deployment blog tutorial: https://huggingface.co/blog/vpkprasanna/deploying-language-models-on-azure

Okay, let's break down the process of deploying the `google/gemma-3-12b-it` model on Azure Kubernetes Service (AKS) using vLLM, based on the provided article but tailored specifically for this model.

Key Differences from the Article:

1. **Model:** We're using `google/gemma-3-12b-it` instead of `mistralai/Mistral-7B-Instruct-v0.1`.
2. **GPU Requirements:** Gemma 12B requires significantly more GPU memory than Mistral 7B. A T4 GPU (16GB VRAM) is **not sufficient**. We'll need a VM with more VRAM, like an NVIDIA A100 (40GB or 80GB). We'll use an A100 80GB VM size in this guide (`Standard_NC48ads_A100_v4`) for better performance and to avoid memory issues, but be aware this is more expensive.
3. **Disk Space:** The Gemma 12B model is larger, so we'll increase the Persistent Volume size.
4. **Hugging Face Token:** Gemma models often require you to accept terms and might need authentication. We'll include steps to use a Hugging Face token.
5. **Resource Names:** We'll change names like `mistral-7b` to `gemma-12b` for clarity.
6. **vLLM Configuration:** We need to explicitly tell the vLLM container which model to load via command-line arguments.

Here is the step-by-step guide:

Phase 1: Prerequisites

- **Ensure all prerequisites from the article are met:**
  - Active Azure Subscription
  - Azure CLI installed and configured (`az login`)
  - `kubectl` installed
  - Required Azure permissions (Contributor role recommended)
  - Hugging Face Account:
    - Go to the `google/gemma-3-12b-it` model page on Hugging Face: https://huggingface.co/google/gemma-3-12b-it
    - Accept any license terms or usage conditions if prompted.
    - Generate a Hugging Face Access Token with `read` permissions from your Hugging Face account settings (https://huggingface.co/settings/tokens). **Copy this token securely.** You'll need it later.

Phase 2: Infrastructure Setup (Azure)

1. **Set Environment Variables:** Open your terminal (Bash, zsh, etc.).

```
export MY_RESOURCE_GROUP_NAME="gemma-12b-deployment-rg"
export MY_AKS_CLUSTER_NAME="gemma-12b-cluster"
export LOCATION="centralindia" # Or choose a region suitable for
you with A100 GPU availability
export MY_GPU_NODEPOOL_NAME="gpunpa100" # Name for the GPU node
pool
export MY_GPU_VM_SIZE="Standard_NC48ads_A100_v4" # VM with 2x A100
80GB GPU. Check availability in your LOCATION!
export K8S_NAMESPACE="llm-gemma" # Kubernetes namespace for
deployment
export HUGGING_FACE_TOKEN="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"
# Paste your token here (or load from a secure place)
```

- **Important:** Verify that the chosen `MY_GPU_VM_SIZE` is available in your selected `LOCATION`. A100 VMs have limited regional availability. Check the Azure documentation for "GPU optimized virtual machine sizes". These VMs are also significantly more expensive than T4 VMs.

2. **Create Resource Group:**

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

3. **Create AKS Cluster (Initial):** Create the cluster initially with a system node pool. We'll add the GPU pool separately.

```
az aks create \
  --resource-group $MY_RESOURCE_GROUP_NAME \
  --name $MY_AKS_CLUSTER_NAME \
  --node-count 1 \
  --node-vm-size Standard_D2s_v3 \ # Small size for system pool
  --generate-ssh-keys \
  --network-plugin azure \
  --network-policy azure \
  --location $LOCATION
```

(This might take several minutes)

4. **Get AKS Credentials:**

```
az aks get-credentials --resource-group $MY_RESOURCE_GROUP_NAME --
name $MY_AKS_CLUSTER_NAME
```

Verify connection: `kubectl get nodes`

5. **Add System Node Pool (Optional but Recommended, I haven't done this):** The `az aks create` command already created a default node pool which can serve as the system pool. If you want a dedicated one (as in the article):

```
# If you want a separate system pool (adjust name/count if needed)
# az aks nodepool add \
#   --resource-group $MY_RESOURCE_GROUP_NAME \
#   --cluster-name $MY_AKS_CLUSTER_NAME \
#   --name systempool \
#   --node-count 1 \
#   --node-vm-size Standard_D2s_v3

# If using the default pool created during `az aks create`, rename
it (optional)
# az aks nodepool update --resource-group $MY_RESOURCE_GROUP_NAME
--cluster-name $MY_AKS_CLUSTER_NAME --name nodepool1 --mode System
```

For simplicity, we'll assume the initial node pool acts as the system pool.

