

resourceURI: Microsoft.Insights/components/my-app-insights

tenantID: <your-tenant-id>

clientID: <your-client-id>

clientSecret: <your-client-secret>

subscriptionID: <your-subscription-id>

resourceGroupName: <your-resource-group>

metricName: requests/count

threshold: '100'

* **동작 방식**:
  1. 지정된 메트릭(예: Azure Monitor에서 수집된 요청 수)이 임계값을 초과하면 KEDA가 파드 수를 증가시켜 확장(Scale Out)합니다.
  2. 메트릭 값이 임계값 이하로 떨어지면 KEDA가 파드 수를 줄여 축소(Scale In)합니다.

**기대 효과 및 고려 사항**

* **기대 효과**:
  + **유연한 자원 관리**: 사용량에 따라 자동으로 자원을 조절하여 최적의 성능을 유지합니다.
  + **비용 절감**: 필요할 때만 자원을 추가로 사용함으로써 비용을 절감할 수 있습니다.
  + **향상된 성능**: 트래픽 급증 시에도 시스템이 원활하게 동작할 수 있도록 확장성을 보장합니다.
* **고려 사항**:
  + **설정 복잡성**: 자동 확장을 설정하고 관리하는 과정이 복잡할 수 있습니다.
  + **모니터링 필요**: 자동 확장 설정이 제대로 작동하는지 지속적으로 모니터링해야 합니다.
  + **리소스 한계**: 클라우드 제공자의 리소스 한계 및 요금 구조를 고려해야 합니다.
  + **적절한 임계값 설정**: 확장 임계값을 적절히 설정하지 않으면 불필요한 리소스 사용 또는 성능 저하가 발생할 수 있습니다.

이와 같은 방식으로 AKS의 확장성을 구현함으로써 효율적인 자원 관리를 할 수 있으며, 필요에 따라 빠르게 자원을 확장하거나 축소할 수 있습니다.

AKS의 Scale In(노드 축소) 과정에서 발생할 수 있는 잠재적 장애와 이를 방지하거나 해결하는 방법을 아래에 설명하겠습니다.

**잠재적 장애**

1. **파드 종료로 인한 서비스 중단**
   * **원인**: 노드 축소 과정에서 해당 노드에 배포된 파드가 종료될 수 있습니다. 이 경우 서비스에 영향을 줄 수 있습니다.
   * **방지 방법**: 중요한 파드에는 Pod Disruption Budget(PDB)을 설정하여 특정 수 이상의 파드가 동시에 종료되지 않도록 합니다.
2. **상태 저장 파드의 데이터 손실**
   * **원인**: 상태 저장 파드(StatefulSets, Persistent Volumes 등)가 있는 노드가 축소될 때, 데이터 손실이 발생할 수 있습니다.
   * **방지 방법**: StatefulSet과 Persistent Volumes을 사용하여 데이터가 안전하게 저장되도록 하며, 해당 데이터를 다른 노드로 미리 마이그레이션합니다.
3. **로드 밸런서 재구성 문제**
   * **원인**: 노드 축소 시 로드 밸런서가 새로운 노드 구성에 맞게 업데이트되지 않아 트래픽 분배에 문제가 발생할 수 있습니다.
   * **방지 방법**: 로드 밸런서 설정이 자동으로 업데이트되도록 확인하고, 업데이트가 누락되지 않도록 모니터링합니다.
4. **노드 드레인(Drain) 실패**
   * **원인**: 노드를 축소하는 과정에서 파드가 다른 노드로 제대로 이동하지 못하면 드레인 실패가 발생할 수 있습니다.
   * **방지 방법**: 노드 드레인 시 kubectl drain 명령을 사용하여 파드를 안전하게 축소합니다. 이 과정에서 --ignore-daemonsets 및 --delete-local-data 옵션을 사용할 수 있습니다.

