# 

|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| **[**Infra\_Optimization**]**  **[Project 3]** |
| |  |  |  | | --- | --- | --- | | **Joshua Demebo** | N**ov 12th, 2022.** | [ **PG DO DevOps Capstone Project** | |

**Project 3**

# DESCRIPTION

Create a DevOps infrastructure for an e-commerce application to run on high-availability mode.

Background of the problem statement:

A popular payment application, EasyPay where users add money to their wallet accounts, faces an issue in its payment success rate. The timeout that occurs with

the connectivity of the database has been the reason for the issue.

While troubleshooting, it is found that the database server has several downtime instances at irregular intervals. This situation compels the company to create their own infrastructure that runs in high-availability mode.

Given that online shopping experiences continue to evolve as per customer expectations, the developers are driven to make their app more reliable, fast, and secure for improving the performance of the current system.

**Implementation requirements:**

1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)
2. Automate the provisioning of an EC2 instance using Ansible or Chef Puppet
3. Install Docker and Kubernetes on the cluster
4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod
5. Create a new user with permissions to create, list, get, update, and delete pods
6. Configure application on the pod
7. Take snapshot of ETCD database
8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured

**The following tools must be used:**

1. EC2
2. Kubernetes
3. Docker
4. Ansible or Chef or Puppet

**The following things to be kept in check:**

1. You need to document the steps and write the algorithms in them.
2. The submission of your GitHub repository link is mandatory. In order to track the tasks, you need to share the link of the repository.
3. Document the step-by-step process starting from creating test cases, then executing them, and recording the results.
4. You need to submit the final specification document, which includes:

* Project and tester details
* Concepts used in the project
* Links to the GitHub repository to verify the project completion
* Your conclusion on enhancing the application and defining the USPs (Unique Selling Points

**Table of Contents**

[0](#_Toc119242966)

[DESCRIPTION 1](#_Toc119242967)

[**EasyPay App Architecture Diagram** 3](#_Toc119242968)

[**Introduction** 4](#_Toc119242969)

[**The following specific tools were used** 4](#_Toc119242970)

[**1.** **Automating the provisioning of an Azure VM using Terraform. Created using local git and using VS Code to edit the script.** 5](#_Toc119242971)

[ **Provider.tf for provisioning a VM in Azure** 5](#_Toc119242972)

[ **Main.tf for provisioning a VM in Azure** 6](#_Toc119242973)

[ **A provisioned VM in Azure (hostname easypay-app01)** 8](#_Toc119242974)

[**2.** **Creating a new app in Visual Studio as an ASP.NET Core Web App, enable Docker support, and make a small change to it. Ensure it builds and runs locally.** 9](#_Toc119242975)

[ **Naming the project as EasyPay** 9](#_Toc119242976)

[ **Selecting ".NET 6.0 (Long-term support)" as the language, "Enable Docker" to enable container support, and "Linux" as the container language**. 10](#_Toc119242977)

[ **EasyPay application created** 10](#_Toc119242978)

[ **Modify this page by changing the text on the screen to have a visual look of the app.** 11](#_Toc119242979)

[**App was Built successfully**. 11](#_Toc119242980)

[ **Test application on Azure VM. Before deploying this solution to Azure, it will be nice to test the application on the tenant VM as a web app.** 11](#_Toc119242981)

[3. The application will be published from Visual Studio to a new Azure Container Registry. A new Azure Container Registry will be created using Visual Studio. Under Build menu, choose Publish. And Choose Azure as the Target. Click Next. 12](#_Toc119242982)

[ **Select Azure Container Registry. Click Next.** 12](#_Toc119242983)

[ **Signing in with my credential** 13](#_Toc119242984)

[ **Create a new Azure Container Registry using same Resource Group as infra\_optimization** 13](#_Toc119242985)

[ **Click Publish. Monitor the output console for messages.** 14](#_Toc119242986)

[ **Successfully pushed to Docker Hub/Desktop and in Visual Studio** 15](#_Toc119242987)

[4. Validate Azure Container Registry 16](#_Toc119242988)

[5. Create a **Kubernetes** on the cluster to run on high-availability (HA) mode 17](#_Toc119242989)

[ **Integrating Container Registry** 18](#_Toc119242990)

[ **Completion of Kubernetes Cluster** 18](#_Toc119242991)

[**6.** **Deploy the Container Image to the AKS (Azure Kubernetes Services) Cluster** 19](#_Toc119242992)

[ **Open the Azure Cloud Shell** 19](#_Toc119242993)

[ **Type “kubectl get nodes” to see the running nodes.** 19](#_Toc119242994)

[ **Create all yaml files and upload to the /home/joshua directory** 20](#_Toc119242995)

[ **Create a custom Namespace “infra-optimize-ns”** 20](#_Toc119242996)

[ **Deployment of “easypay20221111100704.azurecr.io/easypay:latest” custom image and LoadBalancer service also in custom namespace “infra-optimize-ns”apiVersion: apps/v1** 20](#_Toc119242997)

[ **Make a note of the public IP address under “external-IP <20.104.142.166>” to be use** 21](#_Toc119242998)

[ **Copy the IP address to your clipboard. Open a browser and go to that IP address, see that your application is successfully running in Azure Kubernetes Service.** 22](#_Toc119242999)

[7. Implement the **network policies** at the **database** pod to allow ingress traffic from the front-end **application pod** 22](#_Toc119243000)

[ **Network Policy created. “kubectl get networkpolicy --namespace=infra-optimize-ns”** 22](#_Toc119243001)