6. **Add GPU Node Pool:** This is where we add the powerful (and expensive) A100 node.

```
az aks nodepool add \
  --resource-group $MY_RESOURCE_GROUP_NAME \
  --cluster-name $MY_AKS_CLUSTER_NAME \
  --name $MY_GPU_NODEPOOL_NAME \
  --node-count 1 \
  --node-vm-size $MY_GPU_VM_SIZE \
  --node-taints sku=gpu:NoSchedule \
  --enable-cluster-autoscaler \
  --min-count 1 \
  --max-count 3 # Adjust max count based on budget/needs
```

   - `--node-taints sku=gpu:NoSchedule`: Prevents non-GPU pods from being scheduled on these expensive nodes.
   - `--enable-cluster-autoscaler`: Allows scaling the GPU pool based on demand (defined by pending pods requesting GPU resources).

7. **Install NVIDIA Device Plugin:** This DaemonSet allows Kubernetes to recognize and assign the NVIDIA GPUs on the nodes. Create a file named `nvidia-plugin.yaml` with the content provided in the article:

```
# nvidia-plugin.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nvidia-device-plugin-daemonset
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: nvidia-device-plugin-ds
  updateStrategy:
```

```
    type: RollingUpdate
  template:
    metadata:
      labels:
        name: nvidia-device-plugin-ds
    spec:
      tolerations:
      # This toleration allows scheduling the pod on nodes marked
with the NVIDIA taint
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
      # The following toleration is needed IF your GPU nodes have
the taint defined in step 6
      - key: "sku"
        operator: "Equal"
        value: "gpu"
        effect: "NoSchedule"
      priorityClassName: system-node-critical
      containers:
      - image: nvcr.io/nvidia/k8s-device-plugin:v0.14.0 # Use a
recent version if needed
        name: nvidia-device-plugin-ctr
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
        volumeMounts:
          - name: device-plugin
            mountPath: /var/lib/kubelet/device-plugins
      volumes:
        - name: device-plugin
          hostPath:
            path: /var/lib/kubelet/device-plugins
```

Apply it:

```
kubectl apply -f nvidia-plugin.yaml
```

Verify the pod is running on the GPU node: `kubectl get pods -n kube-system -l name=nvidia-device-plugin-ds -o wide`

Phase 3: Kubernetes Configuration (Model Deployment)

1. **Create Namespace:**

```
kubectl create namespace $K8S_NAMESPACE
```

2. **Create Hugging Face Token Secret:** This securely stores your Hugging Face token within Kubernetes.

```
kubectl create secret generic huggingface-secret \
  --from-literal=token=$HUGGING_FACE_TOKEN \
  --namespace $K8S_NAMESPACE
```

3. **Create Persistent Volume Claim (PVC):** This reserves storage for downloading model weights and caching. Create a file named `pvc-gemma.yaml`:

```
# pvc-gemma.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gemma-12b-pvc
  namespace: llm-gemma # Deploy PVC to default namespace or
$K8S_NAMESPACE
spec:
  accessModes:
    - ReadWriteOnce # Can only be mounted by one node at a time
  resources:
    requests:
      storage: 100Gi # Increased size for Gemma 12B
  storageClassName: managed-csi # Standard Azure managed disk CSI
driver
```

   - **Namespace:** The article puts the PVC in `default`. You can put it in `$K8S_NAMESPACE` too, just ensure the Deployment refers to it correctly. Let's keep it in `default` as per the article's structure for now. Apply it:

```
kubectl apply -f pvc-gemma.yaml -n default
```

4. **Create Service:** This exposes the vLLM deployment via an Azure Load Balancer. Create a file named `service-gemma.yaml`:

```
# service-gemma.yaml
apiVersion: v1
kind: Service
metadata:
  name: gemma-12b-service # Changed name
  namespace: llm-gemma # Deploy service in our namespace
spec:
  ports:
  - name: http-llm # Changed name
    port: 80 # External port
    targetPort: 8000 # Port the vLLM container listens on
  selector:
    app: gemma-12b # Matches labels in the Deployment
```

```
  type: LoadBalancer # Creates an Azure Load Balancer with a
public IP
```

