ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Developing and Deploying MERN Stack on Azure AKS: A Comprehensive Guide

Aauti

USA

ABSTRACT

This paper introduces Azure Kubernetes Service (AKS), a managed Kubernetes service provided by Microsoft Azure that enables the execution and management of containerized applications in the cloud. The prerequisite knowledge for engaging with AKS, including familiarity with Docker and Kubernetes fundamentals, is highlighted, alongside resources for acquiring this essential information. The document outlines the preliminary steps necessary to begin utilizing AKS, such as creating a Microsoft Azure account, securing a subscription for billing purposes, and various methods for creating an AKS cluster, including through the Azure portal, Azure CLI, and REST API. Additionally, the paper details the process of installing and configuring Azure CLI and kubectl for AKS cluster management, integrating and attaching a container registry to the cluster, and creating a deployment coupled with a Load Balancer service for external access. Accessing the AKS dashboard through Kubeconfig or Token, and important considerations for managing cloud expenses by deleting unnecessary resources, are also discussed. This comprehensive guide aims to equip users with the knowledge to successfully deploy and manage containerized applications using AKS, emphasizing ease of use, scalability, and operational efficiency.

***Corresponding author**
Aauti, USA.

Received: February 16, 2024; **Accepted:** February 21, 2024, **Published:** February 28, 2024

Keywords: MERN, Azure, Cloud Computing, Programming, Software Development

Azure Kubernetes Service (AKS) is Microsoft Azure's premier managed service offering for Kubernetes, designed to facilitate the deployment and management of containerized applications within the cloud environment. As a managed service, AKS alleviates the burden on users by automating critical aspects such as security protocols, regular maintenance, scalable deployment options, and comprehensive monitoring of the infrastructure. This allows developers to focus on rapid deployment of applications to the Kubernetes cluster, sparing them the complexity of its construction.

In the forthcoming discussion, our focus will be on deploying an application built using the MERN stack. The process begins with containerizing the application, followed by uploading the resultant Docker image to the Azure Container Registry (ACR). Subsequently, the application is deployed and run within the Azure AKS environment. Detailed exploration will cover the steps involved in constructing a Kubernetes cluster using Azure AKS, methods to enable external access to these clusters, configuring the kubectl command-line tool for interaction with the AKS cluster, among other operational insights. This guidance aims to streamline the deployment process, ensuring a smooth transition from development to production in a cloud-based Kubernetes ecosystem.

Prerequisites

If you are new to web development, go through the below link on how to develop and build MERN Stack.

- How to Develop and Build MERN Stack (https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-mern-stack-9a7a1099624)

The other prerequisites to this post are Docker essentials. We are not going to discuss the basics such as what is a container or Docker. Below are the prerequisites you should know before going through this article.

Docker Essentials

You need to understand Docker concepts such as creating images, container management, etc. Below are some of the links that you

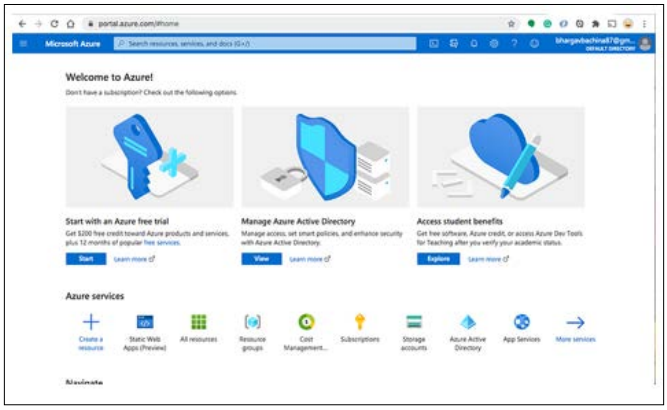
can understand about Docker if you are new.

- Docker Docs
- Docker — A Beginner's guide to Dockerfile with a sample project
- Docker — Image creation and Management
- Docker — Container Management with Examples
- Understanding Docker Volumes with an example

Microsoft Azure Account

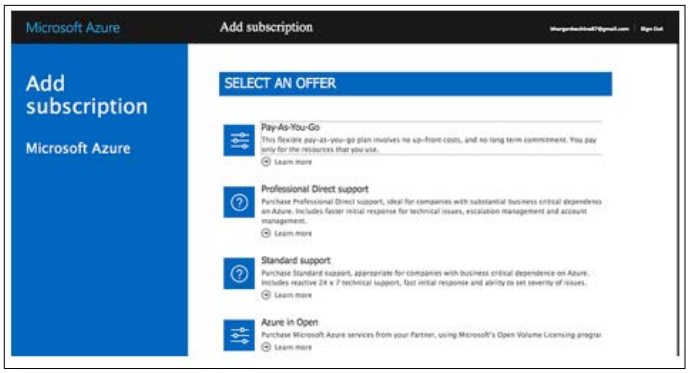
You should have a Microsoft Azure Account. You can get a free account for one year. You should see the below screen after you log in.