[ **Created a postgresql db pod and redis caching to custom namespace=infra-optimize-ns”** 22](#_Toc119243002)

[ **Create a new user (joshua-demebo) with permissions to create, list, get, update, and delete pods** 23](#_Toc119243003)

[ **Create a Role Binding** 24](#_Toc119243004)

[ **Created a Redis for caching** 25](#_Toc119243005)

[ **Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured** 25](#_Toc119243006)

[8. Take snapshot of ETCD database 26](#_Toc119243007)

[ **kubectl get all -o wide** 26](#_Toc119243008)

[ **kubectl get all -o wide** 26](#_Toc119243009)

[**Conclusion** 26](#_Toc119243010)

# **EasyPay App Architecture Diagram**

# **Introduction**

This documentation contains all steps and procedures in successfully managing a DevOps infrastructure for an eCommence application (EasyPay) to run on a high-availability state. The automating of using configuration management, integrated development environment (IDE), infrastructure as code, yaml files to be edited and reuse as desire, also tools used were Docker hub/desktop, Azure Container Registry, and Azure Kubernetes Services, by doing so will help the DevOps team to optimize and autoscaling either vertically or horizontally. Source codes/scripts are kept in the official version control tool. <https://github.com/joshua-demebo/infra_optimization>

## **The following specific tools were used**

1. **Terraform** is used primarily to automate various infrastructure tasks like provisioning the VM in Microsoft Azure, used to test EasyPay app as a localhost, which describes the complete infrastructure as a form of code
2. **Visual Studio** provides a rich support for various language like JavaScript, ASP.Net etc. Is an open-source language often used in large size web app development, it was used to develop the EasyPay app.
3. **Microsoft Azure** a cloud computing platform with solutions including Infrastructure as a Service (IaaS), Platform as a Service and Software as a Service (SaaS) that can be used for services such as analytics, virtual computing, storage, networking, and much more. This was used with various services below:

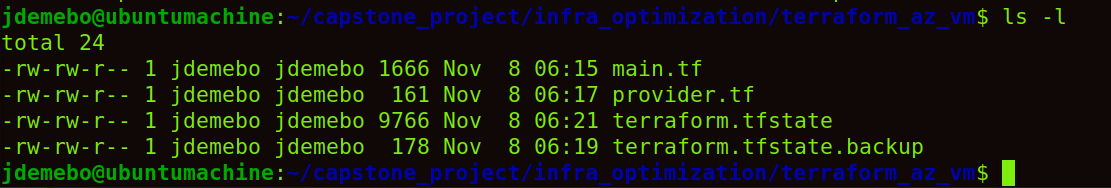
* **Azure Container Registry** **(ACR)** was used to build, store, and manage container images and artifacts in a private registry for Docker container deployment.
* **Azure Kubernetes Service** (**AKS)** simplifies the deployment of managing Kubernetes cluster in Azure by offloading the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance. This was used to provision clusters, nodes, pods to integrating with Docker custom image to build/deploy EasyPay app with a high-available, vertical and horizontal autoscaling etc.

1. **Docker** is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments. To run our EasyPay application in a docker container, a customized docker image was created. This customized docker image includes instructions that install specific packages and copy the code into the docker container and was Publish to ACR via Visual Studio.
2. **GitHub** a version control system that lets you manage and keep track of source code history. GitHub is a cloud-based hosting service that lets you manage Git repositories. If you have open-source projects that use Git, then GitHub is designed to help manage them. All source code like Terraform, yaml etc, were stored in GitHub.

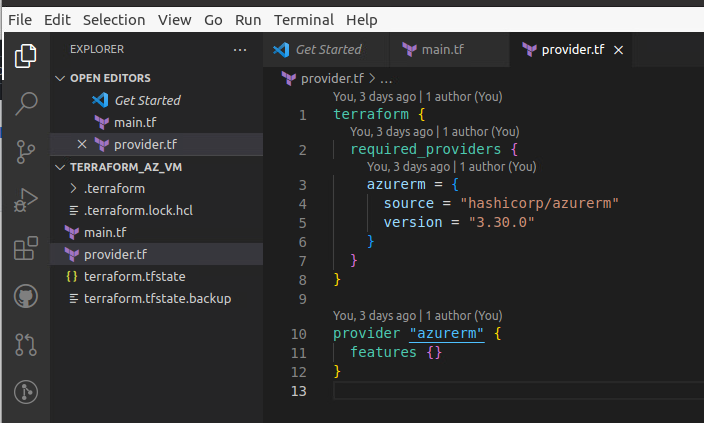
Finally, some tools like Database (Redis and PostgreSQL) were used to enhance the database performance and caching of EasyPay app, a network policy including LoadBalancer to prevent connectivity downtime.

This infra optimization will make the app and give an online shopper more reliable, fast, and secure improved performance.

