Deployment of a Microservices Architecture based Web App by using Docker, Kubernetes, and Azure DevOps

Anik Saha, Mursheda Baby, Shalim Sadman
High Integrity System (MSc.)
Frankfurt University of Applied Sciences

Abstract:

Microservices are a method of building cloud applications that is both architectural and organizational. Microservices break down an application into smaller independent pieces that need a way to be managed. Containerization with Kubernetes orchestration and management is designed to support microservices. Microsoft Azure offers Azure Kubernetes Service that simplifies managed Kubernetes cluster deployment in the public cloud environment and manages the health and monitoring of managed Kubernetes service. This project includes three servers that leverage container-based technologies inside a microservice architecture, making resource utilization easier and more efficient. The idea is to explain containerized application deployment into the Kubernetes cluster with Azure DevOps, Dockerhub, and a Github repository.

1. Introduction

Microservices are based on the concept of deploying individual apps, each of which delivers a specific service and then connecting these microservices to form a larger service. Microservices make it easier to deploy and upgrade individual apps, making it easier to create sophisticated services [16]. The usage of Docker to host microservices is becoming increasingly popular. Docker provides a command-line interface (CLI) for creating Docker images, which may then be used as templates for creating containers [8]. Images can be maintained locally or published to the Docker Hub, where they can be made privately or publicly visible. It's simple to install and configure one or more Docker containers on a host to establish an integrated service. However, managing a cluster of compute nodes, each of which hosts several containers, is more complicated, and some ways of cluster management are required. Kubernetes is a container orchestration system that may be used in various contexts, including Google Compute Engine and Microsoft Azure.
The Azure Kubernetes Service (AKS) is the most convenient method to start with Kubernetes on Azure. It is unquestionably an ideal platform for developers to build modern apps using Kubernetes on the Azure architecture, with Azure Container Instances being an excellent alternative for public cloud container deployment. Azure Container Instances allow developers to install and execute their applications on Kubernetes architecture with less stress.

Organizations are increasingly modernizing application development by incorporating open source technologies into a comprehensive architecture for cloud delivery of high-quality workloads [4]. This document will introduce some basic Kubernetes ideas and provide step-by-step guidance on deploying a simple web application using Azure DevOps with Kubernetes and Docker.

2. System overview

This section contains a brief summary of the technology, processes, and approaches used in the system. Finally, the techniques and architecture utilized to construct the overall system are described in depth.

2.1 Microservices Technology and Architecture

Microservices are mostly utilized to alleviate issues caused by monolithic systems. Microservices is a design approach that reflects an application's structure by merging all of the application's autonomous services. Microservices architecture allows the distribution of an application among services that are independent of one another. The following are some key characteristics of a microservice:

High availability – Every microservice should respond to all requests in a reasonable amount of time.

Partition tolerance – If a microservice instance fails, the system should not be affected. Should ensure that the availability of microservices is consistent.

Eventual consistency – When data in a single microservice changes, it should eventually be propagated to other microservices that are relevant.

Asynchronous communication -- Microservices should not connect asynchronously. Instead, they should delegate communication to a message broker [15].

2.2 Communication Between Microservices on Kubernetes

There are several approaches to expose a Kubernetes-based application:

The most common technique is to use a Kubernetes service, a network abstraction or logical entity of pods. There are a variety of services available, each with its own set of capabilities; this project makes use of the following:

ClusterIP uses Kubernetes' default service option and exposes the service on a cluster-internal IP.

NodePort exposes the application to the outside world via a node-level static IP and port combination.

LoadBalancer makes the application available as a service to a cloud-based load balancer [17].

2.3 Kubernetes Basics:

Kubernetes is an open-source framework that helps to manage containerized workloads and services, including declarative and automated setup capabilities [2]. Kubernetes consist of numerous components that are completely unaware of each other. All of the components interact with one another via the API server. Before exposing metrics that can be gathered for further monitoring, each component has a specific purpose [10].

Let's go over the Kubernetes fundamentals that were used to create this project:

A "pod" in Kubernetes is a collection of functionally connected containers.

A "service" is a collection of connected pods that perform the same set of tasks. Kubernetes gives each Pod its IP address and a collection of Pods a single DNS name. IP addresses for pods are not stable because each new pod is allocated a new IP address; hence, a direct connection between pods is not often viable. On the other hand, services have their relatively constant IP addresses; consequently, an external resource requests a service rather than a pod, and the service forwards the request to an available pod [5].

For non-confidential data, a ConfigMap is a key-value pair storage API object. Pods can use ConfigMaps as environment variables, command-line options, and disk configuration files. With a ConfigMap, we can isolate environment-specific configuration from container images, making the apps more portable.

A Kubernetes deployment is a resource object that allows apps to get declarative updates in Kubernetes. A deployment allows defining an application's life cycle, including which images to use, how many pods should be present, and how they should be updated.

The PersistentVolume subsystem provides a user-friendly API that separates storage provisioning from storage consumption. Two new API resources that help with this are PersistentVolume and PersistentVolumeClaim.

A secret is a small amount of confidential data, such as a password, token, or key. Alternatively, such information might be included in a Pod specification or an image. Secrets can be created by users, and the system will also generate certain Secrets.

When a workload demands surges or falls, Kubernetes can automatically raise or reduce the number of pod replicas providing the workload, known as horizontal scaling. This is a dynamic feature with a reconciliation loop that uses experimental measurements to drive the workload's capacity toward the capacity established by the workload's owner.

2.4 System Architecture

Azure DevOps, which manages container deployment to the Kubernetes cluster, serves as the system architecture's backbone. As a source control repository, we used Github, and as a content repository, we used DockerHub. The Azure DevOps pipeline is in charge of container creation and deployment. When the developer updates the Github source, the pipeline is started. To create apps and communicate with docker hub, GitHub, and Azure Kubernetes services, Azure DevOps uses plugins. The services are dockerized and hosted in a public DockerHub repository before being deployed to an Azure Kubernetes cluster. A load balancer service in a Kubernetes cluster directs traffic to available pods. Figure 1 depicts our high-level system design.

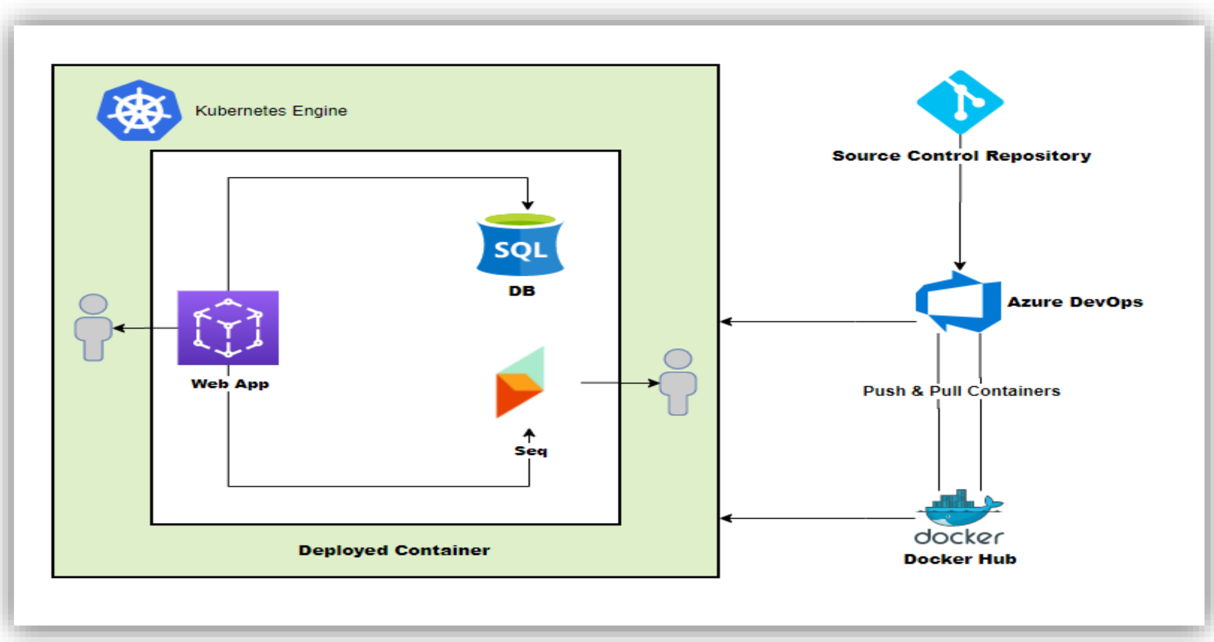


Figure 01: System Architecture

2.5 Tools and Methodologies

Deployment tool: Kubernetes, Azure DevOps

Programming Language: C#, HTML, CSS, JavaScript

Source Control Repository: GitHub

Container repository: Docker Hub

Log platform: Seq

Testing tool: k6

3. Implementation:

The section details the implementation steps towards achieving our desired objectives. Three servers have been used for the implementation. They are Web app (two pods), log server (1 pod), and MySQL server (1 pod). Cmd is used to run the Kubernetes command.

3.1 About MVC Web Application:

It's a simple web application. There are only three pages in this section. The first is the homepage. The Privacy page is the second, and the Product Information System is the third. We just save product names in our system because our whole focus is on Kubernetes right now. A product's name can be easily added, deleted, or edited by anyone.

