Azure Kubernetes Service

Introduction

Kubernetes is a rapidly evolving platform that manages container-based applications and their associated networking and storage components. Kubernetes focuses on the application workloads, not the underlying infrastructure components. Kubernetes provides a declarative approach to deployments, backed by a robust set of APIs for management operations.

You can build and run modern, portable, microservices-based applications, using Kubernetes to orchestrate and manage the availability of the application components. Kubernetes supports both stateless and stateful applications as teams progress through the adoption of microservices-based applications.

Containers:

In Kubernetes, containers refer to lightweight, standalone executable units of software that package up code and dependencies to run consistently across multiple computing environments. Containers allow developers to build, package, and deploy applications easily and quickly, without worrying about the underlying infrastructure.

Node:

A node refers to a worker machine in the cluster that runs containerized applications. Each node runs one or more containers managed by Kubernetes. A node can be a virtual or physical machine, depending on the configuration of the cluster.

Node pools

Nodes of the same configuration are grouped together into node pools. A Kubernetes cluster contains at least one node pool.

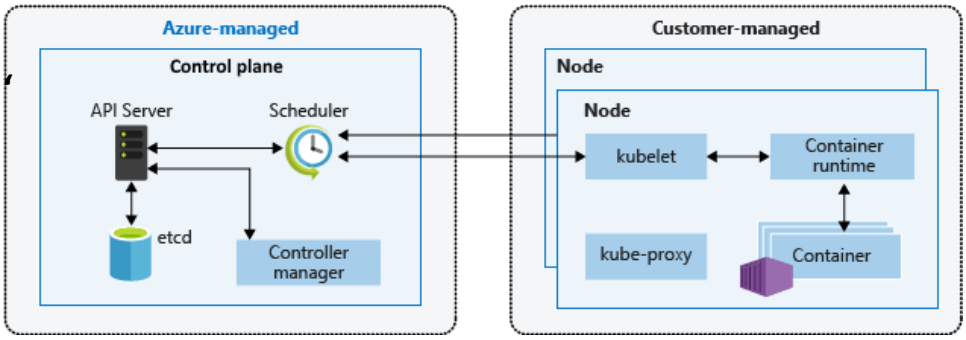
Cluster:

A cluster in Kubernetes is a group of nodes that work together to run containerized applications. A cluster consists of a control plane and one or more worker nodes. The control plane manages the cluster, while the worker nodes run the containerized applications.

A Kubernetes cluster is divided into two components:

Control plane: provides the core Kubernetes services and orchestration of application workloads.

Nodes: run your application workloads.



Pods:

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

Namespaces

Kubernetes resources, such as pods and deployments, are logically grouped into a namespace to divide an AKS cluster and restrict create, view, or manage access to resources. For example, you can create namespaces to separate business groups. Users can only interact with resources within their assigned namespaces.

Azure Kubernetes Service

Azure Kubernetes Service (AKS) simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance. When you create an AKS cluster, a control plane is automatically created and configured. This control plane is provided at no cost as a managed Azure resource abstracted from the user. You only pay for and manage the nodes attached to the AKS cluster.

AKS nodes run on Azure virtual machines (VMs). With AKS nodes, you can connect storage to nodes and pods, upgrade cluster components, and use GPUs. AKS supports Kubernetes clusters that run multiple node pools to support mixed operating systems and Windows Server containers.

As demand for resources change, the number of cluster nodes or pods that run your services automatically scales up or down. You can adjust both the horizontal pod autoscaler or the cluster autoscaler to adjust to demands and only run necessary resources.

Deploy Models to AKS

1. The first step is to connect to your workspace. You will need **a subscription ID, resource group name and the workspace name.**

   You can do this by using the following code

```
ws = Workspace(subscription_id="5a341dea-d53d-438c-914c-5405206db17d",
               resource_group="DefaultResourceGroup-centralindia",
               workspace_name="deploy-model")
```

2. The next step is to register your model. You can register a model from a **local file or from an Azure ML training Job.** We will register from a local file.

   You can do this by using the following code

```
model = Model.register(ws, model_name="sentiment_analysis", model_path="./sentiment_analysis_model.pkl")
```

3. Create an Environment Specification which is used to create Docker Container. You can define an environment from a **conda specification, Docker image, or Docker build context.** We will be using conda specification.

   You can do this by using the following code

```
conda_deps = CondaDependencies.create(conda_packages=['numpy', 'pandas', 'spacy', 'scikit-learn'],
                                      pip_packages=['azureml-defaults', 'inference-schema'])
env = Environment(name='myenv')
env.python.conda_dependencies = conda_deps
```

4. Create an entry script. The entry script receives data submitted to a deployed web service and passes it to the model. It then returns the model's response to the client. The script is specific to your model. The entry script must understand the data that the model expects and returns.

   The two things you need to accomplish in your entry script are:

   a. Loading your model (using a function called init())
   b. Running your model on input data (using a function called run())

Code for entry script :

```
def init():
    global model

    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sentiment_analysis_model.pkl')
    model = joblib.load(model_path)


def run(data):
    r = json.loads(data)["review"]
    prediction = int(model.predict([r])[0])
    return prediction
```

5. Define an inference configuration. An inference configuration describes the Docker container and files to use when initializing your web service. The inference configuration below specifies that the machine learning deployment will use the file **scoring.py** in the **./source_dir** directory to process incoming requests and that it will use the Docker image with the Python packages specified in the **env** environment.

```
inf_config = InferenceConfig(
    environment=env,
    source_directory="./source_dir",
    entry_script="./scoring.py",
)
```

6. Make an AKS Cluster. We can make this using Azure Portal or Python SDK.

We have used azure portal to make the cluster and then refer that cluster in the script.

```
aks_target = ComputeTarget(workspace=ws, name="my-aks")
```

7. Create a deployment configuration that describes the compute resources needed. For example, the number of cores and memory. Our case we use **1 CPU core** and **1 GB memory.**

```
deployment_config = AksWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)
```

8. Call the deploy function to deploy the model to AKS. The functions takes the following parameters: workspace, endpoint name, registered model, inference config, deployment config and the Kubernetes Compute we created earlier.

```
service = Model.deploy(
    ws,
    "myservice",
    [model],
    inf_config,
    deployment_config,
    aks_target,
    overwrite=True,
)
service.wait_for_deployment(show_output=True)
```

After the model is successfully deployed on AKS you can consume the model to make predictions.

```
@app.route("/predict")
def predict():
    review = request.args.get('review')
    data = {"review": review}
    url = 'http://20.207.64.67:80/api/v1/service/myservice/score'
    api_key = 's9dWawXU0BaDV0fw6p8x4e6WyqkFmFYa'
    headers = {'Content-Type': 'application/json', 'Authorization': ('Bearer ' + api_key)}
    req = requests.post(url, headers=headers, json=data)
    result = req.text
    return f"<h1>{result}</h1>"
```

Then deploy the Flask API to Azure Web App to make it available to users.

Test endpoints :

https://sentimentdeployapp.azurewebsites.net/predict?review=movie%20was%20good

https://sentimentdeployapp.azurewebsites.net/predict?review=movie%20was%20bad

https://sentimentdeployapp.azurewebsites.net/predict?review=star%20wars%20is%20great

https://sentimentdeployapp.azurewebsites.net/predict?review=bollywood%20is%20getting%20bad

https://sentimentdeployapp.azurewebsites.net/predict?review=ott%20is%20getting%20good

https://sentimentdeployapp.azurewebsites.net/predict?review=marvel%20that%20was%20bad

https://sentimentdeployapp.azurewebsites.net/predict?review=End%20Game%20Deserved%20a%2010