- Azure Account

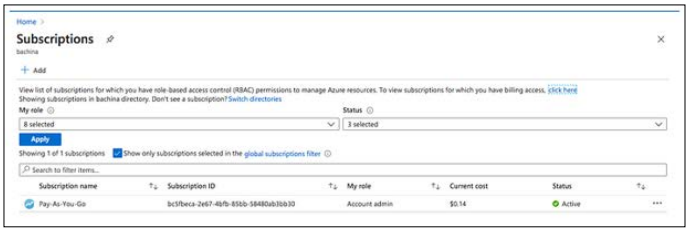


Azure Home Screen

You need to create a subscription for your account. The most common is Pay as You Go subscription.



Subscription Offers



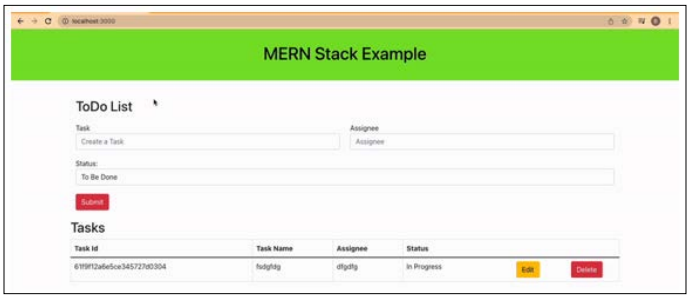
Pay-As-You-Go Subscription

You need a subscription to be associated with your tenant so that all the cost is billed to this subscription. If you are new to Azure, please go through the below article on how to get started with Azure.

- How To Get Started with Azure

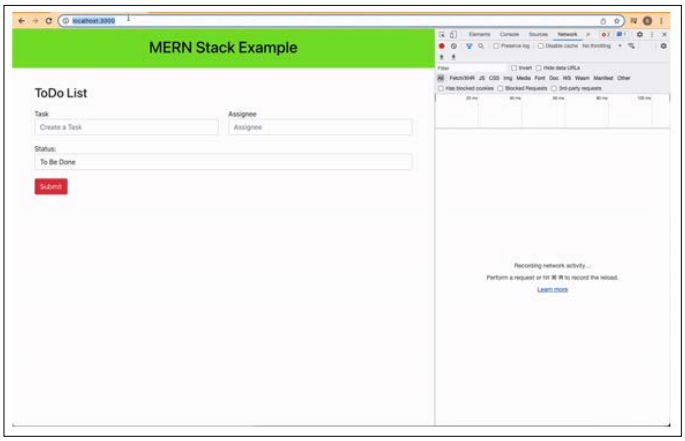
Example Project

Here is an example of a simple tasks application that creates, retrieves, edits, and deletes tasks. We actually run the API on the NodeJS server and you can use MongoDB to save all these tasks.



Example Project

As you add users, we are making an API call to the nodejs server to store them and get the same data from the server when we retrieve them. You can see network calls in the following video.



Network Calls

Here is a GitHub link to this project. You can clone it and run it on your machine.

```
// clone the project
git clone https://github.com/bbachi/mern-stack-example

// React Code
cd ui
npm install
npm start

// API code
cd api
npm install
npm run dev
```

Install Azure CLI and Configure

Once you have the Azure Account you can install Azure CLI. You can go to the below documentation and install Azure CLI based on your operating system. You can configure Azure CLI with your subscription.