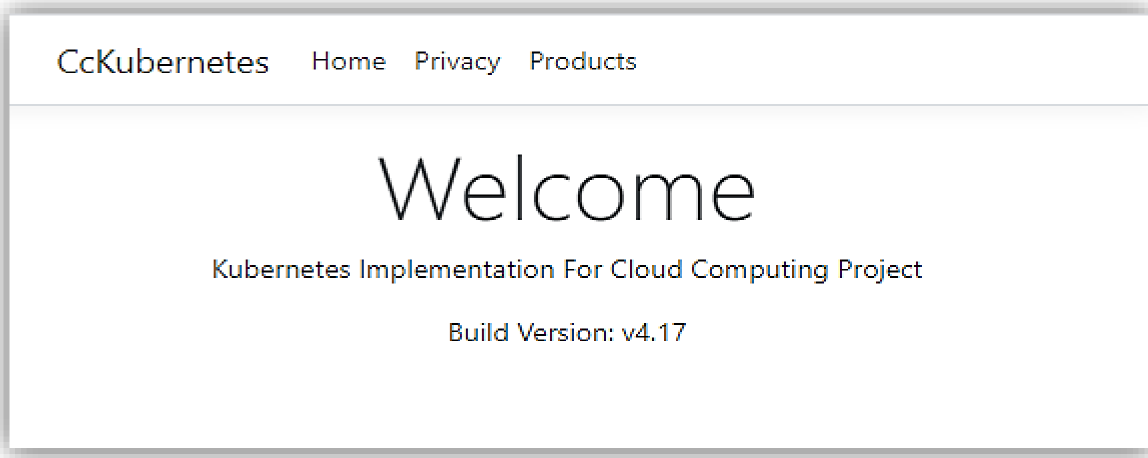


Figure 02: Home Page

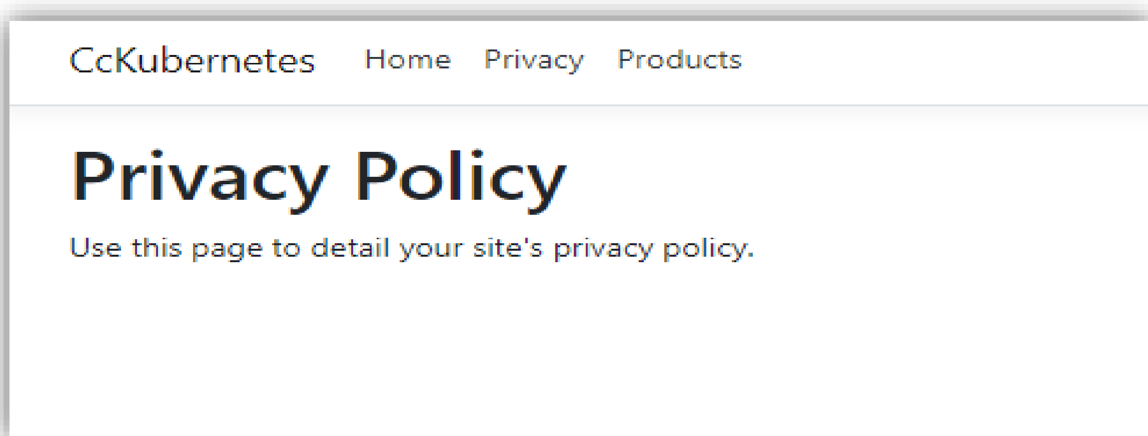


Figure 03: Privacy page

The code of the home and privacy page is available here.

Then we are adding create, delete, edit of product.

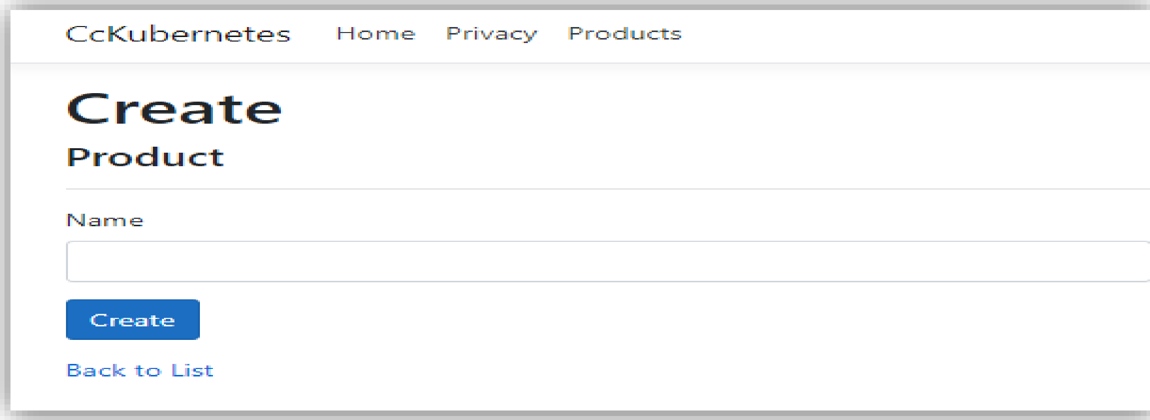


Figure 04: Creating product

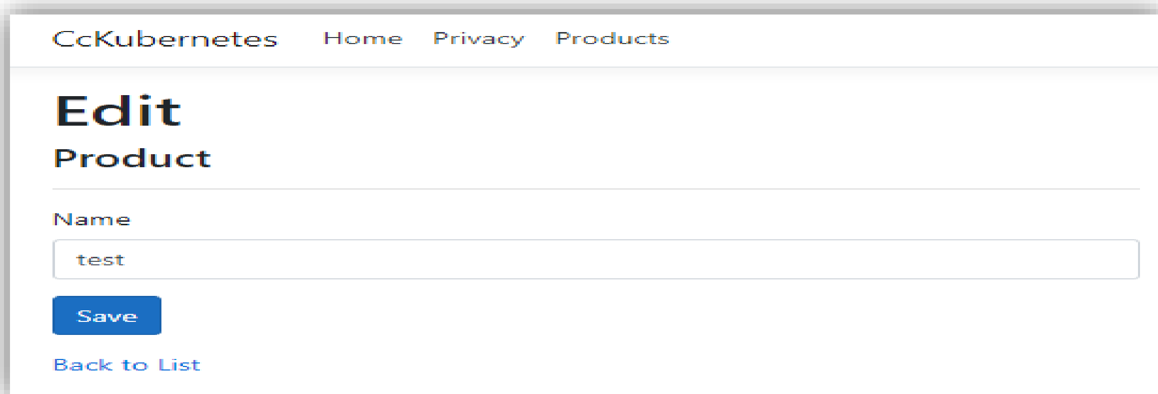


Figure 05: Editing product

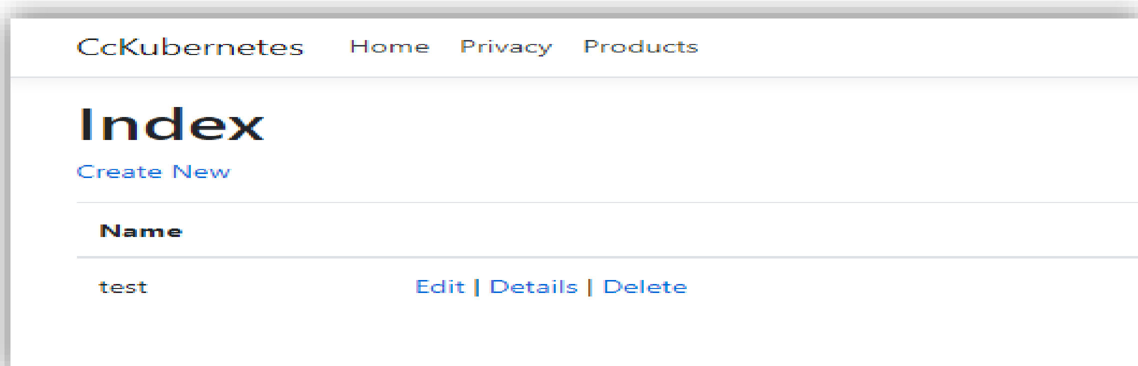


Figure 06: Added product

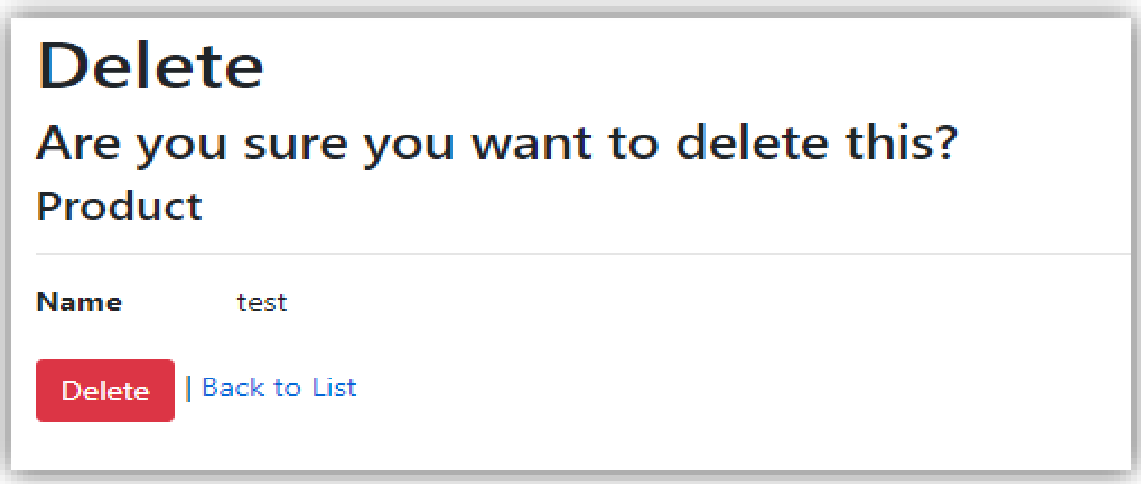


Figure 07: Deleting product

The code of the whole product information system is here.

3.2 Creating a Docker Image (Creating the Docker containers for the app)

The following set of commands inside 'Dockerfile' is used to build and run the web app into the Docker container.