# **Automating the provisioning of an Azure VM using Terraform. Created using local git and using VS Code to edit the script.**



## **Provider.tf for provisioning a VM in Azure**



terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "3.30.0"

}

}

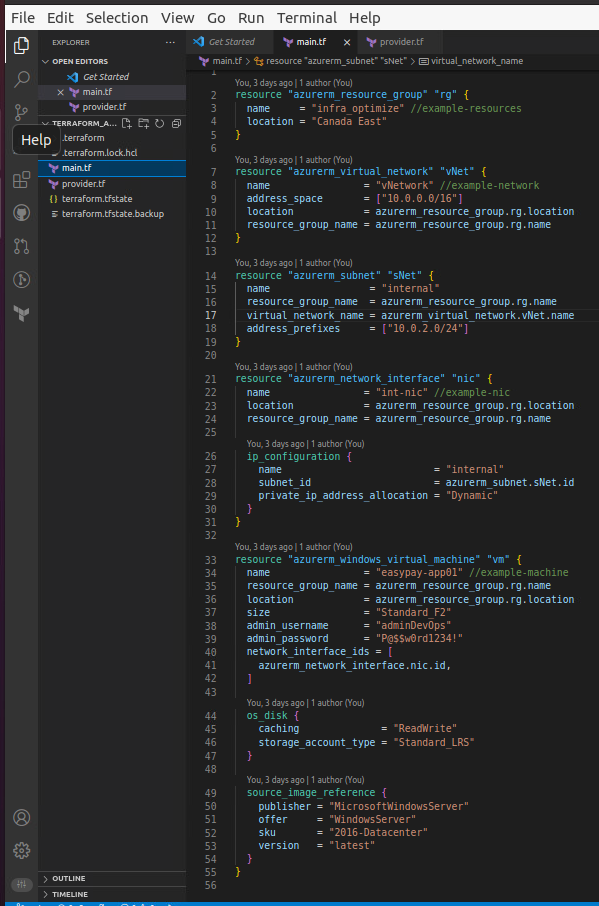
}

provider "azurerm" {

features {}

}

## **Main.tf for provisioning a VM in Azure**



resource "azurerm\_resource\_group" "rg" {

  name     = "infra\_optimize" //example-resources

  location = "Canada East"

}

resource "azurerm\_virtual\_network" "vNet" {

  name                = "vNetwork" //example-network

  address\_space       = ["10.0.0.0/16"]

  location            = azurerm\_resource\_group.rg.location

  resource\_group\_name = azurerm\_resource\_group.rg.name

}

resource "azurerm\_subnet" "sNet" {

  name                 = "internal"

  resource\_group\_name  = azurerm\_resource\_group.rg.name

  virtual\_network\_name = azurerm\_virtual\_network.vNet.name

  address\_prefixes     = ["10.0.2.0/24"]

}

resource "azurerm\_network\_interface" "nic" {

  name                = "int-nic" //example-nic

  location            = azurerm\_resource\_group.rg.location

  resource\_group\_name = azurerm\_resource\_group.rg.name

  ip\_configuration {

    name                          = "internal"

    subnet\_id                     = azurerm\_subnet.sNet.id

    private\_ip\_address\_allocation = "Dynamic"

  }

}

resource "azurerm\_windows\_virtual\_machine" "vm" {

  name                = "easypay-app01" //example-machine

  resource\_group\_name = azurerm\_resource\_group.rg.name

  location            = azurerm\_resource\_group.rg.location

  size                = "Standard\_F2"

  admin\_username      = "adminDevOps"

  admin\_password      = "P@$$w0rd1234!"

  network\_interface\_ids = [

    azurerm\_network\_interface.nic.id,

  ]

  os\_disk {

    caching              = "ReadWrite"

    storage\_account\_type = "Standard\_LRS"

  }

  source\_image\_reference {

    publisher = "MicrosoftWindowsServer"

    offer     = "WindowsServer"

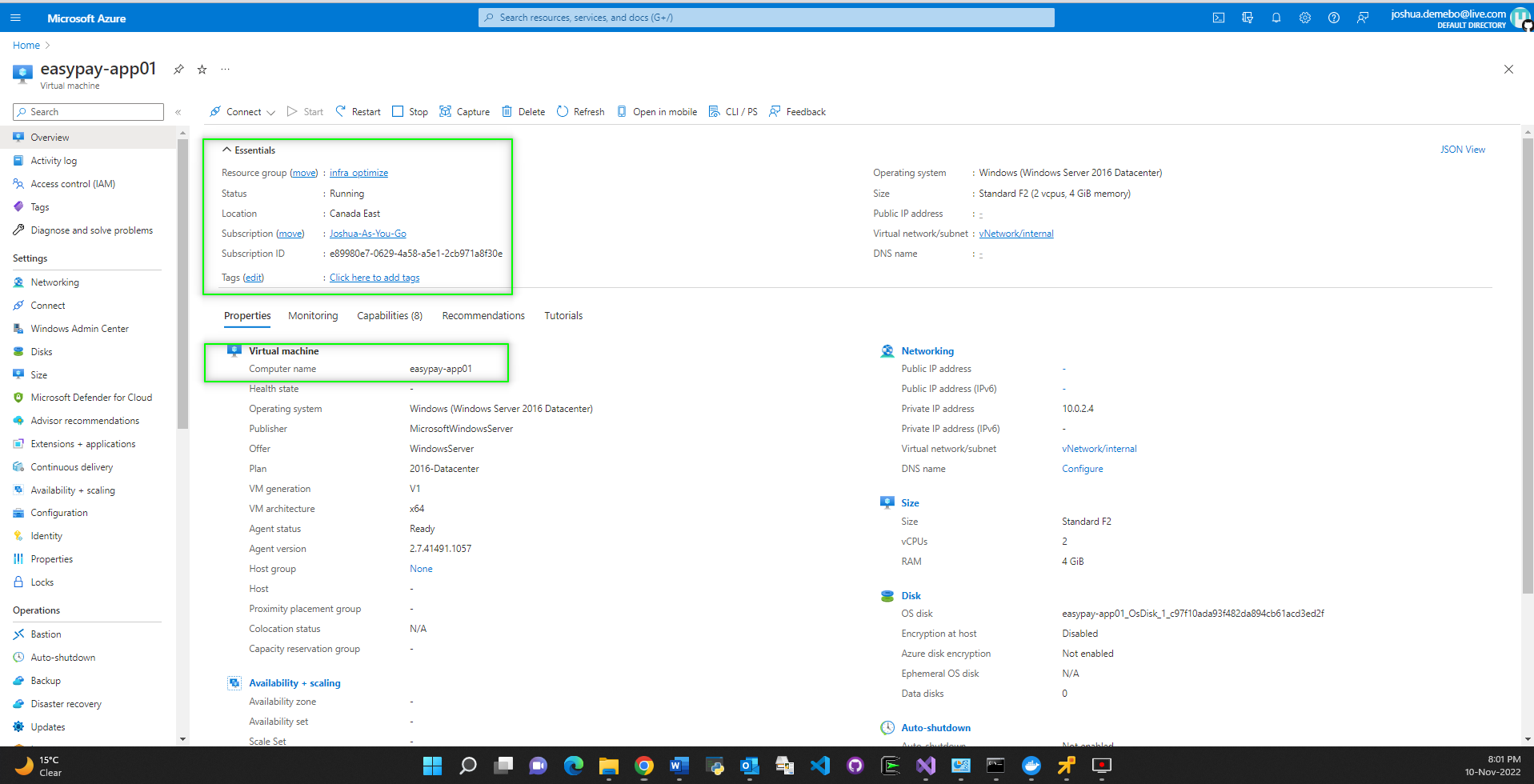
    sku       = "2016-Datacenter"