- Install Azure CLI
- Login into your account

```
Bhargavs-MacBook-Pro:sample-workspace bhargavbachina$ az login
You have logged in. Now let us find all the subscriptions to which you have access...
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "44e2e07f-a19d-42b6-80a9-a8e4097f3948",
    "id": "bc5fbeca-2e67-4bfb-85bb-58480ab3bb30",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Pay-As-You-Go",
    "state": "Enabled",
    "tenantId": "44e2e07f-a19d-42b6-80a9-a8e4097f3948",
    "user": {
      "name": "bhargavbachina87@gmail.com",
      "type": "user"
    }
  }
]
```

az login

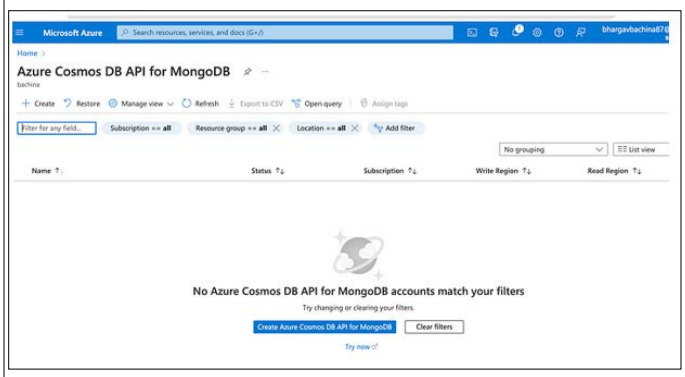
Let's list the subscription with the following command.
az account list

Creating a COSMOSDB with MONGO API

You need to have a Microsoft Azure Account to create a CosmosDB Account. If you are new to Microsoft Azure or not familiar with it. I would recommend you go through the below article.

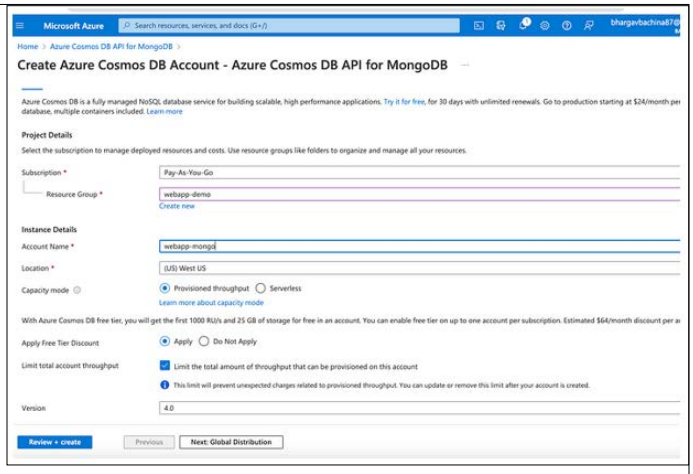
• How To Get Started with Azure

Let's go to CosmosDB and create one in the portal. The important part is that you need to select the API. Since we are creating CosmosDB for MongoDB API you must select that one. You can even select the version of MongoDB.



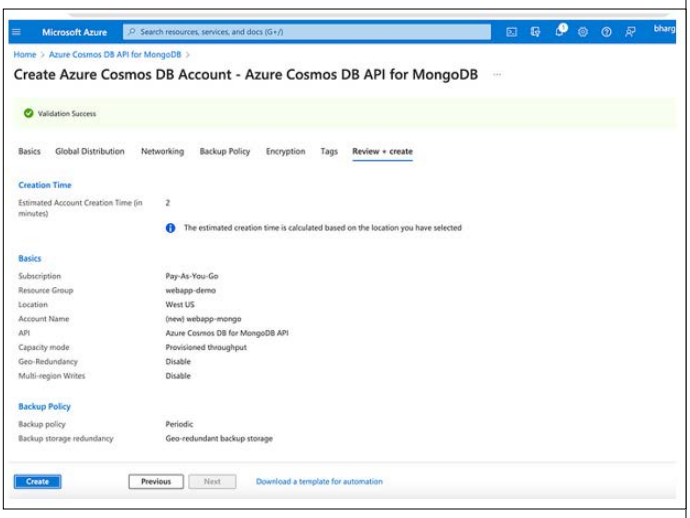
Creating Azure Cosmos DB account

On the next screen, you need to give a name for the account and select which resource group you want to put this in.



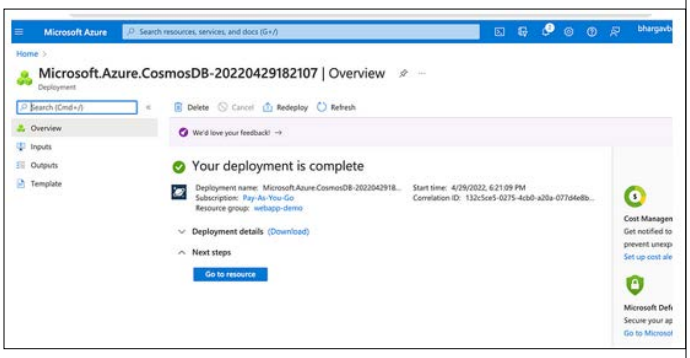
Creating Azure Cosmos DB account

You can configure Networking, Backup policy, tags, etc. in the next tabs. It takes around 2 minutes to create a CosmosDB Account.