Apply it:

```
kubectl apply -f service-gemma.yaml -n $K8S_NAMESPACE
```

5. **Create Deployment:** This defines how to run the vLLM container with the Gemma model. Create a file named `deployment-gemma.yaml`:

```
# deployment-gemma.yaml (showing relevant container section)
apiVersion: apps/v1
kind: Deployment
metadata:
name: gemma-12b-deployment
namespace: llm-gemma
spec:
strategy: # <-- Add this
    type: Recreate # <-- Add this
replicas: 1 # Still running one pod, but this pod will use 2 GPUs
selector:
    matchLabels:
    app: gemma-12b
template:
    metadata:
    labels:
        app: gemma-12b
    spec:
    nodeSelector:
        agentpool: gpunpa100 # Make sure this matches your GPU
pool name
    tolerations:
    - key: "sku"
        operator: "Equal"
        value: "gpu"
        effect: "NoSchedule"
    volumes:
    - name: model-cache
        persistentVolumeClaim:
        claimName: gemma-12b-pvc
    - name: dshm
        emptyDir:
        medium: Memory
        sizeLimit: 32Gi # Increased shared memory slightly, might
help with TP communication
    containers:
    - name: vllm-gemma-12b
        image: vllm/vllm-openai:latest
```

```
        args: # <-- Modify Args
        - "--model"
        - "google/gemma-3-12b-it"
        - "--max-model-len"
        - "8192"
        - "--trust-remote-code"
        # --- ADD THIS LINE ---
        - "--tensor-parallel-size"
        - "2" # Use 2 GPUs for tensor parallelism
        # Optional: Adjust GPU memory utilization if needed,
default is 0.9
        # - "--gpu-memory-utilization"
        # - "0.95"
        ports:
        - containerPort: 8000
        env:
        - name: HUGGING_FACE_HUB_TOKEN
        valueFrom:
            secretKeyRef:
            name: huggingface-secret
            key: token
        resources: # <-- Modify Resources
        requests:
            # --- CHANGE THIS LINE ---
            nvidia.com/gpu: 2 # Request 2 GPUs
            memory: "64Gi" # Increased system RAM request slightly
        limits:
            # --- CHANGE THIS LINE ---
            nvidia.com/gpu: 2 # Limit to 2 GPUs
            memory: "200Gi" # Ensure ample system RAM limit
        volumeMounts:
        - name: model-cache
        mountPath: /root/.cache/huggingface
        - name: dshm
        mountPath: /dev/shm
        readinessProbe:
        tcpSocket:
            port: 8000
        initialDelaySeconds: 90 # May need slightly longer startup
with TP
        periodSeconds: 10
        timeoutSeconds: 5
        failureThreshold: 9 # Allow more failures during init
        livenessProbe:
        tcpSocket:
            port: 8000
        initialDelaySeconds: 240
```

```
        periodSeconds: 20
        timeoutSeconds: 5
        failureThreshold: 3
```

Important Notes on Deployment YAML:

- `nodeSelector`: Ensures the pod runs *only* on the specified GPU node pool (`gpunpa100`).
- `tolerations`: Allows the pod to be scheduled on nodes with the `sku=gpu:NoSchedule` taint.
- `volumes` & `volumeMounts`: Connects the PVC (`gemma-12b-pvc`) to the container's cache directory `/root/.cache/huggingface`. Also mounts `/dev/shm` for potential performance gains.
- `args`: Tells vLLM which model (`google/gemma-3-12b-it`) to load and sets the max sequence length. `--trust-remote-code` is often required.
- `env`: Securely passes the Hugging Face token from the Kubernetes secret to the container.
- `resources`: Requests and limits 1 GPU and specifies system memory (RAM, not VRAM). Adjust memory based on your VM size and observation.
- `readinessProbe` & `livenessProbe`: Help Kubernetes manage the pod's health. The initial delays are long to allow for model download and loading.