bash

코드 복사

kubectl drain <node-name> --ignore-daemonsets --delete-local-data

1. **리소스 불균형**
   * **원인**: 특정 노드에 리소스가 집중되면 노드 축소 시 자원 불균형이 발생할 수 있습니다.
   * **방지 방법**: 클러스터 오토스케일러 설정을 통해 리소스가 고르게 분배되도록 하고, 노드 축소 전에 리소스 분포를 점검합니다.
2. **네트워크 세션 중단**
   * **원인**: 노드 축소로 인해 네트워크 세션이 중단될 수 있습니다.
   * **방지 방법**: 세션 유지를 위해 클라이언트 측 재시도 메커니즘을 구현하거나, 노드 축소 전에 세션을 적절히 마이그레이션합니다.

**장애 방지 및 해결 방법**

1. **Pod Disruption Budget (PDB) 설정**
   * 특정 수 이상의 파드가 동시에 종료되지 않도록 제한하여 서비스 중단을 방지합니다.

yaml

코드 복사

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: myapp-pdb

spec:

minAvailable: 3

selector:

matchLabels:

app: myapp

1. **상태 저장 파드의 데이터 백업 및 마이그레이션**
   * 데이터가 안전하게 저장되도록 StatefulSet 및 Persistent Volumes을 사용합니다.
   * 노드 축소 전 데이터를 다른 노드로 미리 마이그레이션합니다.
2. **로드 밸런서 설정 확인**
   * 노드 축소 시 로드 밸런서가 새로운 노드 구성에 맞게 자동으로 업데이트되는지 확인합니다.
   * 주기적으로 로드 밸런서 설정을 점검하고 업데이트가 누락되지 않도록 모니터링합니다.
3. **노드 드레인 전략**
   * 노드 드레인 시 kubectl drain 명령을 사용하여 파드를 안전하게 이동합니다.
   * 드레인 과정에서 발생할 수 있는 문제를 최소화하기 위해 충분한 리소스가 있는지 확인합니다.
4. **리소스 분포 점검**
   * 클러스터 오토스케일러 설정을 통해 리소스가 고르게 분배되도록 합니다.
   * 노드 축소 전에 리소스 분포를 점검하고, 자원 불균형을 사전에 해결합니다.
5. **네트워크 세션 관리**
   * 클라이언트 측에서 네트워크 세션 유지 및 재시도 메커니즘을 구현합니다.
   * 노드 축소 전에 세션을 적절히 마이그레이션하여 중단을 최소화합니다.

**요약**

AKS의 Scale In 과정에서 발생할 수 있는 잠재적 장애를 방지하고, 문제 발생 시 신속하게 해결하기 위해 사전 준비와 모니터링이 중요합니다. 위에서 설명한 방법들을 통해 안정적인 노드 축소와 서비스 운영을 보장할 수 있습니다.

AKS의 Scale In 과정에서 웹 서비스에서 504 Gateway Timeout 오류가 발생할 수 있는 몇 가지 일반적인 원인과 이를 해결하는 방법은 다음과 같습니다.

**원인**

1. **파드 종료로 인한 요청 처리 실패**
   * **설명**: Scale In 과정에서 웹 서비스 파드가 종료되면서 해당 파드가 처리 중인 요청이 실패할 수 있습니다. 이로 인해 클라이언트는 504 오류를 받게 됩니다.
   * **해결 방법**: 중요한 파드에 대해 Pod Disruption Budget (PDB)을 설정하여 동시에 너무 많은 파드가 종료되지 않도록 합니다.
2. **로드 밸런서 업데이트 지연**
   * **설명**: 노드 축소 시 로드 밸런서가 새로운 노드 및 파드 상태를 반영하는 데 지연이 발생할 수 있습니다. 이로 인해 트래픽이 아직 존재하지 않는 파드로 라우팅되어 504 오류가 발생할 수 있습니다.
   * **해결 방법**: 로드 밸런서 설정 및 업데이트가 적시에 반영되도록 설정을 확인하고, 필요 시 로드 밸런서의 헬스 체크 주기를 짧게 조정합니다.
3. **네트워크 세션 중단**
   * **설명**: Scale In 과정에서 노드가 축소되면서 기존 네트워크 세션이 중단될 수 있습니다. 이로 인해 클라이언트 요청이 제대로 처리되지 않아 504 오류가 발생할 수 있습니다.
   * **해결 방법**: 클라이언트 측에 재시도 로직을 구현하고, 세션 유지를 위한 네트워크 설정을 조정합니다.
4. **노드 드레인 과정의 문제**
   * **설명**: 노드가 축소되는 과정에서 파드가 적절히 드레인되지 않으면, 파드가 새로운 노드로 이동하기 전에 트래픽을 처리하지 못하게 되어 504 오류가 발생할 수 있습니다.
   * **해결 방법**: 노드 드레인 시 kubectl drain 명령을 사용하여 파드를 안전하게 이동하고, 필요한 경우 --grace-period 옵션을 사용하여 파드가 안전하게 종료될 시간을 충분히 부여합니다.