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0-buster-slim AS build
WORKDIR /src
COPY "CcKubernetes.csproj" .
RUN dotnet restore "CcKubernetes.csproj"
COPY . .
RUN dotnet build . -c Release -o /app/build

#RUN apt-get install curl

FROM build AS publish
RUN dotnet publish "CcKubernetes.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish.
ENTRYPOINT ["dotnet", "CcKubernetes.dll"]
```

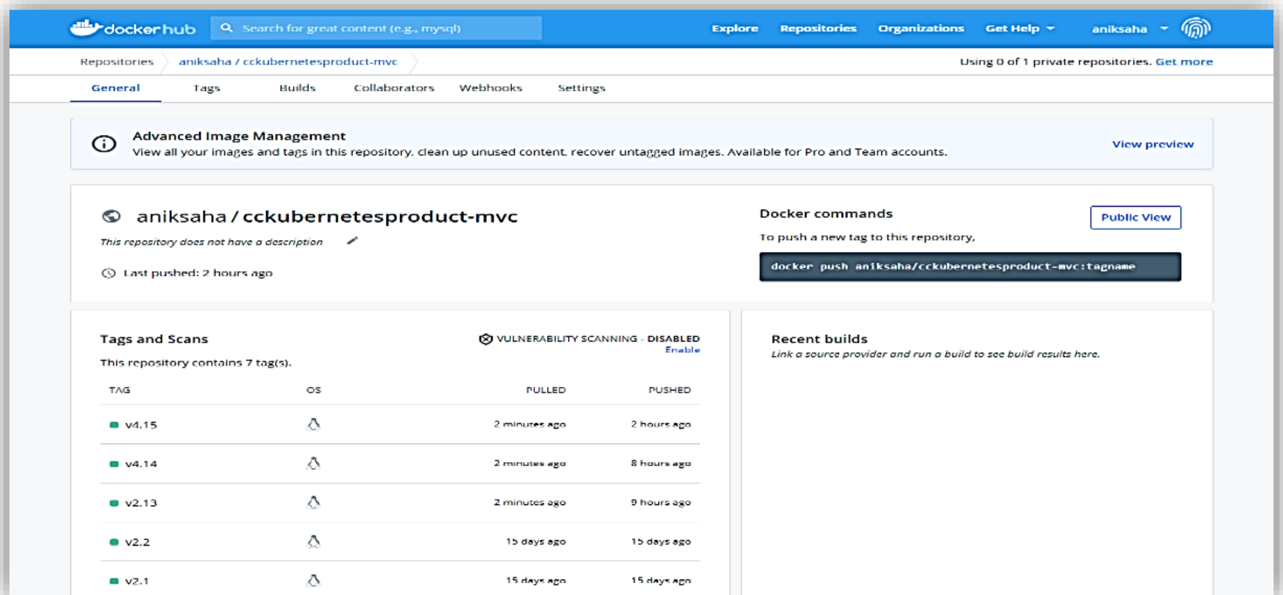


Figure 08: Image on Docker Hub

For SQL server and Seq log server, Official images for Microsoft SQL Server and Seq log server have been used respectively for Docker Engine.

Log server - Seq:

Seq is a modern structured logging system that includes message templates. It combines free-text and regular expression searches with intuitive expression-based filtering. Instead of wasting time and effort trying to extract data from plain-text logs using inaccurate log parsing, the properties of each log event are collected and sent to Seq in a clean JSON format. Message templates are natively supported by ASP.NET Core, and because our application is built on it, it provides the optimum diagnostic logging for our platform [9].

It is a third-party log for this project. We extract this from the docker image. Two ports are set in this seq log to get logs that operate through the web app connection. Seq provides the visibility to quickly detect and fix problems in complex systems and microservices.

These logs are delivered over the network to Seq, who displays and searches them:

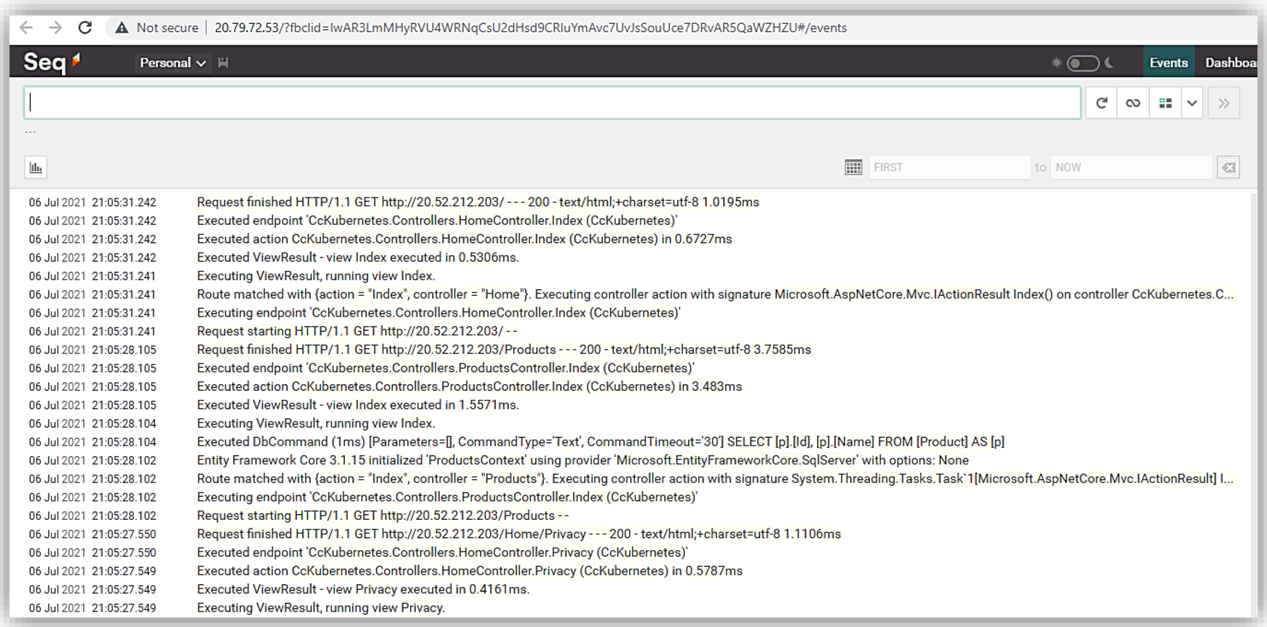


Figure 09: Log query of web application

Here we have given the code of connecting code between seq and our mvc app.

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
          .UseSerilog((ctx, provider, loggerConfig) =>
          {
            loggerConfig
              .ReadFrom.Configuration(ctx.Configuration) // minimum levels defined per project in json files
              .Enrich.FromLogContext()
              .WriteTo.Console()
              .WriteTo.Seq($"http://{ctx.Configuration.GetConnectionString("Seq")}:{ctx.Configuration.GetCo
nnectionString("SeqPort")}");
          })
          .ConfigureWebHostDefaults(webBuilder =>
          {
            webBuilder.UseStartup<Startup>();
          });
  }
```

3.3 Kubernetes on Docker Desktop

Kubernetes is a container-based system. As a result, an account on the Docker hub was first created to push the microservices image and MySQL image to the Docker hub, which is required for Kubernetes deployment. We aim to build up our Kubernetes in a local workstation before playing with the Azure Kubernetes service. It's also simple to set up Kubernetes with Docker Desktop to build and push docker images into the docker hub. We deployed the Kubernetes cluster during Docker installation, which runs all Kubernetes components in containers.

Docker Desktop will download all of the Kubernetes images and start things up in the background. When it's ready, two green lights will appear at the bottom of the settings screen, indicating that Docker and Kubernetes are running.

The Kubectl is then verified with the following command:

```
PS D:\his\Cloud Computing\CloudComputing\CcKubernetes> kubectl version
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.7", GitCommit:"1dd5338295409edcfff11505e7bb246f0d325d15", GitTreeState:"clean", BuildDate:"2021-01-13T13:23:52Z",
GoVersion:"go1.15.5", Compiler:"gc", Platform:"windows/amd64"}
Server Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.11", GitCommit:"c6a2f08fc4378c5381dd948d9ad9d1080e3e6b33", GitTreeState:"clean", BuildDate:"2021-05-13T17:14:18Z",
 GoVersion:"go1.15.12", Compiler:"gc", Platform:"linux/amd64"}
```

Figure 10: kubectl version

3.4 Kubernetes Deployment:

A Kubernetes Deployment instructs Kubernetes on how to generate or change instances of pods that contain containerized applications. Eight YAML files make up the project, which is used to deploy everything in the Kubernetes cluster. Three of these files are deployed in the local machine, and the rest are configured for the Azure platform. The service name and the values of additional YAML files like configmap, secret, and seq log are all contained in the MVC deployment file. A load balancer is included in the file for a new service to make it accessible to others. A similar method is followed when deploying MySQL.

To handle individual components, it's time to put up an orchestrator like Kubernetes. Kubernetes contains several tools for scaling, networking, securing, and supporting containerized applications in addition to the capabilities of containers.

3.5 Describing apps using Kubernetes YAML

In Kubernetes, all containers are scheduled as pods, which are groupings of co-located containers and share some resources. Manifests called Kubernetes YAML files can and should be used to describe all Kubernetes objects. These YAML files explain all of a Kubernetes app's components and configurations and can be used to efficiently construct and destroy apps in any Kubernetes environment.

The configuration file for the web app in this project is called **cckubernetesproduct-mvc.deployment .azure.yaml**. For our mvc project, we've written deployment, service, and horizontal scaler pod code in this file. The default replica set in the deployment phase is 2. Our resources were similarly limited. We also specify that just port 80 is open. We get connection strings from secret for environment variables and build version and Seq port from the config map.