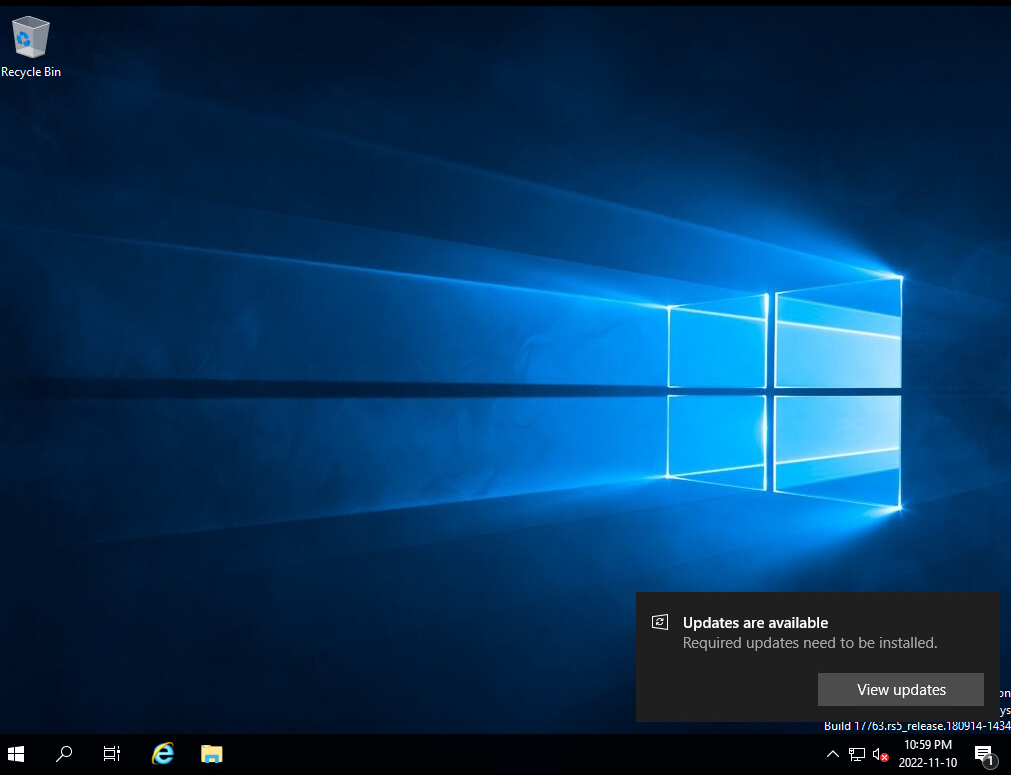
    version   = "latest"