bash

코드 복사

kubectl drain <node-name> --ignore-daemonsets --delete-local-data --grace-period=<seconds>

1. **리소스 부족**
   * **설명**: Scale In으로 인해 남은 노드에 리소스 부족이 발생할 수 있으며, 이로 인해 파드가 적절히 스케줄링되지 않아 요청을 처리하지 못할 수 있습니다.
   * **해결 방법**: Scale In 전에 충분한 리소스가 있는지 확인하고, HPA 설정을 통해 필요 시 파드 수를 자동으로 조정합니다.

**해결 방안**

1. **Pod Disruption Budget (PDB) 설정**
   * 동시에 너무 많은 파드가 종료되지 않도록 PDB를 설정합니다.

yaml

코드 복사

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: webapp-pdb

spec:

minAvailable: 3

selector:

matchLabels:

app: webapp

1. **로드 밸런서 헬스 체크 주기 조정**
   * 로드 밸런서의 헬스 체크 주기를 짧게 조정하여 상태 변화가 빠르게 반영되도록 합니다.

yaml

코드 복사

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: webapp-ingress

spec:

rules:

- host: example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: webapp-service

port:

number: 80

1. **노드 드레인 명령 사용**
   * kubectl drain 명령을 사용하여 파드를 안전하게 이동하고, --grace-period 옵션을 사용하여 파드가 안전하게 종료될 시간을 충분히 부여합니다.

bash

코드 복사

kubectl drain <node-name> --ignore-daemonsets --delete-local-data --grace-period=<seconds>

1. **리소스 모니터링 및 자동 확장 설정**
   * 클러스터의 리소스를 지속적으로 모니터링하고, HPA 설정을 통해 필요 시 파드 수를 자동으로 조정합니다.

yaml

코드 복사

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: webapp-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: webapp

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

1. **클라이언트 측 재시도 로직 구현**
   * 클라이언트 애플리케이션에서 재시도 로직을 구현하여 네트워크 세션 중단 시 자동으로 요청을 재시도하도록 합니다.

**결론**

AKS의 Scale In 과정에서 발생하는 504 Gateway Timeout 오류는 주로 파드 종료, 로드 밸런서 업데이트 지연, 네트워크 세션 중단 등의 원인으로 발생할 수 있습니다. 이를 방지하기 위해 PDB 설정, 로드 밸런서 설정 조정, 노드 드레인 명령 사용, 리소스 모니터링 및 자동 확장 설정, 클라이언트 측 재시도 로직 구현 등의 해결 방안을 적용할 수 있습니다. 이를 통해 서비스 중단 없이 안정적인 웹 서비스를 제공할 수 있습니다.

AKS 구성을 통해 504 오류가 발생하지 않도록 하기 위한 설계 방안을 제시합니다. 이 설계는 높은 가용성, 안정성, 그리고 원활한 서비스 운영을 목표로 합니다.