Apply it:

```
# Replace $MY_GPU_NODEPOOL_NAME in deployment-gemma.yaml first if
needed
# For example, using sed:
# sed -i "s/\$MY_GPU_NODEPOOL_NAME/$MY_GPU_NODEPOOL_NAME/g"
deployment-gemma.yaml
kubectl apply -f deployment-gemma.yaml -n $K8S_NAMESPACE
```

Phase 4: Deployment Process & Verification

1. **Check Pod Status:** The first time, the pod needs to download the Gemma 12B model (which is large) into the PVC. This can take a significant amount of time (10-30 minutes or more depending on network speed and disk performance).

   ```
   kubectl get pods -n $K8S_NAMESPACE -w # Watch status changes
   ```

   Wait until the pod status is `Running`. If it gets stuck in `ContainerCreating` or `Init`, check its events:

   ```
   kubectl describe pod <pod-name> -n $K8S_NAMESPACE
   ```

   Check logs for download progress or errors:

   ```
   kubectl logs <pod-name> -n $K8S_NAMESPACE -f
   ```

Look for messages from vLLM indicating model loading progress or any errors (like authentication issues, out-of-memory errors).

2. **Verify Service and Get External IP:** Once the pod is `Running`, check if the LoadBalancer service got an external IP address.

   ```
   kubectl get service gemma-12b-service -n $K8S_NAMESPACE -w
   ```

   Wait until `EXTERNAL-IP` changes from `<pending>` to an actual IP address. Copy this IP.

   ```
   export SERVICE_IP=$(kubectl get service gemma-12b-service -n
   $K8S_NAMESPACE -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
   echo "Service IP: $SERVICE_IP"
   ```

Phase 5: Testing and Validation

1. **API Testing (Chat Completions):** Use `curl` to send a request to the vLLM OpenAI-compatible endpoint. Since `gemma-3-12b-it` is an instruction-tuned model, use the chat completions endpoint (`/v1/chat/completions`).

   ```
   curl http://$SERVICE_IP/v1/chat/completions \
     -H "Content-Type: application/json" \
     -d '{
       "model": "google/gemma-3-12b-it",
       "messages": [
         {"role": "system", "content": "You are a helpful
   assistant."},
         {"role": "user", "content": "Explain the concept of
   Kubernetes in simple terms."}
       ],
       "max_tokens": 200,
       "temperature": 0.7
     }'
   ```

   You should receive a JSON response containing the model's answer.

2. **Performance Testing (Basic):**

   - **GPU Usage:** Check GPU utilization while sending requests.

     ```
     # Find the pod name
     POD_NAME=$(kubectl get pods -n $K8S_NAMESPACE -l app=gemma-12b
     -o jsonpath='{.items[0].metadata.name}')
     # Execute nvidia-smi in the pod
     kubectl exec -it $POD_NAME -n $K8S_NAMESPACE -- nvidia-smi
     ```

   - **Response Time:** Use `time` with the `curl` command to measure latency. Send multiple concurrent requests to check throughput (tools like `k6`, `locust`, or `hey` are

better for proper load testing).

Phase 6: Production Considerations, Maintenance, Monitoring

- **Follow the advice in the original article:**
  - **Security:** Implement Network Policies, consider Azure DDoS, add authentication/API keys (using an API Gateway in front of the service is common), set rate limits.
  - **Cost Optimization:** Monitor Azure costs closely (A100s are expensive!). Use the cluster autoscaler effectively. Consider Azure Spot Instances for the GPU nodes if workload tolerance allows (can lead to significant savings but requires handling potential interruptions). Scale down to `0` replicas when not in use: `kubectl scale deployment gemma-12b-deployment -n $K8S_NAMESPACE --replicas=0`.
  - **Maintenance:** Plan for regular updates (vLLM image, base OS, AKS version), implement backup/restore for the PVC (using Velero or Azure Disk snapshots).
  - **Monitoring:** Set up Prometheus/Grafana for metrics (GPU util, memory, request rates, latency) and Azure Monitor/Container Insights for logging and alerting.

This detailed guide provides the specific steps to deploy `google/gemma-3-12b-it` on AKS using vLLM, addressing the model's specific requirements like GPU VRAM, Hugging Face token, and vLLM configuration. Remember to manage costs carefully due to the high-end GPU requirement.