  }

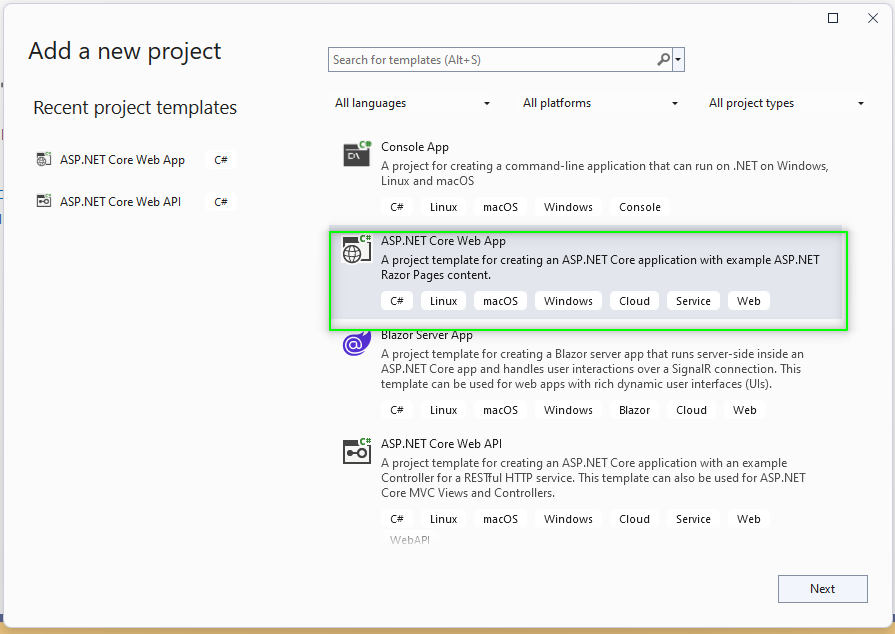
}

## **A provisioned VM in Azure (hostname easypay-app01)**

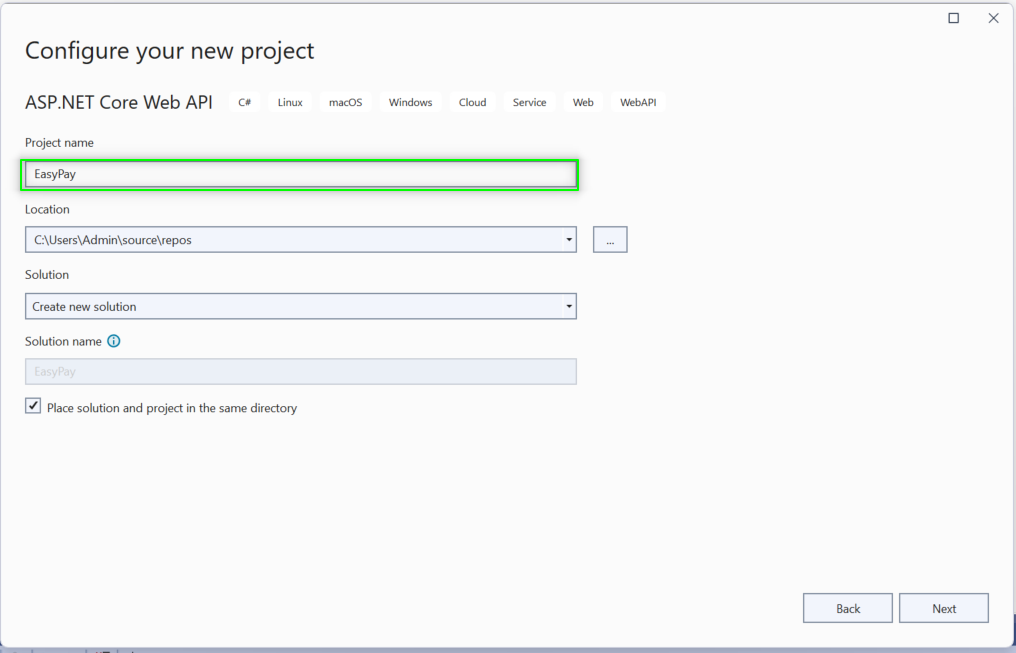




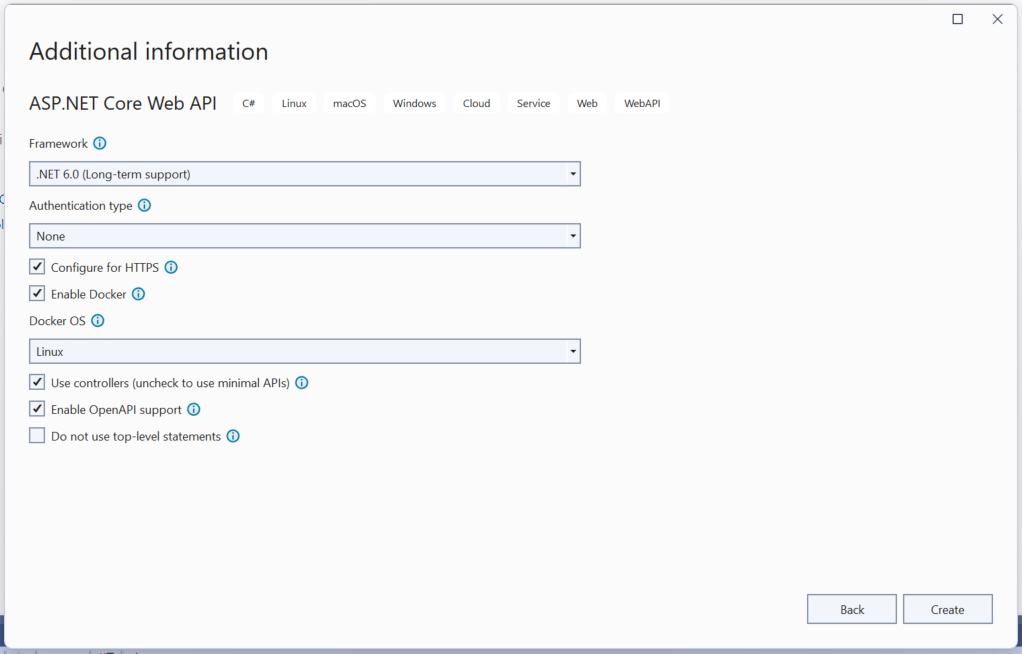
# **Creating a new app in Visual Studio as an ASP.NET Core Web App, enable Docker support, and make a small change to it. Ensure it builds and runs locally.**