**설계 방안**

**1. 고가용성 클러스터 설정**

* **노드 풀**: AKS 클러스터에서 여러 노드 풀을 사용하여 노드를 분산합니다. 이로 인해 한 노드 풀의 문제가 전체 서비스에 영향을 미치지 않도록 합니다.
* **Availability Zones**: Azure의 Availability Zones을 사용하여 노드를 여러 가용성 영역에 분산합니다.

**2. Pod Disruption Budget (PDB) 설정**

* **설정 방법**: 중요한 애플리케이션에 대해 PDB를 설정하여 동시에 종료되는 파드 수를 제한합니다.

yaml

코드 복사

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: webapp-pdb

spec:

minAvailable: 3

selector:

matchLabels:

app: webapp

**3. Horizontal Pod Autoscaler (HPA) 사용**

* **설정 방법**: HPA를 설정하여 애플리케이션의 파드 수를 자동으로 조절합니다.

yaml

코드 복사

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: webapp-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: webapp

minReplicas: 3

maxReplicas: 20

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

**4. 로드 밸런서 및 네트워크 설정 최적화**

* **헬스 체크 주기 조정**: 로드 밸런서의 헬스 체크 주기를 짧게 설정하여 상태 변화를 빠르게 반영합니다.

yaml

코드 복사

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: webapp-ingress

spec:

rules:

- host: example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: webapp-service

port:

number: 80

**5. 노드 드레인 전략 적용**

* **설정 방법**: 노드 축소 시 kubectl drain 명령을 사용하여 파드를 안전하게 이동합니다.

bash

코드 복사

kubectl drain <node-name> --ignore-daemonsets --delete-local-data --grace-period=<seconds>

**6. 리소스 모니터링 및 관리**

* **Azure Monitor 및 Log Analytics**: Azure Monitor 및 Log Analytics를 통해 클러스터와 애플리케이션의 성능 및 상태를 지속적으로 모니터링합니다.
* **Alerts 설정**: 특정 임계값을 초과할 경우 알림을 설정하여 신속하게 대응할 수 있도록 합니다.

**7. 클라이언트 측 재시도 로직 구현**

* **설정 방법**: 클라이언트 애플리케이션에 재시도 로직을 구현하여 네트워크 세션이 중단되었을 때 자동으로 요청을 재시도하도록 합니다.

**종합 설계 예시**

yaml

코드 복사

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp

spec:

replicas: 3

selector:

matchLabels:

app: webapp

template:

metadata:

labels:

app: webapp

spec:

containers:

- name: webapp

image: <image>

resources:

requests:

cpu: "100m"

memory: "128Mi"

limits:

cpu: "500m"

memory: "512Mi"

---

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: webapp-pdb

spec:

minAvailable: 3

selector:

matchLabels:

app: webapp

---

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: webapp-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: webapp

minReplicas: 3

maxReplicas: 20

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: webapp-ingress

spec:

rules:

- host: example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: webapp-service

port:

number: 80

**기대 효과**

1. **높은 가용성 보장**: 여러 노드 풀과 가용성 영역을 사용하여 단일 장애 지점을 제거.
2. **무중단 서비스 제공**: PDB와 HPA를 사용하여 서비스가 중단되지 않도록 보장.
3. **효율적인 리소스 사용**: 자동 확장을 통해 필요할 때만 자원을 추가로 사용하여 비용 절감.
4. **신속한 문제 대응**: Azure Monitor 및 Log Analytics를 통한 지속적인 모니터링과 알림 설정으로 신속한 문제 대응.

**고려 사항**

1. **복잡한 설정**: 다중 노드 풀과 가용성 영역 설정, PDB, HPA 설정 등의 초기 설정이 복잡할 수 있음.
2. **비용 관리**: 자동 확장으로 인한 비용 증가를 모니터링하고 관리 필요.
3. **지속적인 모니터링**: 클러스터와 애플리케이션의 상태를 지속적으로 모니터링하고 조정 필요.

이 설계를 통해 AKS에서 504 오류를 최소화하고 안정적인 웹 서비스를 제공할 수 있습니다.