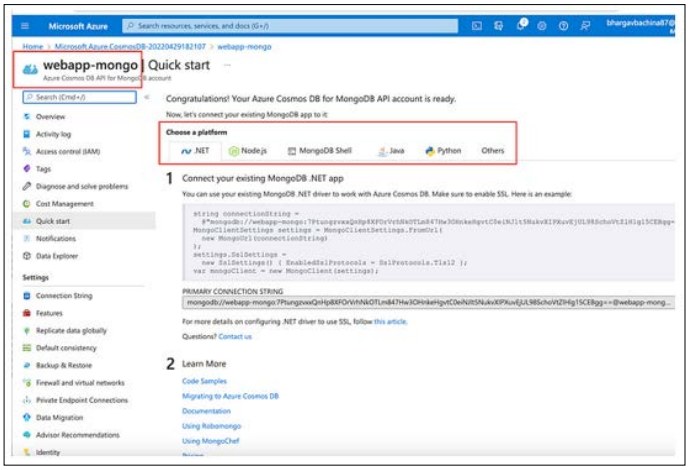
Creating Azure Cosmos DB account

Once completed you can see the message completed and you can click on the Go to resource button.



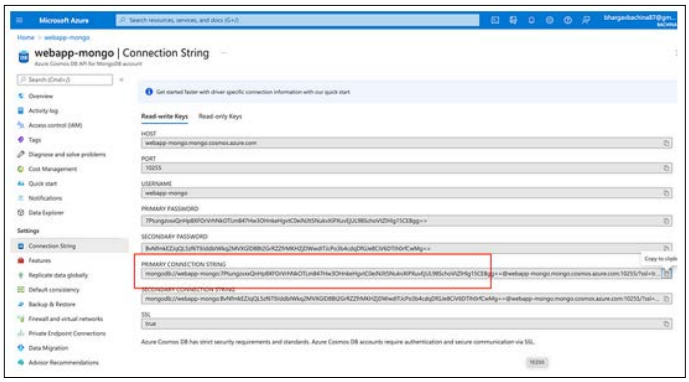
Deployment Complete

Here is the overview page. You can see the account name and the number of ways you can connect to this Cosmos DB Account with different platforms.



Cosmos DB Overview page

Here is the connection string that we need to use to connect to the database. You can take it from the Connection String section.



Connection String

Build for Production

Numerous approaches exist for constructing a MERN Stack for production deployment, with the optimal strategy varying based on the specific use case or deployment environment. This paper delineates various methodologies for preparing the MERN Stack for production use.

• How to Build MERN Stack for Production (https://medium.com/bb-tutorials-and-thoughts/how-to-build-mern-stack-for-production-1462e70a35cb)

Externalize Environment Variables

Reading environment variables is one of the most common things that we do when we are building apps. It doesn't matter whether you are developing front end app or back-end API you have so many variables that should be outside of your application source code that makes your app or API more configurable. For example, if you want to hide logger statements in production or do something else based on the environment you can pass this as an environment variable. If you want to change later all you need to change is in one place.

• Reading Environment Variables in NodeJS api (https://medium.com/bb-tutorials-and-thoughts/reading-environment-variables-in-nodejs-rest-api-e75bb04b813d)

When it comes to this application, there are two environment variables, one is the Mongo Connection string, and another one is PORT.

```
PORT=80
MONGO_CONNECTION_STRING=mongodb+srv://admin123:admin123@todo-cluster.zpikr.mongodb.net/?retryWrites=true&w=majority
```

You must put these in the webpack.config.js file so that these values are used when we dockerize the app for production.

• Webpack.config.js file (https://gist.github.com/bbachi/aa25aec5b82320d28cc5ee137bb8b8cf#file-webpack-config-js)

Dockerize the Webapp

Azure AKS is a managed service that makes it easy for you to run Kubernetes on Azure. The first thing you need to do is to dockerize your project.

We use multi-stage builds for efficient docker images. Building efficient Docker images are very important for faster downloads and lesser surface attacks. In this multi-stage build, building a React app and putting those static assets in the build folder is the first step. The second step involves building the API. Finally, the third step involves taking those static build files and API build and serving the React static files through the API server.