We create a service type loadbalancer on port 80 so that everyone may see our web app. The minimum pod count for horizontal pod scaling is 2, and the maximum pod count is 5. It will also work if the desired CPU utilization is more than or equal to 50.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cckubernetesproduct-mvc
spec:
  selector:
    matchLabels:
      app: mvc
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: mvc
    spec:
      containers:
      - name: mvc
        image: aniksaha/cckubernetesproduct-mvc:v2.#{Build.BuildId}#
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: "0.4"
            memory: "200Mi"
          requests:
            cpu: "0.2"
            memory: "100Mi"
        env:
          - name: ConnectionStrings__ProductsContext
            valueFrom:
              secretKeyRef:
                name: cckubernetesproduct-secret
                key: db-connection-string-secret
          - name: ConnectionStrings__BuildVersion
            valueFrom:
              configMapKeyRef:
                name: cckubernetesproduct-configmap
                key: build-version
          - name: ConnectionStrings__Seq
            value: cckubernetesproduct-seq-log-service
          - name: ConnectionStrings__SeqPort
            valueFrom:
              configMapKeyRef:
                name: cckubernetesproduct-configmap
                key: seq-log-port
---
kind: Service
apiVersion: v1
metadata:
  name: cckubernetesproduct-mvc-service
spec:
  selector:
    app: mvc
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: cckubernetesproduct-mvc-hpa
spec:
  maxReplicas: 5 # define max replica count
  minReplicas: 2  # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
```

```yaml
    kind: Deployment
    name: cckubernetesproduct-mvc
  targetCPUUtilizationPercentage: 50 # target CPU utilization
```

In this file we have written deployment and service for MSSQL . In the deployment part, the default replica is set to 1. We here define one port 1433 that is accessible by others. For environment variables, we collect db password strings from Kubernetes secret.

This MSSQL is exposed to our web app only. So we make a service type Nodeport. That's how our db can not be accessible by others. So no one can temper our db data.

cckubernetesproduct-mssql.deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cckubernetesproduct-mssql
spec:
  replicas: 1
  selector:
     matchLabels:
       app: mssql
  template:
    metadata:
      labels:
        app: mssql
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: mssql
        image: microsoft/mssql-server-linux
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "0.1"
        ports:
        - containerPort: 1433
        env:
        - name: ACCEPT_EULA
          value: "Y"
        - name: SA_PASSWORD
          valueFrom:
            secretKeyRef:
              name: cckubernetesproduct-secret
              key: db-password-secret
        volumeMounts:
        - name: cckubernetesproduct-mssql-persistent-storage
          mountPath: /var/opt/mssql
      volumes:
      - name: cckubernetesproduct-mssql-persistent-storage
        persistentVolumeClaim:
          claimName: cckubernetesproduct-mssql-persistent-volume-claim
---
apiVersion: v1
kind: Service
metadata:
  name: cckubernetesproduct-mssql-service
spec:
  selector:
    app: mssql
  ports:
    - protocol: TCP
      port: 1433
      targetPort: 1433
      nodePort: 30200
  type: NodePort
```

In this file, we have written deployment and service for seq(logging web project). In the deployment part, we set the default replica as 1. We also limited our resources. We have to define two ports. One(5341) is for writing logging from a web project, and another port(80) is opened for showing it externally.

Here, two services as two ports are opened for this pod. Port 5341 is not opened for all. Only our web project will access it internally. So we define cckubernetesproduct-seq-log-service as no cluster IP. On the other hand, Port 80 can be accessed by all as we set type loadbalancer for cckubernetesproduct-seq-ui-service service.

cckubernetesproduct-seq.deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cckubernetesproduct-seq
spec:
  selector:
    matchLabels:
      app: seq-app
  replicas: 1
  template:
    metadata:
      labels:
        app: seq-app
    spec:
      containers:
      - name: seq
        image: datalust/seq:latest
        ports:
        - containerPort: 5341
        - containerPort: 80
        resources:
          limits:
            cpu: "0.5"
            memory: "200Mi"
          requests:
            cpu: "0.25"
            memory: "100Mi"
        env:
        - name: ACCEPT_EULA
          value: "Y"
---
apiVersion: v1
kind: Service
metadata:
  name: cckubernetesproduct-seq-ui-service
spec:
  selector:
    app: seq-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: cckubernetesproduct-seq-log-service
spec:
  selector:
    app: seq-app
  ports:
    - protocol: TCP
      port: 5341
      targetPort: 5341
```

This YAML creates a ConfigMap for external configuration with the build version and the seq logport value that is set to 5341.

cckubernetesproduct-configmap.deployment.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cckubernetesproduct-configmap
data:
  build-version: v2.#{Build.BuildId}#
  seq-log-port: "5341"
```

Here is a configuration file that was used to create secure credentials. All values inside this file are base64 encoded to not be stolen easily.

cckubernetesproduct-secret.deployment.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name:  cckubernetesproduct-secret
data:
   db-connection-string-
secret: U2VydmVyPWNja3ViZXJuZXRlc3Byb2R1Y3QtbXNzcWwtc2VydmljZTtEYXRhYmFzZT1DQ0t1YmVybmV0ZXNQcm9kdWN0cztVc2VyPVNBO1Bhc3N3b3JkPTEyMz
Q1Njc4QWE7SW50ZWdyYXRlZCBTZWN1cml0eT1mYWxzZTtNdWx0aXBsZUFjdGl2ZVJlc3VsdFNldHM9dHJ1ZTs=
   db-password-secret: MTIzNDU2NzhBYQ==
type: Opaque
```

To create Pods with persistent storage, we used a StorageClass and PersistentVolumeClaim. The configuration is similar to a Deployment but provides predictable names for the pods and allows us to add persistent storage classes. This yaml file defines persistent storage that can be mounted in containers. Here, this includes the ReadWriteOnce access mode so that the volume can be mounted by a single container in read-write mode. We also apply a size limitation to 1GB. The general structure of the configuration is as follows:

cckubernetesproduct-mssql-persistent-volume.deployment.azure.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: cckubernetesproduct-mssql-persistent-volume
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: Managed
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cckubernetesproduct-mssql-persistent-volume-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: cckubernetesproduct-mssql-persistent-volume
spec:
  storageClassName: default
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

These are all configuration files used in this project. Now that we have the full definition of our application, all the particular servers can be deployed with the **kubectl apply** command specifying as a parameter the path of the YAML file which we have just created. For example, The following command is in use to deploy our web app,

-- kubectl apply cckubernetesproduct-mvc.deployment.azure.yaml

The Output shows the successfully created deployments and services.

3.6 Azure Kubernetes service:

AKS is one of many PaaS (Platform-as-a-Service) Azure services. It implies we won't have to worry about manually constructing, maintaining, and configuring virtual machines when we deploy a Kubernetes cluster [12].

We set up the Azure kubernetes service from the azure portal by selecting configuration from the web. For AKS it won't take that much time.