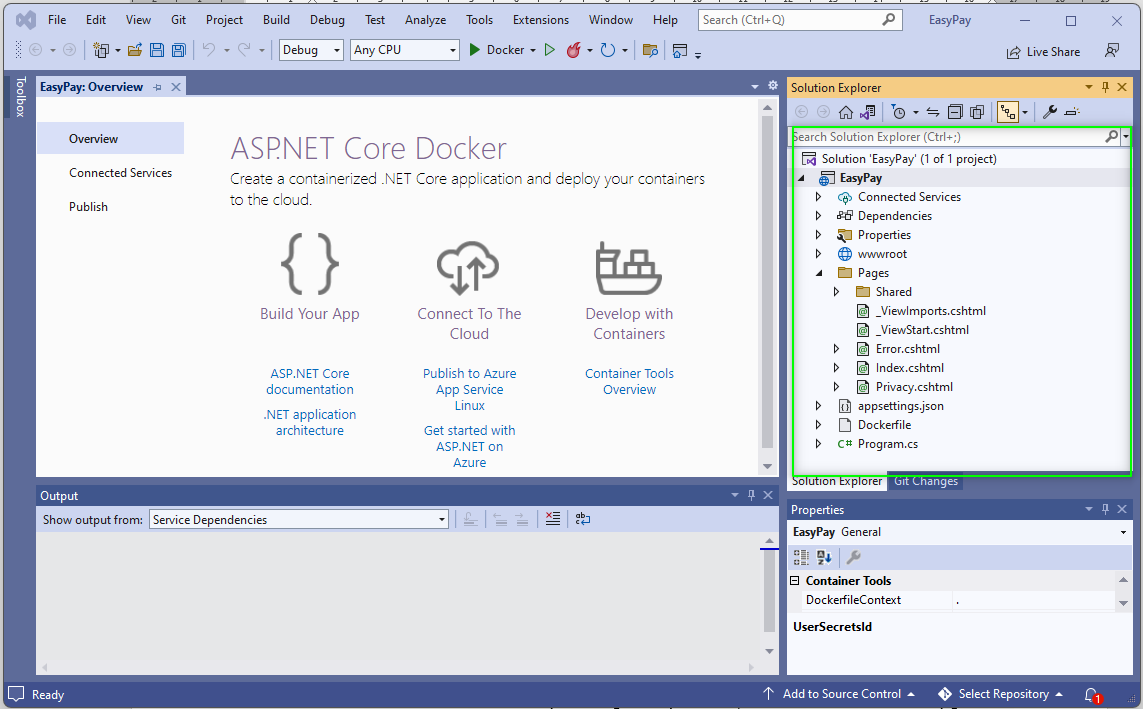
## **Naming the project as EasyPay**



## **Selecting ".NET 6.0 (Long-term support)" as the language, "Enable Docker" to enable container support, and "Linux" as the container language**.

