Deploying Language Models on Azure Kubernetes: A Complete Beginner's Guide

PRASANNA KUMAR V

NOV 10, 2024

2

Sha

Deploying Language Models on Azure Kubernetes Service (AKS)

A Detailed Step-by-Step Implementation Guide

Introduction

This comprehensive guide explains how to deploy Large Language Models (LLMs) o Azure Kubernetes Service using the vLLM serving engine. Each step is broken dow with detailed explanations of why it's necessary and how it contributes to the overa deployment.

Table of Contents

Prerequisites

Azure Infrastructure Requirements

1. **Azure Subscription**
    - **What**: An active Azure subscription with billing enabled

- **Why**: Required for creating and managing Azure resources
- **How to verify**:

```
az account show
```

1. **Azure CLI**
   - **What**: Command-line tool for managing Azure resources
   - **Why**: Enables automated resource creation and management
   - **Installation**:

```
# For Ubuntu/Debian
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# For macOS
brew install azure-cli
```

1. **Kubernetes Tools**
   - **What**: kubectl and related tools
   - **Why**: Required for interacting with Kubernetes clusters
   - **Installation**:

```
# Install kubectl
az aks install-cli
```

Required Permissions

1. **Azure Permissions**

- Contributor role or higher on subscription/resource group
- Network Contributor for virtual network configuration
- Why: Enables creation and management of all required resources

2. **Hugging Face Account (for gated models)**
   - Account with approved access to gated models
   - Access token with read permissions
   - Why: Required for downloading and using gated models like Llama

Infrastructure Setup

1. Environment Preparation

```
# Set environment variables
export MY_RESOURCE_GROUP_NAME="llm-deployment-rg"
export MY_AKS_CLUSTER_NAME="llm-cluster"
export LOCATION="eastus"
```

Why these variables?

- Resource group name: Logical container for related resources
- Cluster name: Unique identifier for your AKS cluster
- Location: Determines data center location (choose based on latency requireme

2. Resource Group Creation

```
az group create \
  --name $MY_RESOURCE_GROUP_NAME \
  --location $LOCATION
```

Purpose:

- Creates a logical container for all deployment resources
- Enables easier resource management and billing tracking
- Allows for bulk operations and access control

3. AKS Cluster Creation

```
az aks create \
  --resource-group $MY_RESOURCE_GROUP_NAME \
  --name $MY_AKS_CLUSTER_NAME \
  --node-count 1 \
  --generate-ssh-keys \
  --network-plugin azure \
  --network-policy azure
```

Key Configuration Explained:

- `node-count`: Initial number of nodes (start small, scale as needed)
- `generate-ssh-keys`: Automatic SSH key generation for node access
- `network-plugin`: Azure CNI for advanced networking features
- `network-policy`: Enables network policy enforcement

4. Node Pool Configuration

System Node Pool

```
az aks nodepool add \
  --resource-group $MY_RESOURCE_GROUP_NAME \
  --cluster-name $MY_AKS_CLUSTER_NAME \
  --name system \
```

```
  --node-count 3 \
  --node-vm-size D2s_v3
```

Why these specifications?

- `node-count: 3`: Provides high availability for system components
- `D2s_v3`: Balanced CPU/memory for system services
- Dedicated pool for system components ensures stability

GPU Node Pool

```
az aks nodepool add \
  --resource-group $MY_RESOURCE_GROUP_NAME \
  --cluster-name $MY_AKS_CLUSTER_NAME \
  --name gpunp \
  --node-count 1 \
  --node-vm-size Standard_NC4as_T4_v3 \
  --node-taints sku=gpu:NoSchedule \
  --enable-cluster-autoscaler \
  --min-count 1 \
  --max-count 3
```

Configuration Details:

- `Standard_NC4as_T4_v3`: T4 GPU for optimal LLM inference
- `node-taints`: Ensures only GPU workloads run on these expensive nodes
- `enable-cluster-autoscaler`: Automatic scaling based on demand
- `min-count/max-count`: Scaling boundaries for cost control

5. NVIDIA Device Plugin Installation

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nvidia-device-plugin-daemonset
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: nvidia-device-plugin-ds
  template:
    metadata:
      labels:
        name: nvidia-device-plugin-ds
    spec:
      tolerations:
      - key: "sku"
        operator: "Equal"
        value: "gpu"
        effect: "NoSchedule"
      priorityClassName: "system-node-critical"
      containers:
      - image: nvcr.io/nvidia/k8s-device-plugin:v0.14.0
        name: nvidia-device-plugin-ctr
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
        volumeMounts:
        - name: device-plugin
          mountPath: /var/lib/kubelet/device-plugins
```

Why each component matters:

- `DaemonSet`: Ensures plugin runs on all GPU nodes
- `tolerations`: Allows running on GPU-tainted nodes
- `priorityClassName`: Ensures plugin isn't evicted
- `securityContext`: Implements security best practices

Model Deployment Configuration

1. Persistent Volume Setup

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mistral-7b
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: default
```