We need to update the server.js file in the NodeJS API to let Express know about the React static assets and send the index.html as a default route. Here is the updated server.js file. Notice the line numbers 41 and 20.

• Server.js file (https://gist.github.com/bbachi/e828985a85cfb9da08164afa88549211#file-server-js)

Let's build an image with the Dockerfile. Here are the things we need for building an image.

In constructing a production-ready application using the MERN Stack, the process begins with a base image of node:14-slim. Initially, both package.json files—one for the Node.js server and the other for the React UI—are copied into the Docker file system, with dependencies installed to enhance build speed for subsequent changes. This preemptive step prevents the redundancy of reinstalling dependencies with each source modification. Following this, all source files are copied, and dependencies installed, culminating in the execution of npm run build to generate the React application assets within a 'build' folder inside the 'ui' directory. The second stage also utilizes the node:14-slim base image, focusing on the Node.js environment by copying its package.json into an './api' directory, installing necessary dependencies, and incorporating the server.js file into this directory. The final stage combines the elements, starting again with the node:14-slim image, to amalgamate the built UI and API files, concluding with the command node api.bundle.js to run the bundled server application, thereby streamlining the deployment of the MERN Stack for production.

Here is the Complete Dockerfile link where you can run on your machine.

• Docker file (https://gist.github.com/bbachi/06eecfc6c956d01c99180523c2677c15#file-dockerfile)

Let's build the image with the following command.

```
// build the image
docker build -t mern-image .

// check the images
docker images
```

Running the Webapp on Docker

Once the Docker image is built. You can run the image with the following command.

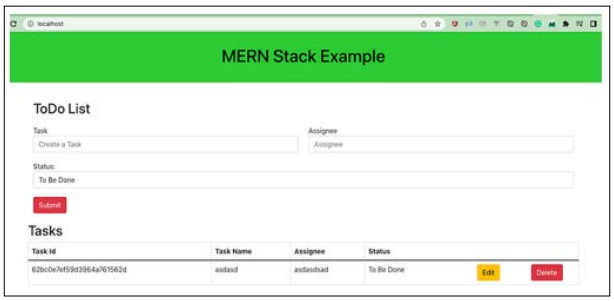
```
// run the container
docker run -d -p 80:80 --name mern-stack mern-image

// list the container
docker ps

// logs
docker logs mern-stack

// exec into running container
docker exec -it mern-stack /bin/sh
```

You can access the application on the web at this address http://localhost



Example Project

Pushing Docker Image to ACR

Azure AKS works with any Docker registry such as Docker Hub, etc. But, in this post, we see how we can use the Azure container registry to store our Docker images. Once you set up the Azure portal account and create a resource group as above you can create a container registry as below.

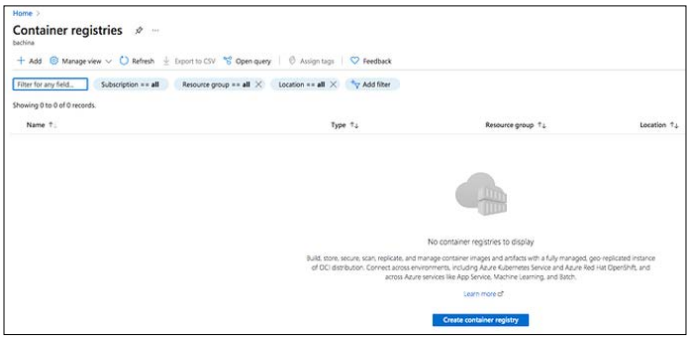
Make sure your application is running on port 80 before you create an image by changing the port that the web app is listening to. You can change the default port, or you need to make sure you must pass in the environment variables.

• Dockerfile (https://gist.github.com/bbachi/794565dce3558f19ae06cfdc46eff448#file-dockerfile)

Let's first create a resource group called webapp-demo with the following command.

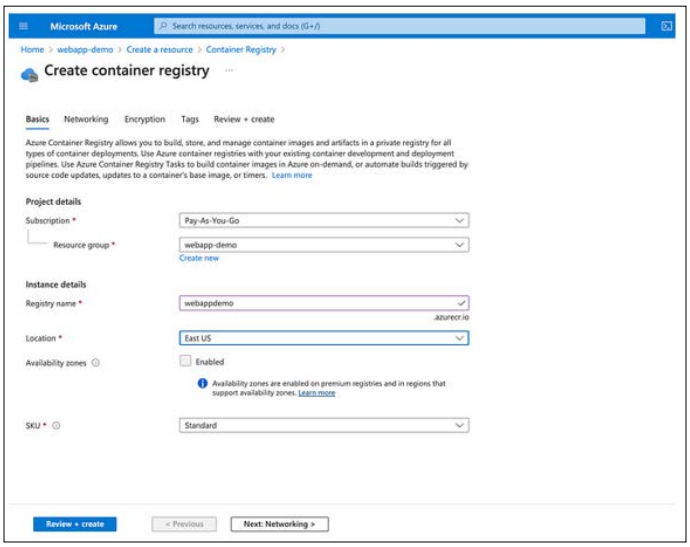
```
// create a resource group
az group create --name webapp-demo --location westus
```