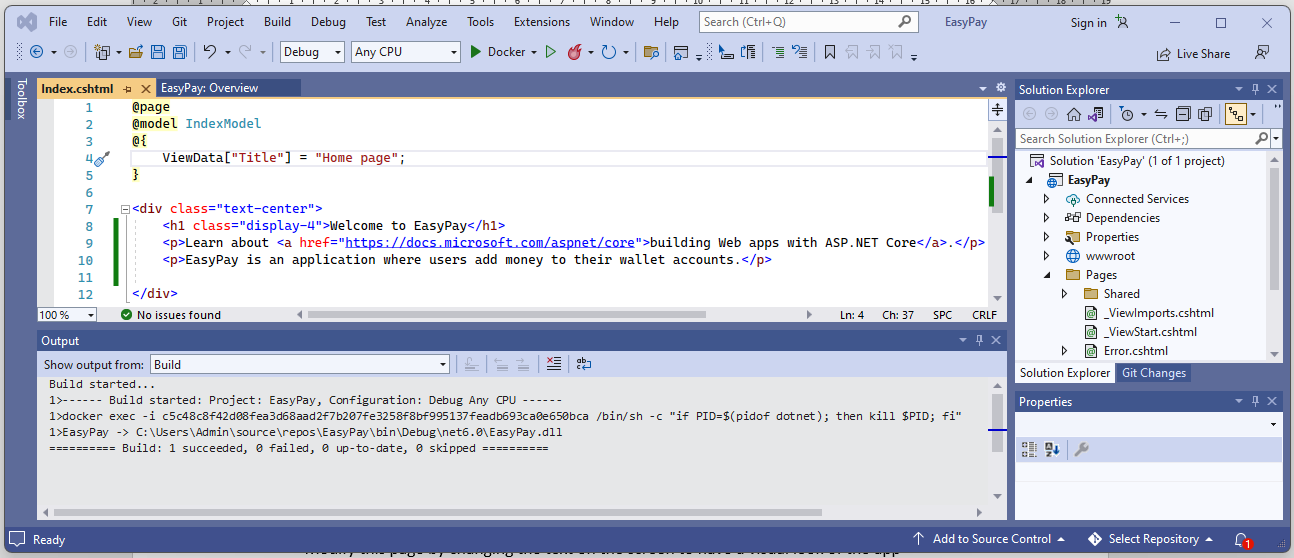


## **EasyPay application created**

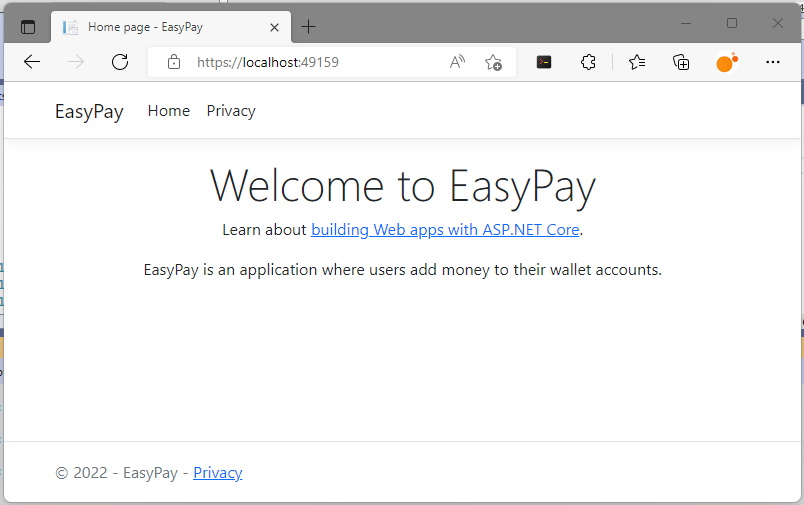


## **Modify this page by changing the text on the screen to have a visual look of the app.**

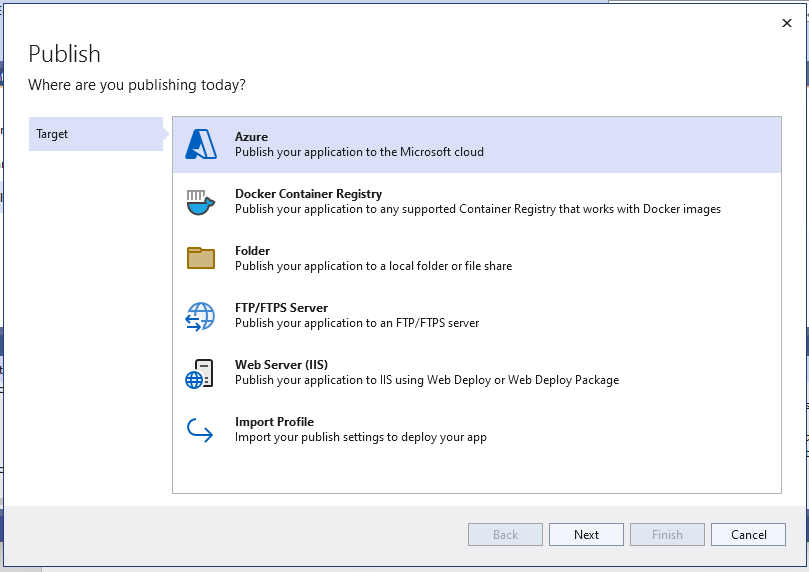
## **App was Built successfully**.



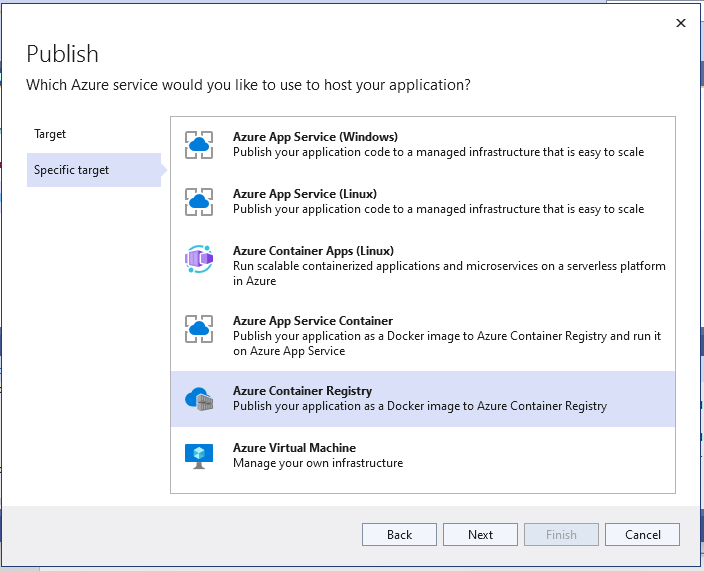
## **Test application on Azure VM. Before deploying this solution to Azure, it will be nice to test the application on the tenant VM as a web app.**



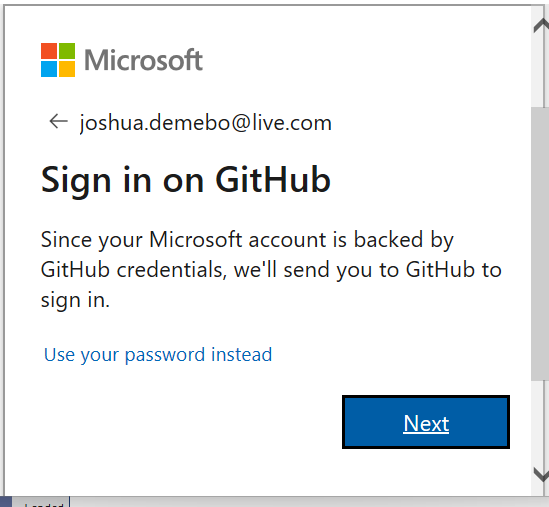
# The application will be published from Visual Studio to a new Azure Container Registry. A new Azure Container Registry will be created using Visual Studio. Under Build menu, choose Publish. And Choose Azure as the Target. Click Next.