Purpose of each setting:

- `50Gi storage`: Accommodates model weights and cache
- `ReadWriteOnce`: Single node access for data consistency
- `default` storage class: Uses Azure managed disks

2. Service Configuration

```
apiVersion: v1
kind: Service
metadata:
  name: mistral-7b
  namespace: default
spec:
  ports:
  - name: http-mistral-7b
    port: 80
```

```
    targetPort: 8000
  selector:
    app: mistral-7b
  type: LoadBalancer
```

Key components explained:

- `LoadBalancer` type: Provides external access
- Port mapping: Routes external port 80 to container port 8000
- Selector: Links service to specific deployment

3. Deployment Configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mistral-7b
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: mistral-7b
        image: vllm/vllm-openai:latest
        resources:
          limits:
            nvidia.com/gpu: 1
            memory: 20G
          requests:
            nvidia.com/gpu: 1
            memory: 6G
        volumeMounts:
        - mountPath: /root/.cache/huggingface
          name: cache-volume
```

Configuration details:

- `replicas: 1`: Single instance per GPU
- Resource limits: Prevents memory issues
- Volume mounts: Persists model cache
- Health probes: Ensures container health

Deployment Process

1. Basic Deployment

```
# Create namespace
kubectl create namespace llm-serving

# Apply configurations
kubectl apply -f volume.yaml
kubectl apply -f service.yaml
kubectl apply -f deployment.yaml
```

2. Verify Deployment

```
# Check pod status
kubectl get pods -n llm-serving
kubectl describe pod <pod-name>

# Verify service
kubectl get service mistral-7b
```

Testing and Validation

1. API Testing

```
# Get external IP
export SERVICE_IP=$(kubectl get service mistral-7b -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')

# Test API
curl --location "http://$SERVICE_IP/v1/completions" \
--header 'Content-Type: application/json' \
--data '{
    "model": "mistralai/Mistral-7B-Instruct-v0.1",
    "prompt": "Test prompt",
    "max_tokens": 50
}'
```

2. Performance Testing

```
# Monitor GPU usage
kubectl exec -it <pod-name> -- nvidia-smi

# Check response times
time curl -X POST "http://$SERVICE_IP/v1/completions" ...
```

Production Considerations

Security Implementation

1. **Network Security**

```
# Create network policy
kubectl apply -f network-policy.yaml

# Enable Azure DDoS protection
az network ddos-protection enable ...
```

1. **API Security**
    - Implement authentication
    - Set up rate limiting
    - Enable monitoring

Cost Optimization

1. **Resource Monitoring**

```
# Monitor costs
az cost management query ...

# Scale based on usage
kubectl scale deployment mistral-7b --replicas=0
```

1. **Cost Reduction Strategies**
    - Use spot instances for non-critical workloads
    - Implement automatic scaling
    - Monitor and optimize resource usage

Maintenance Procedures

1. Regular Updates

```
# Update deployment
kubectl set image deployment/mistral-7b mistral-7b=vllm/vllm-openai:new-versi

# Verify update
kubectl rollout status deployment/mistral-7b
```

2. Backup and Recovery

```
# Backup persistent volumes
velero backup create llm-backup

# Restore if needed
velero restore create --from-backup llm-backup
```

Troubleshooting Guide

Common Issues and Solutions

1. **GPU Not Detected**
   - Verify NVIDIA plugin installation
   - Check node labels and taints
   - Validate GPU driver installation
2. **Memory Issues**
   - Adjust resource limits
   - Monitor memory usage
   - Check for memory leaks
3. **Network Issues**
   - Verify network policy configuration
   - Check service endpoint availability
   - Validate load balancer configuration

Monitoring Setup

1. Metrics Collection

```
# Install Prometheus
helm install prometheus prometheus-community/prometheus

# Configure Grafana
helm install grafana grafana/grafana
```

2. Log Management

```
# Enable log analytics
az monitor log-analytics workspace create ...

# Configure container insights
az aks enable-addons -a monitoring ...
```

Additional Resources and References

1. **Documentation**
   - Azure Kubernetes Service
   - vLLM Documentation
   - Hugging Face Models
2. **Community Support**
   - Azure Kubernetes Service GitHub
   - vLLM Discord community
   - Hugging Face forums

2 Likes

Next

Discussion about this post

Comments Restacks

Write a comment...