Once you have created the above resource group you can create a container registry by going to the following screen.



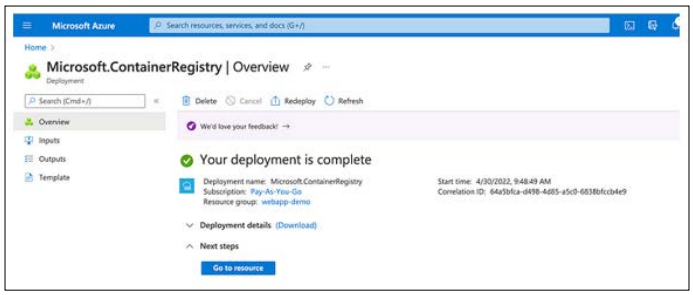
Container Registries

Let's give the basic details on the create screen and click on the create button.



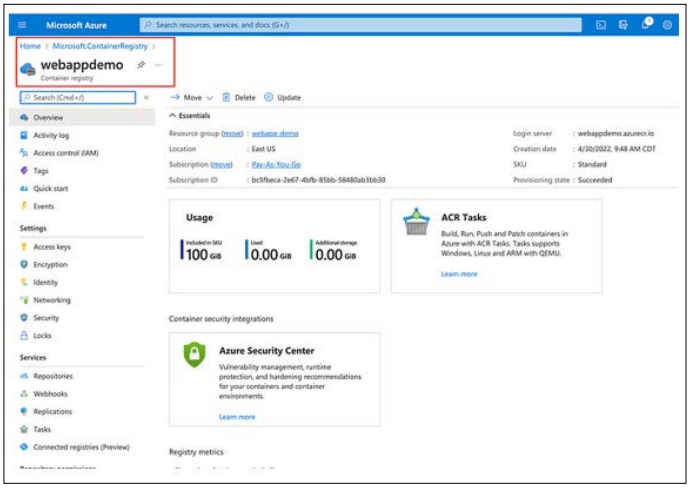
Creating container registry

Once you review and create it you can see the following screen.



Deployment Completed

You can see the main container registry page below.



Container Registry Created.

You can do the same things with the Azure CLI with the following commands. Make sure you log in to your Azure Account with CLI with this command az login before running the below commands.

```
// create a resource group
az group create --name webapp-demo --location westus

// create a container registry
az acr create --resource-group webapp-demo \
  --name webappdemo --sku Basic
```

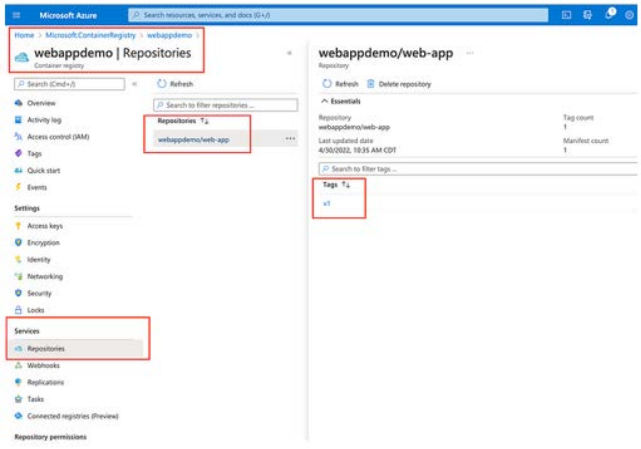
It's time to build and push the Docker image with the following command. Clone the above example project and go to the root folder where Dockerfile resides and run this command.

```
az acr build --image webappdemo/web-app:v1 \
  --registry webappdemo \
  --file Dockerfile .
```