```
{
  "availabilityZones": [
    "1",
    "2",
    "3"
  ],
  "count": 2,
  "enableAutoScaling": false,
  "enableEncryptionAtHost": null,
  "enableFips": false,
  "enableNodePublicIp": null,
  "gpuInstanceProfile": null,
  "kubeletConfig": null,
  "kubeletDiskType": "OS",
  "linuxOsConfig": null,
  "maxCount": null,
  "maxPods": 110,
  "minCount": null,
  "mode": "System",
  "name": "agentpool",
  "nodeImageVersion": "AKSUbuntu-1804gen2c
  "nodeLabels": {},
  "nodePublicIpPrefixId": null,
  "nodeTaints": null,
  "orchestratorVersion": "1.19.11",
  "osDiskSizeGb": 128,
  "osDiskType": "Managed",
  "osSku": "Ubuntu",
  "osType": "Linux",
  "podSubnetId": null,
  "powerState": {
    "code": "Running"
  },
  "provisioningState": "Succeeded",
  "proximityPlacementGroupId": null,
  "scaleSetEvictionPolicy": null,
  "scaleSetPriority": null,
  "spotMaxPrice": null,
  "tags": null,
  "type": "VirtualMachineScaleSets",
  "upgradeSettings": null,
  "vmSize": "Standard_B2s"
}
```


Figure 11: AKS Configuration

3.7 Kubernetes Dashboard

The Kubernetes Dashboard is a web-based UI for Kubernetes clusters that may be used for various purposes. It enables users to administer and debug cluster-based applications and operate the cluster itself [7].

To deploy Dashboard, the following command has executed:

--kubectl apply –f https://raw.githubusercontent.com/kubernetes/dashboard/v2.1.0/aio/deploy/recommended.yaml

The command to Create a Sample User Account that can Access the Dashboard via Token is:

--kubectl apply -f https://gist.githubusercontent.com/dahlsailrunner/bbd453f3bb6259b66c08a70d0908283f /raw/ 5727723217e2df4b65d8933adf04d009cfb0fe3f/local-dashboard-account.yml

It is necessary to capture the Token. It's the token value returned by the command below.

PowerShell:

--kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | sls admin-user | ForEach-Object { $_ -Split '\s+' } | Select -First 1)

To create a secure route to the Kubernetes cluster from the local desktop to access Dashboard, the command below has executed:

--start kubectl proxy

3.8 Connect to AKS Dashboard and kubectl

The easiest way to connect to AKS cluster is to open the Azure dashboard and, in the Overview section, choose View Kubernetes dashboard [6]. From the right, a new panel with Azure CLI commands will appear.

To create the connection of the AKS cluster from a local machine, we follow some steps that set the --resource-group and the --name parameters with the specific values of our created cluster.

The following screenshot shows it:

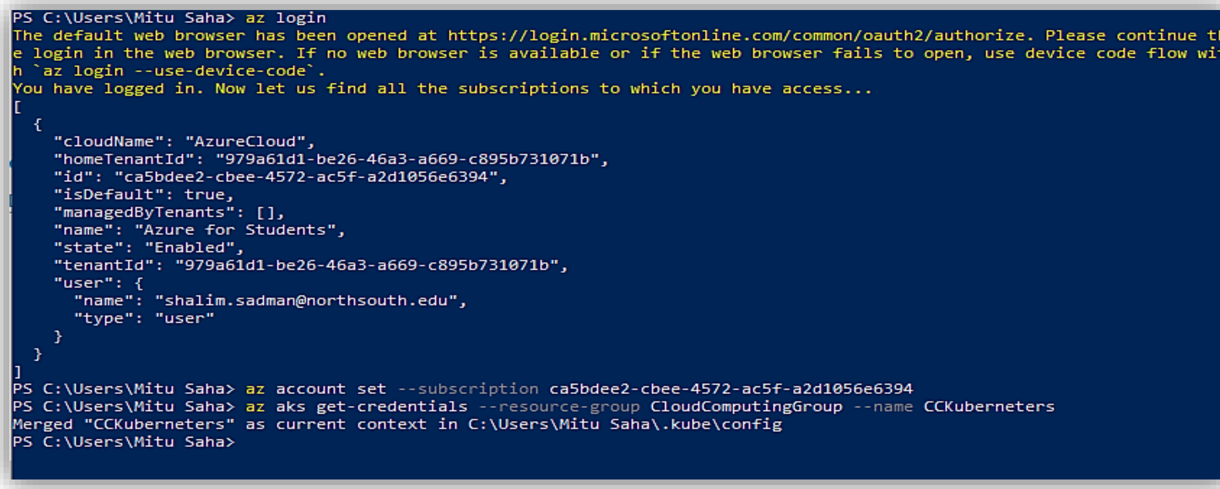


Figure 12: Azure commands

The command below is precisely what we need to see on the dashboard. It will start a proxy on your local system that will launch the dashboard with data from the AKS instance.

--az aks browse --resource-group CloudComputingGroup --name CCKuberneters

The dashboard provides information on the Kubernetes resources in cluster and any errors that may have occurred.

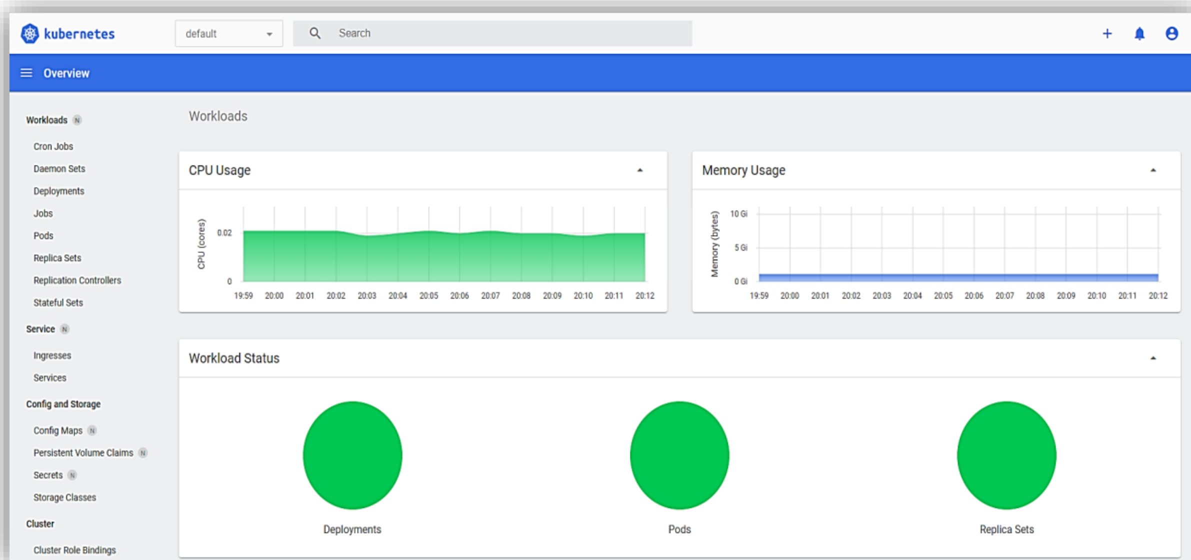
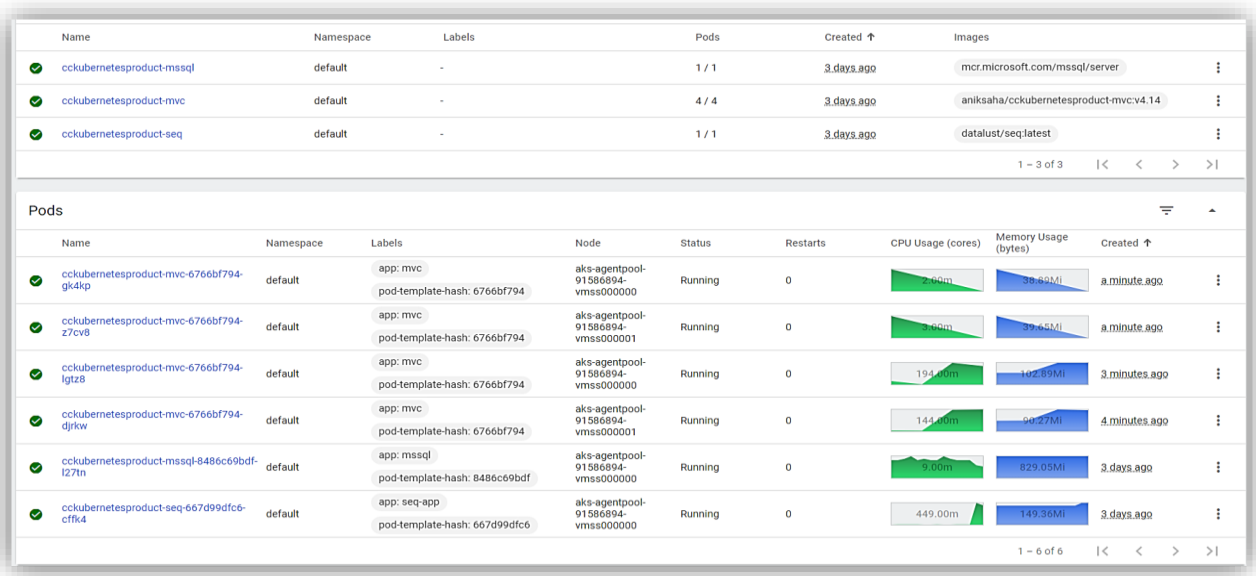


Figure 13: Kubernetes dashboard

The dashboard will review all the events that happened during the load test. In the Pods section, each pod shows the CPU spike it had:



Name	Namespace	Labels	Pods	Created	Images
cckubernetesproduct-mssql	default	-	1 / 1	3 days ago	mcr.microsoft.com/mssql/server
cckubernetesproduct-mvc	default	-	4 / 4	3 days ago	aniksaha/cckubernetesproduct-mvc:v4.14
cckubernetesproduct-seq	default	-	1 / 1	3 days ago	datalust/seq:latest

1 – 3 of 3

Pods

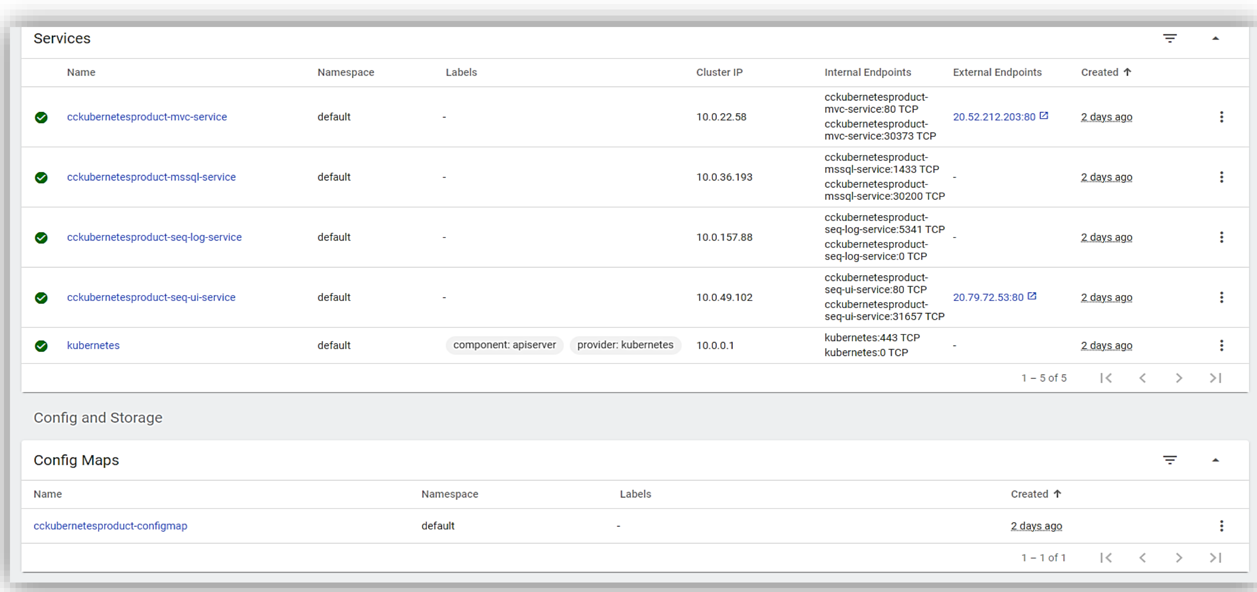
Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
cckubernetesproduct-mvc-6766bf794-gk4kp	default	app: mvc pod-template-hash: 6766bf794	aks-agentpool-91586894-vmss000000	Running	0	2.00m	38.89Mi	a minute ago
cckubernetesproduct-mvc-6766bf794-z7cv8	default	app: mvc pod-template-hash: 6766bf794	aks-agentpool-91586894-vmss000001	Running	0	3.00m	39.65Mi	a minute ago
cckubernetesproduct-mvc-6766bf794-lgtz8	default	app: mvc pod-template-hash: 6766bf794	aks-agentpool-91586894-vmss000000	Running	0	194.00m	102.89Mi	3 minutes ago
cckubernetesproduct-mvc-6766bf794-djrkw	default	app: mvc pod-template-hash: 6766bf794	aks-agentpool-91586894-vmss000001	Running	0	144.00m	90.27Mi	4 minutes ago
cckubernetesproduct-mssql-8486c69bdf-l27tn	default	app: mssql pod-template-hash: 8486c69bdf	aks-agentpool-91586894-vmss000000	Running	0	9.00m	829.05Mi	3 days ago
cckubernetesproduct-seq-667d99dfc6-cffk4	default	app: seq-app pod-template-hash: 667d99dfc6	aks-agentpool-91586894-vmss000000	Running	0	449.00m	149.36Mi	3 days ago

1 – 6 of 6

Figure 14: deployment and pods

As a consequence, three new replica pods are running for the cckubernetesproduct. mvc deployment.

The following figure shows all the services and configmap that has been used:



Services

Name	Namespace	Labels	Cluster IP	Internal Endpoints	External Endpoints	Created
cckubernetesproduct-mvc-service	default	-	10.0.22.58	cckubernetesproduct-mvc-service:80 TCP cckubernetesproduct-mvc-service:30373 TCP	20.52.212.203:80	2 days ago
cckubernetesproduct-mssql-service	default	-	10.0.36.193	cckubernetesproduct-mssql-service:1433 TCP cckubernetesproduct-mssql-service:30200 TCP	-	2 days ago
cckubernetesproduct-seq-log-service	default	-	10.0.157.88	cckubernetesproduct-seq-log-service:5341 TCP cckubernetesproduct-seq-log-service:0 TCP	-	2 days ago
cckubernetesproduct-seq-ui-service	default	-	10.0.49.102	cckubernetesproduct-seq-ui-service:80 TCP cckubernetesproduct-seq-ui-service:31657 TCP	20.79.72.53:80	2 days ago
kubernetes	default	component: apiserver provider: kubernetes	10.0.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	2 days ago

1 – 5 of 5

Config and Storage

Config Maps

Name	Namespace	Labels	Created
cckubernetesproduct-configmap	default	-	2 days ago

1 – 1 of 1

Figure 15: Services, Config, and Storage

The subsequent figures show the AKS dashboard view of persistent volumes, secret, storage classes, and Replicas.

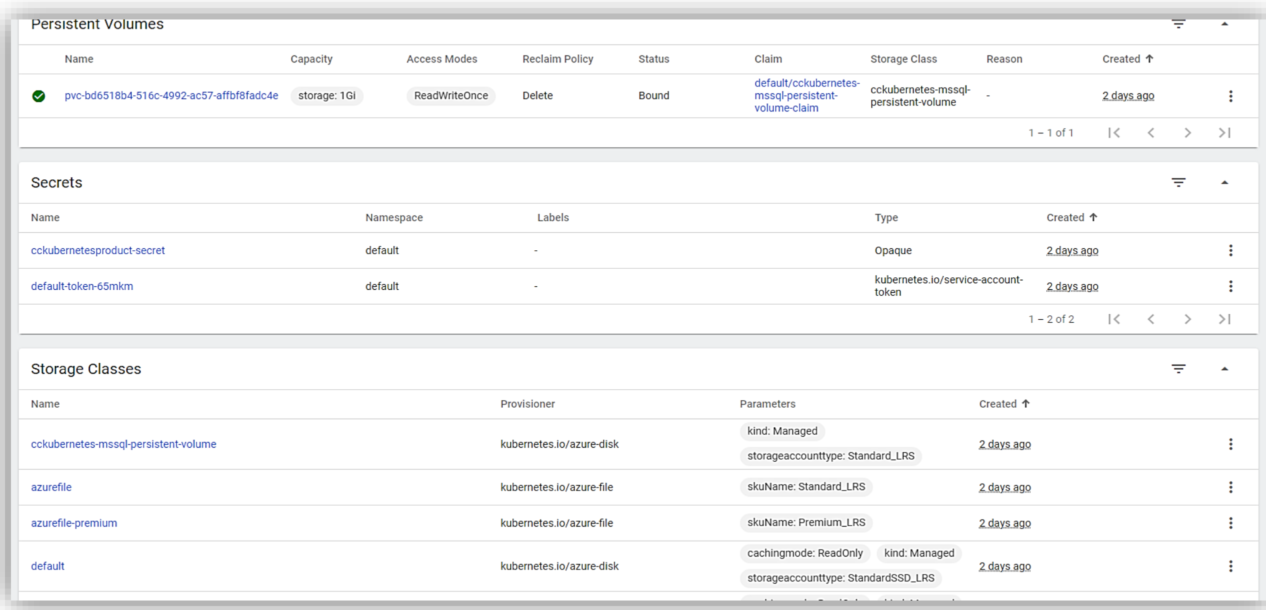


Figure 16: Persistent volumes, Secrets, Storage Classes

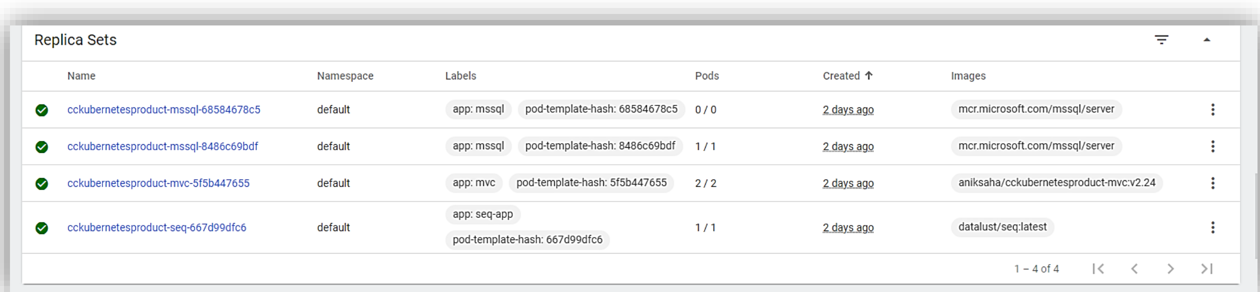


Figure 17: Replica Sets

3.9 AKS Cluster nodes

The following command provides us with pods, available services, deployments, and replica sets.

--kubectl get all

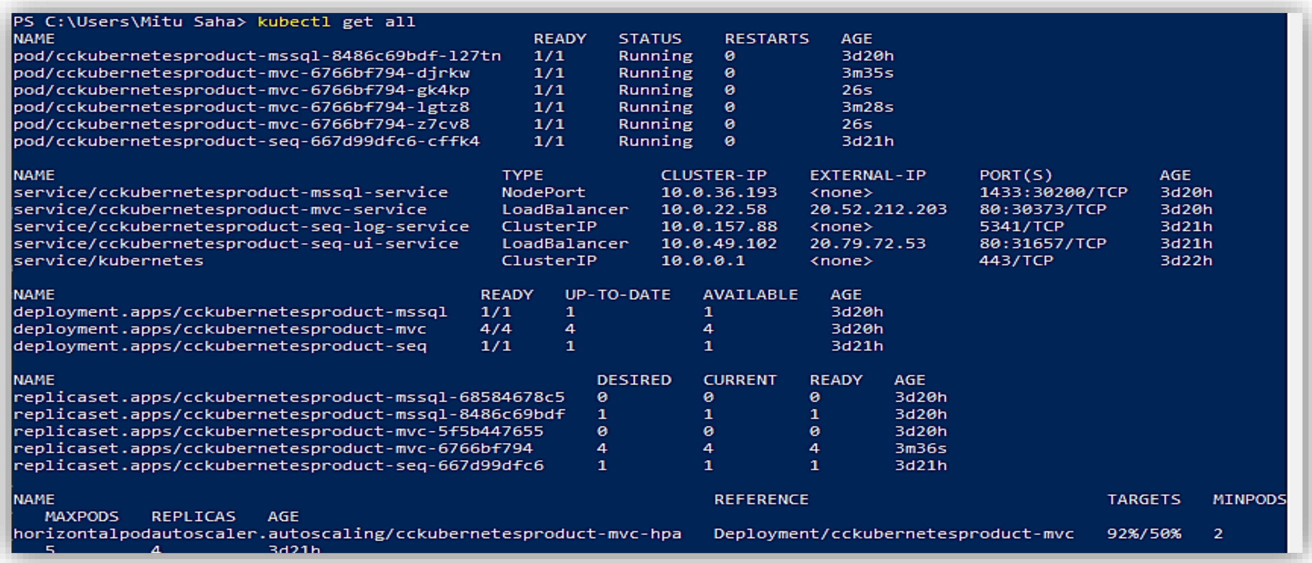