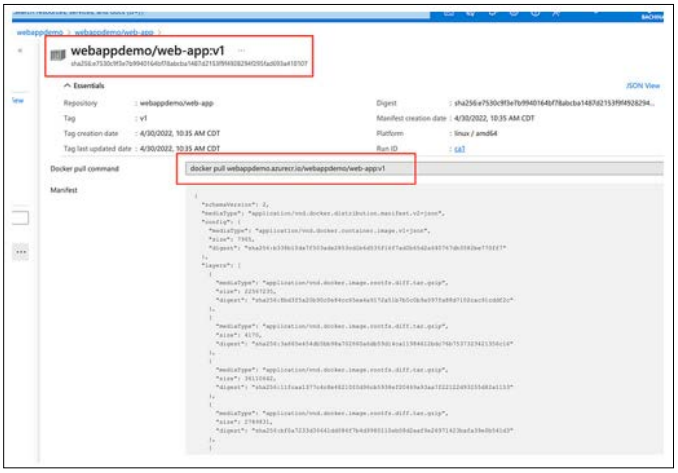
You need to make sure the API is serving the React static files from the folder ui/build as below on lines 20 and 41 since we are placing the API and UI build files in that project structure in the Dockerfile.

• Server.js file (https://gist.github.com/bbachi/e8267a871506caad0f2a5f3722766b7c#file-server-js)

You can see all the details in the portal as well.



Repository Details



Docker Pull Command

If you want to pull this repository you need to use this command.

docker pull webappdemo.azurecr.io/webappdemo/web-app:v1

Creating AKS Cluster

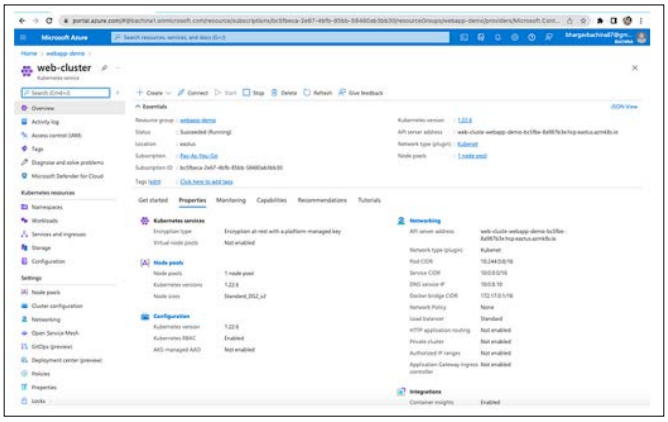
First, you need a resource group for all your resources. Let's create a resource with the following command.

az group create --name webapp-demo --location westus

Let's create a cluster with the following command. Notice that we are using the same resource group that we created above. You can see the JSON formatted result after a few minutes.

az aks create --resource-group webapp-demo --name web-cluster --node-count 3 --enable-addons monitoring --generate-ssh-keys

You can see the following cluster in the console.



Web-Cluster Created

Configure Kubectl with AKS Cluster

Kubectl is the command-line utility for the Kubernetes. You need to install kubectl before you configure it. Run the first command only if you don't have kubectl on your local machine.

```
// install CLI
az aks install-cli

// connect to your cluster
az aks get-credentials --resource-group webapp-demo --name web-cluster

// get all the contexts
kubectl config get-contexts

// verify the current context
kubectl config current-context

// get the node
kubectl get nodes
```

Configure kubectl with AKS Cluster

Deploy Kubernetes Objects on Azure AKS Cluster

Now we have configured kubectl to use Azure AKS from our own machine. You need to integrate the container registry with the AKS. Let's attach the container registry with the cluster with the following command. You can explore the docs here regarding this.

az aks update -n web-cluster -g webapp-demo --attach-acr webappdemo

Let's create deployment and service objects and use the image from the Azure container registry. Here is the manifest file which contains these objects.

Let's create deployment and service objects and use the image from the Azure container registry. Here is the manifest file which contains these objects. Here is the complete Manifest YAML file

• manifest.yaml (https://gist.github.com/bbachi/2343dceb2c8c6ae5f1cf00eb8a8b2ef6#file-manifest-yaml)

If you cloned the above example project and you are at the root folder just use this command to create objects kubectl create -f manifest.yaml

You can use the following commands to verify all the objects are in the desired state.

```
// list the deployment
kubectl get deploy

// list the pods
kubectl get po

// list the service
kubectl get svc
```