Figure 18: kubectl commands

Since the service type we have created is LoadBalancer, our application is exposed to the outside world through an external IP.

4. DevOps for Kubernetes using Azure DevOps

Microservices are growing increasingly popular in recent years. Kubernetes is where these microservices spend the majority of their time. A speedy and reliable deployment is a goal to achieve with microservices [11]. It will be demonstrated how to use Azure DevOps to set up a CI/CD pipeline to deploy the web app to a Kubernetes cluster.

4.1 Source control

We make a connection of our GitHub project with Azure DevOps.

4.2 Create a Build / Continuous Integration (CI) pipeline

The enabled Continuous integration lets the entire project work through a built-in Azure DevOps procedure (inside pipelines). Those steps include building and pushing images (in the docker hub), replacing build versions in yaml files, and copying and publishing artifacts at the end.

Here, for every new push in the Github repository, the continuous integration will fire up.

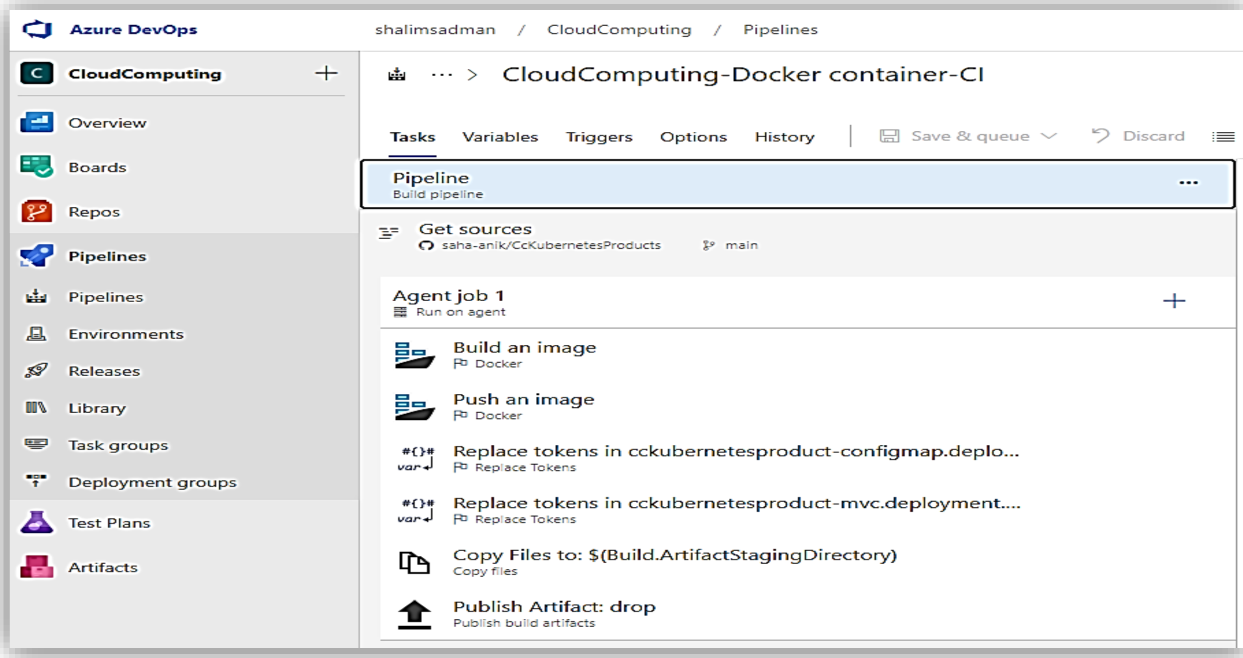


Figure 19: Continuous Integration (CI) pipeline

After running the web app pipeline, once the build process starts, the following build jobs occur in progress.

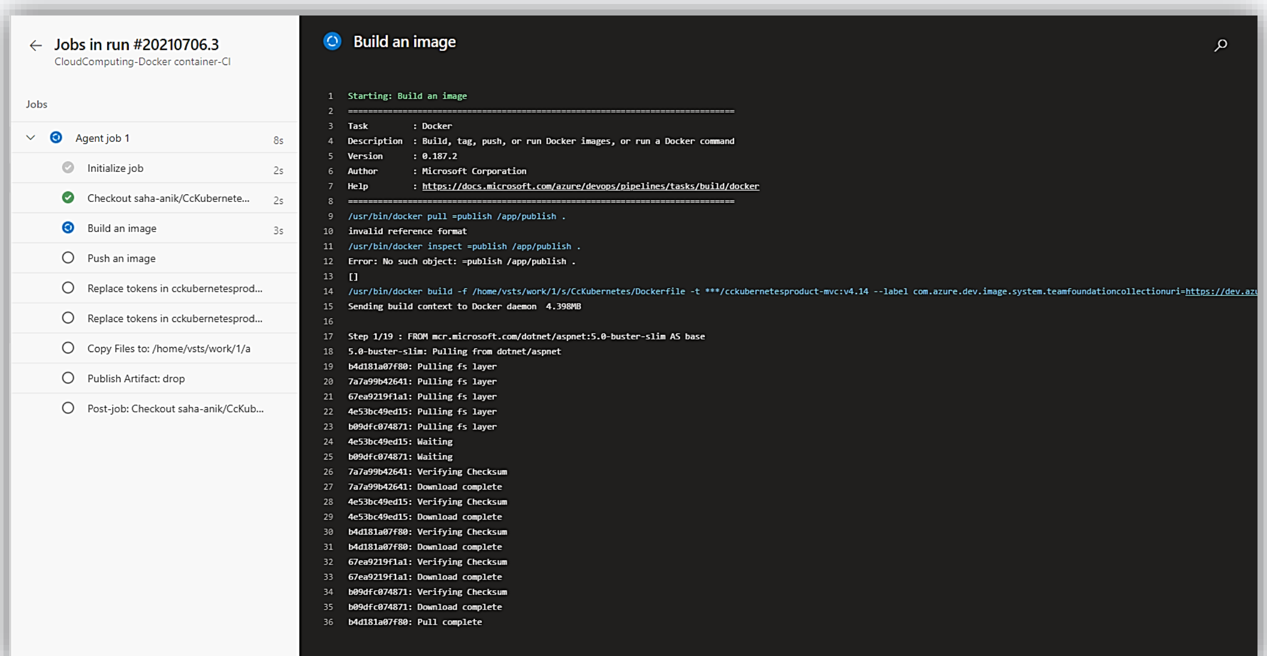


Figure 20: Progress jobs in Continuous Integration

These are the brief descriptions of all executed pipelines in DevOps.

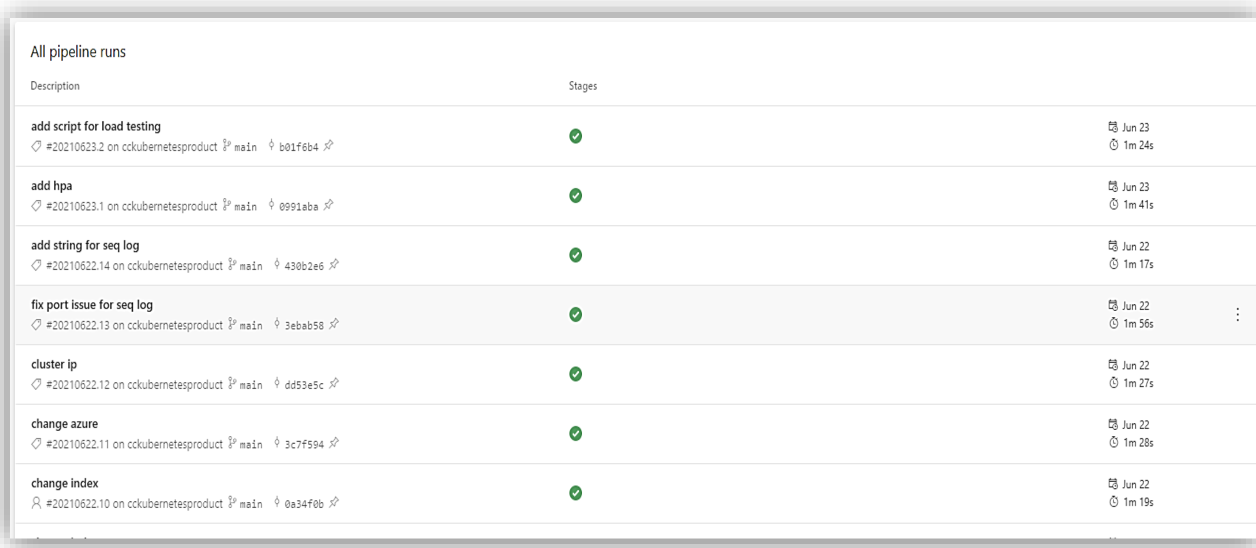


Figure 21: Executed pipeline in Continuous Integration

4.3 Create a Release / Continuous Delivery (CD) pipeline

In the release, we apply kubectl to configure and connect all the YAML files of k8s components with azure k8s. An Azure release pipeline is created for the mvc app to be able to deploy it via kubectl.
The **Releases** tab in the **Pipelines** section shows each command that connects azure pipelines to Kubernetes.

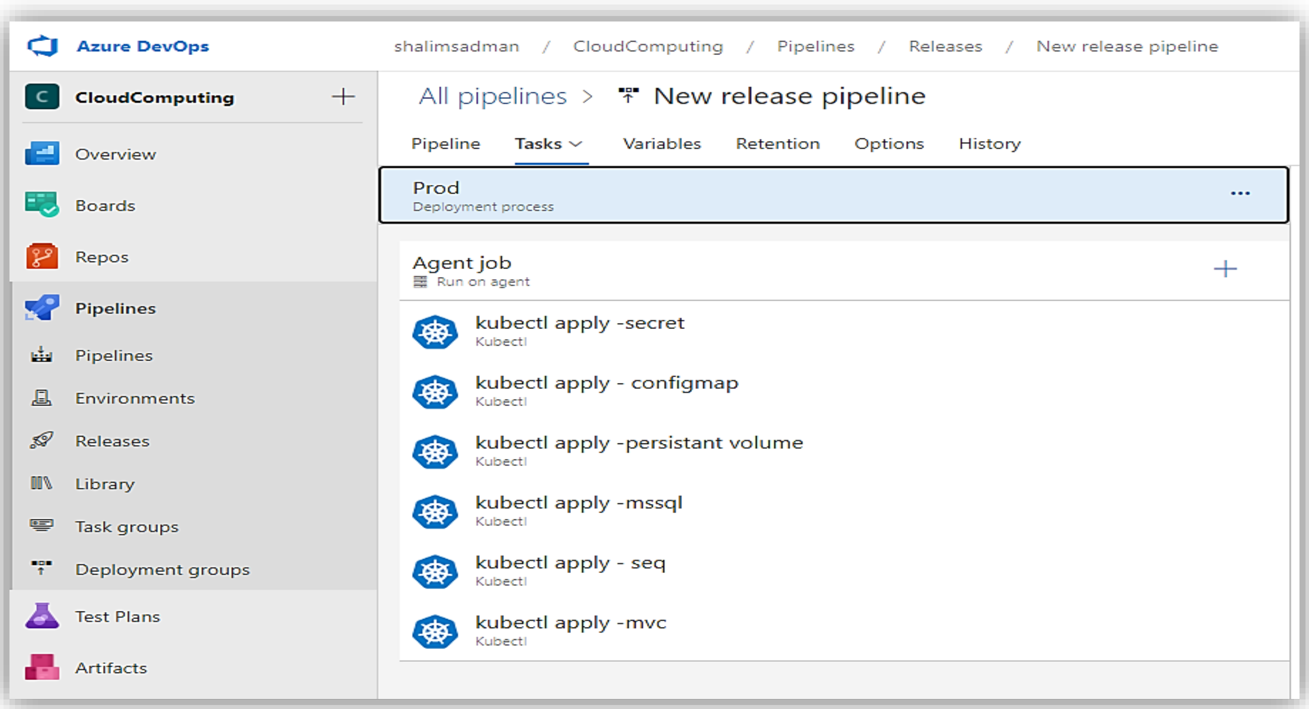


Figure 22: Release pipelines

The continuous delivery process will start after each successful continuous integration.

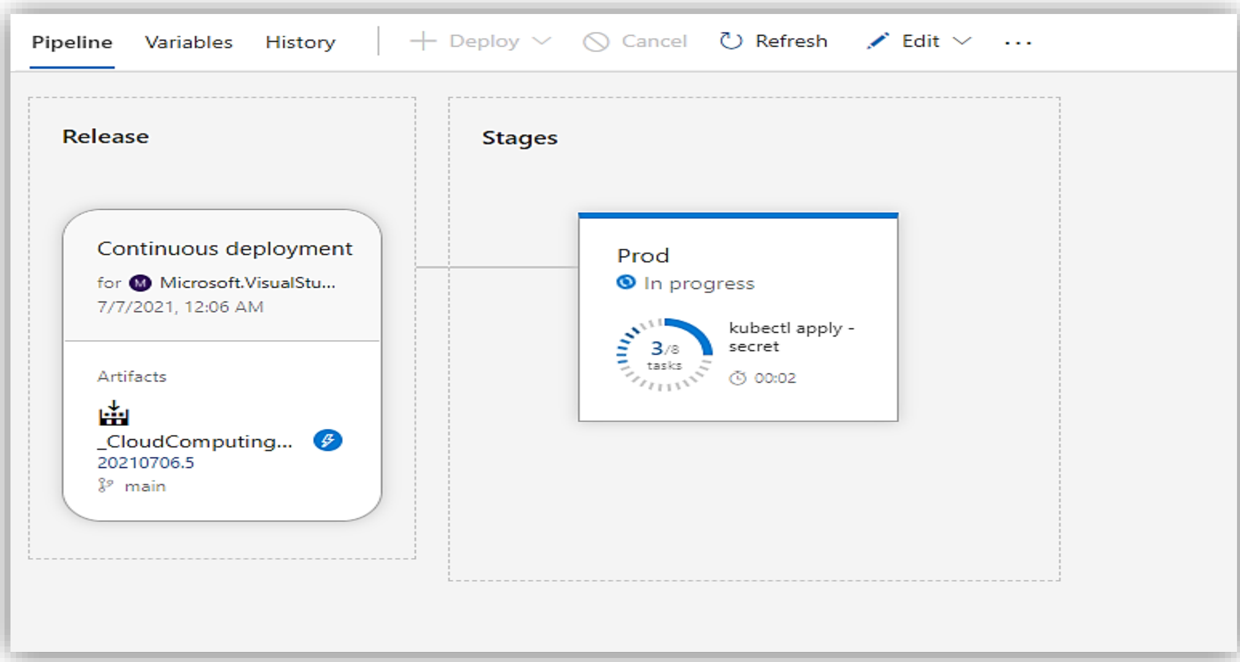


Figure 23: Continuous delivery

Finally, the web app has deployed along with MySQL and seq in a Kubernetes cluster by leveraging Azure DevOps, Azure Kubernetes Service, Docker.

5. Load Testing:

Load testing evaluates a software application's performance under a specific load. It determines how the software application operates when multiple users access it at the same time. Load testing is used to identify performance bottlenecks and ensure software applications' stability and smooth operation before deployment [13].

To test horizontal pod autoscaling, we use load testing. For our account online, we have two pods. We increase the number of pods when CPU utilization reaches 50%. Our web project has a maximum of 5 pods. We utilize the k6 framework for load testing because it is an open-source load testing tool. The k6 APIs are simple to use, versatile, and capable. Writing tests in JavaScript is similarly simple [14].

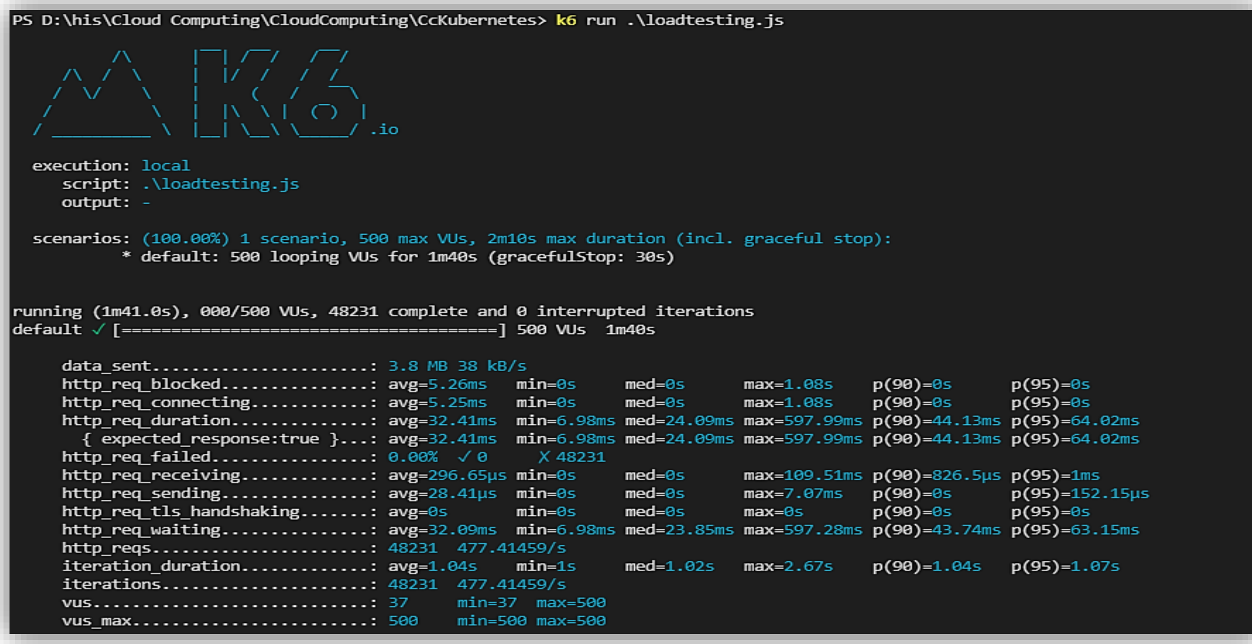


Figure 24: Iterations after executing load testing

our load testing js script is as follows:

```
import http from 'k6/http';
import { sleep } from 'k6';
export let options = {
  vus: 200,
  duration: '60s',
};
export default function () {
  http.get('http://20.79.117.111/Products');
  sleep(1);
}
```

6. Conclusion & future work:

With the introduction of the cloud, the computing world has forever changed. Cloud computing provides developers with rapid, low-cost access to infrastructure at nearly unlimited scales. Because of the cloud's agility and high availability, monolithic architectures have been stressed, resulting in the growth of microservices-based systems. In this paper, we've covered the fundamentals of microservices, Kubernetes, and Azure DevOps, as well as how to use them to deploy a simple web application. It contains detailed instructions for configuring and installing the required tools and services. Finally, we went over how to get to the AKS cluster's Kubernetes web interface, test deployments, and represent the deployment results. We also integrated the CI/CD Pipeline with Azure DevOps. To ensure that our deployment was stable, we performed load and log testing.

With all of the items that must be handled, such as ConfigMaps, services, pods, and Persistent Volumes, as well as the number of releases that must be maintained, the system we worked on might be quite complex. As a result, we'd like to use Helm to parameterize our YAML in the future. It's a package manager that makes application deployment simple, standardized, and reusable, boosting developer productivity, reducing deployment complexity, improving operational readiness, and accelerating the adoption of cloud-native apps.

7. Reference:

1. Kubernetes (K8s). URL: https://github.com/kubernetes/kubernetes (visited on 27/06/2021).
2. WHAT IS KUBERNETES? URL: https://kemptechnologies.com/blog/what-is-kubernetes/?fbclid=IwAR07gKxCVy1LgYqJvTfPMhYy2tZHlniHi625vyWi9T7xLZYsW3n28sbiY4g (visited on 08/07/2021)
3. Azure Kubernetes Service (AKS). URL: https://azure.microsoft.com/en-us/services/kubernetes-service/#features (visited on 01/07/2021).
4. Developing Microservices Architecture on Microsoft Azure with Open Source Technologies. URL: https://www.microsoftpressstore.com/store/developing-microservices-architecture-on-microsoft-9780136819387?fbclid=IwAR0eC35v3h0_QnH9bQ-Uf2guYMZ7xHryjoRbk1VX-6haaR-RoBvEuMel-sA (visited on 30/06/2021).
5. Kubernetes vs. Docker. URL: https://azure.microsoft.com/en-us/topic/kubernetes-vs-docker/?fbclid=IwAR2V4Eyvs6bl4DdkoqqXYdZHKFxOGmrQUSUeWIabw4hyfewujUF_RNvPFVE (visited on 01/07/2021).
6. First steps with Docker and Kubernetes - Moving to Azure Kubernetes Service. URL: https://techcommunity.microsoft.com/t5/windows-dev-appconsult/first-steps-with-docker-and-kubernetes-moving-to-azure/ba-p/360010 (visited on 02/07/2021).
7. Using the K8s Dashboard Locally. URL: https://gist.github.com/dahlsailrunner/1a47b0e38f6e3ba64d4d61835c73b7e2?fbclid=IwAR1d-R9HCgnnL9By9WHDzCbUKhXHhcFWFN-Z7OZhS-cbelXWP6vwXrBEjZc (visited on 03/07/2021).
8. How-To deploy Docker images to Azure Kubernetes Services (AKS). URL: https://purple.telstra.com.au/blog/how-to-deploy-docker-images-to-azure-kubernetes-services-aks (visited on 02/07/2021).
9. SEQ, Machine data, for humans. URL: https://datalust.co/seq (visited on 02/07/2021).
10. Kubernetes vs Docker in 2020. URL: https://technofaq.org/posts/2020/06/kubernetes-vs-docker-in-2020/?fbclid=IwAR11bbElLMFrCA_PDCvJ4HJQEjz1i5C5lGkLUxZ-FOHh7SC33l65yRxqG3Q (Visited on 08/07/2021)
11. Using Azure DevOps to setup a CI/CD pipeline and deploy to Kubernetes. URL: https://cloudblogs.microsoft.com/opensource/2018/11/27/tutorial-azure-devops-setup-cicd-pipeline-kubernetes-docker-helm/ (visited on 01/07/2021).
12. Deploying a multi-container application to Azure Kubernetes Services. URL: https://azuredevopslabs.com/labs/vstsextend/kubernetes/#:~:text=Azure%20Kubernetes%20Service%20(AKS)%20is,applications%20without%20container%20orchestration%20expertise (visited on 28/06/2021).
13. Load Testing Tutorial: What is? How to? URL: https://www.guru99.com/load-testing-tutorial.html (visited on 02/07/2021).
14. The best developer experience for load testing. URL: https://k6.io/ (visited on 02/07/2021).
15. Microservices with Azure Kubernetes and Docker. URL: https://medium.com/@sumindaniro/microservices-with-azure-kubernetes-and-docker-49de617f0341 (visited on 04/07/2021).
16. Introduction to Docker and Kubernetes on Azure. URL: https://dzone.com/articles/introduction-docker-and (visited on 03/07/2021).
17. Build and deploy a microservice with Kubernetes. URL: https://searchitoperations-techtarget-com.cdn.ampproject.org/v/s/searchitoperations.techtarget.com/tutorial/Build-and-deploy-a-microservice-with-Kubernetes?amp_js_v=a6&amp_gsa=1&amp=1&usqp=mq331AQKKAFQArABIIACAw%3D%3D#aoh=16256498080184&amp_ct=1625650078960&referrer=https%3A%2F%2Fwww.google.com&amp_tf=From%20%251%24s&ampshare=https%3A%2F%2Fsearchitoperations.techtarget.com%2Ftutorial%2FBuild-and-deploy-a-microservice-with-Kubernetes (visited on 04/07/2021).