We can see 5 pods running since we have defined 5 replicas for the deployment.

```
bash-3.2$ kubectl create -f manifest.yaml
deployment.apps/webapp created
service/webapp created
bash-3.2$
bash-3.2$
bash-3.2$ kubectl get po
NAME                      READY   STATUS    RESTARTS   AGE
webapp-54dd88b99d-f28ct   1/1     Running   0          6s
webapp-54dd88b99d-l7qn4   1/1     Running   0          6s
webapp-54dd88b99d-p4zwc   1/1     Running   0          6s
webapp-54dd88b99d-rr747   1/1     Running   0          6s
webapp-54dd88b99d-rzc7m   1/1     Running   0          6s
bash-3.2$
bash-3.2$
bash-3.2$ kubectl get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
kubernetes   ClusterIP      10.0.0.1       <none>          443/TCP        21m
webapp       LoadBalancer   10.0.165.79    52.226.231.14   80:30168/TCP   11s
bash-3.2$
```

Kubernetes Objects Running on Azure AKS

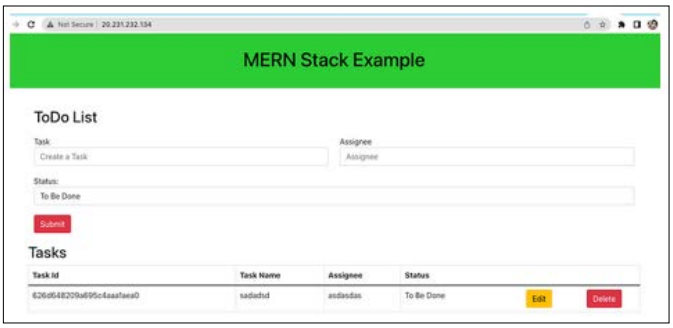
Acess the Webapp from the Browser

We have created a service with the Load Balancer type. You can get the external IP from the service and access the entire from the browser.

```
bash-3.2$ kubectl get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.0.0.1       <none>           443/TCP        47m
webapp       LoadBalancer   10.0.174.133   20.231.232.134   80:30836/TCP   13s
bash-3.2$
```

Service

You can access the webapp with the following URL.
http://20.231.232.134/



Accessing from the browser

Delete the Cluster

You can just delete the cluster or the resource group. I created a resource group just for this, so I am deleting the resource group with the following command. Make sure you delete it if you don't want to incur charges [1-4].

az group delete --name webapp-demo

Summary

- AKS is Microsoft Azure's managed Kubernetes solution that lets you run and manage containerized applications in the cloud.
- Before starting this, you need to have docker and Kubernetes essentials. If you don't have these essentials, please go through them with the links provided.
- You need to create a Microsoft Azure Account here.
- You need a subscription to be associated with your tenant so that all the cost is billed to this subscription.
- You can create the AKS cluster through a portal, Azure CLI, and REST API as well.
- You can install Azure CLI and configure it to use with your AKS Cluster.

- Configure kubectl to use the AKS cluster.
- You need to integrate the container registry with the AKS and you need to attach the container registry to the cluster in many ways. You can explore the docs here regarding this.
- Create a deployment and service with Loadbalancer so that you can access it from the outside.
- You can access the dashboard with either Kubeconfig or Token.

Conclusion

In conclusion, Azure Kubernetes Service (AKS) stands out as Microsoft Azure's comprehensive managed service offering for Kubernetes, designed to streamline the deployment and management of containerized applications in the cloud. The journey to leveraging AKS involves a series of preparatory steps, beginning with the acquisition of foundational knowledge in Docker and Kubernetes, followed by the creation of an Azure account and the association of a subscription for billing purposes. The flexibility in creating an AKS cluster is demonstrated through multiple interfaces, including the Azure portal, CLI, and REST API, with further configurations facilitated by Azure CLI and kubectl for effective cluster management. Integration with Azure Container Registry is essential for a seamless deployment process, emphasizing the importance of creating a deployment and service equipped with a Load Balancer for external accessibility. Additionally, accessing the AKS dashboard through Kubeconfig or Token is detailed, underscoring the importance of resource management to avoid unnecessary charges. This comprehensive guide illustrates the streamlined process enabled by AKS for deploying and managing containerized applications, highlighting its efficacy and the critical steps involved from initiation to successful deployment and beyond.

References

1. Bhargav Bachina (2022) How to develop and build MERN Stack https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-mern-stack-9a7a1099624.
2. Bhargav Bachina (2022) How to Build MERN Stack for Production https://medium.com/bb-tutorials-and-thoughts/how-to-build-mern-stack-for-production-1462e70a35cb.
3. Azure Cloud https://azure.microsoft.com/en-us.
4. Kubernetes Documentation https://kubernetes.io/docs/home/.