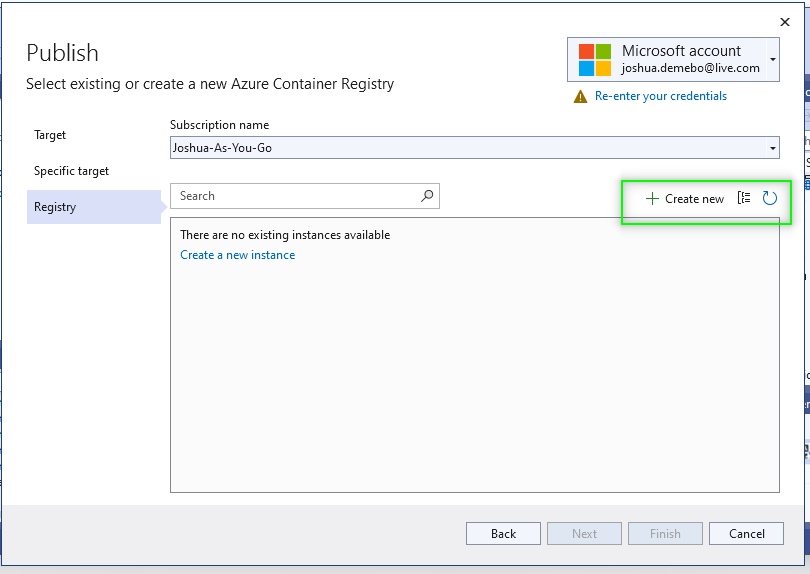
## **Select Azure Container Registry. Click Next.**

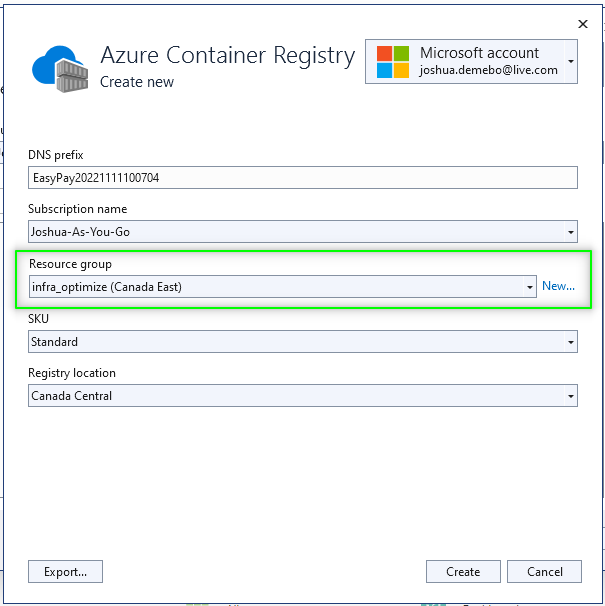


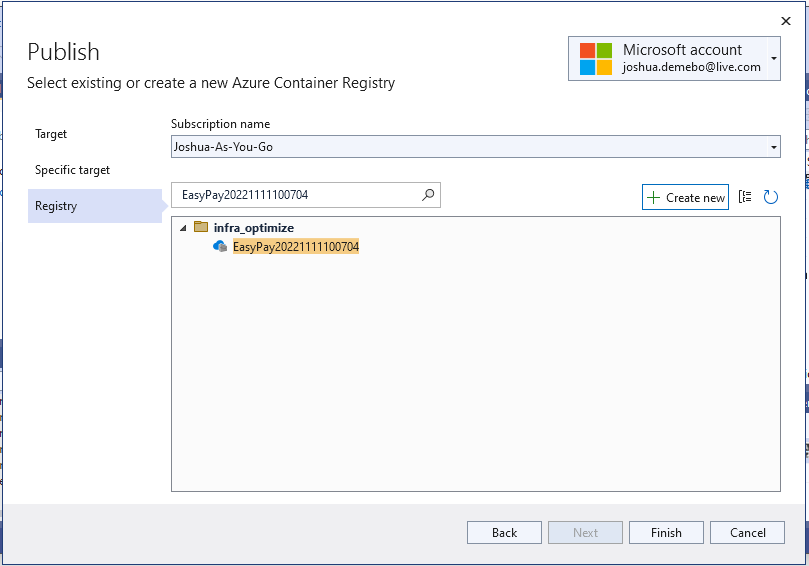
## **Signing in with my credential**



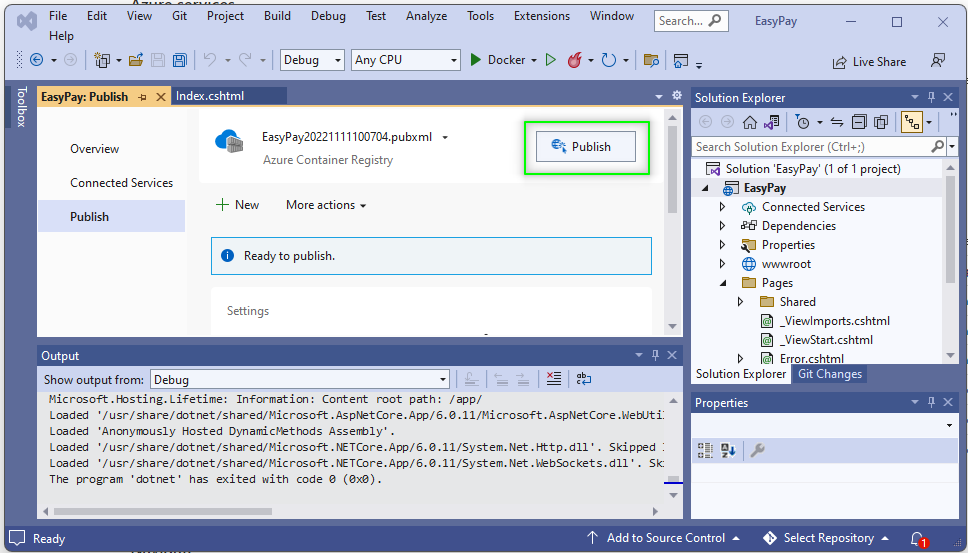
## **Create a new Azure Container Registry using same Resource Group as infra\_optimization**



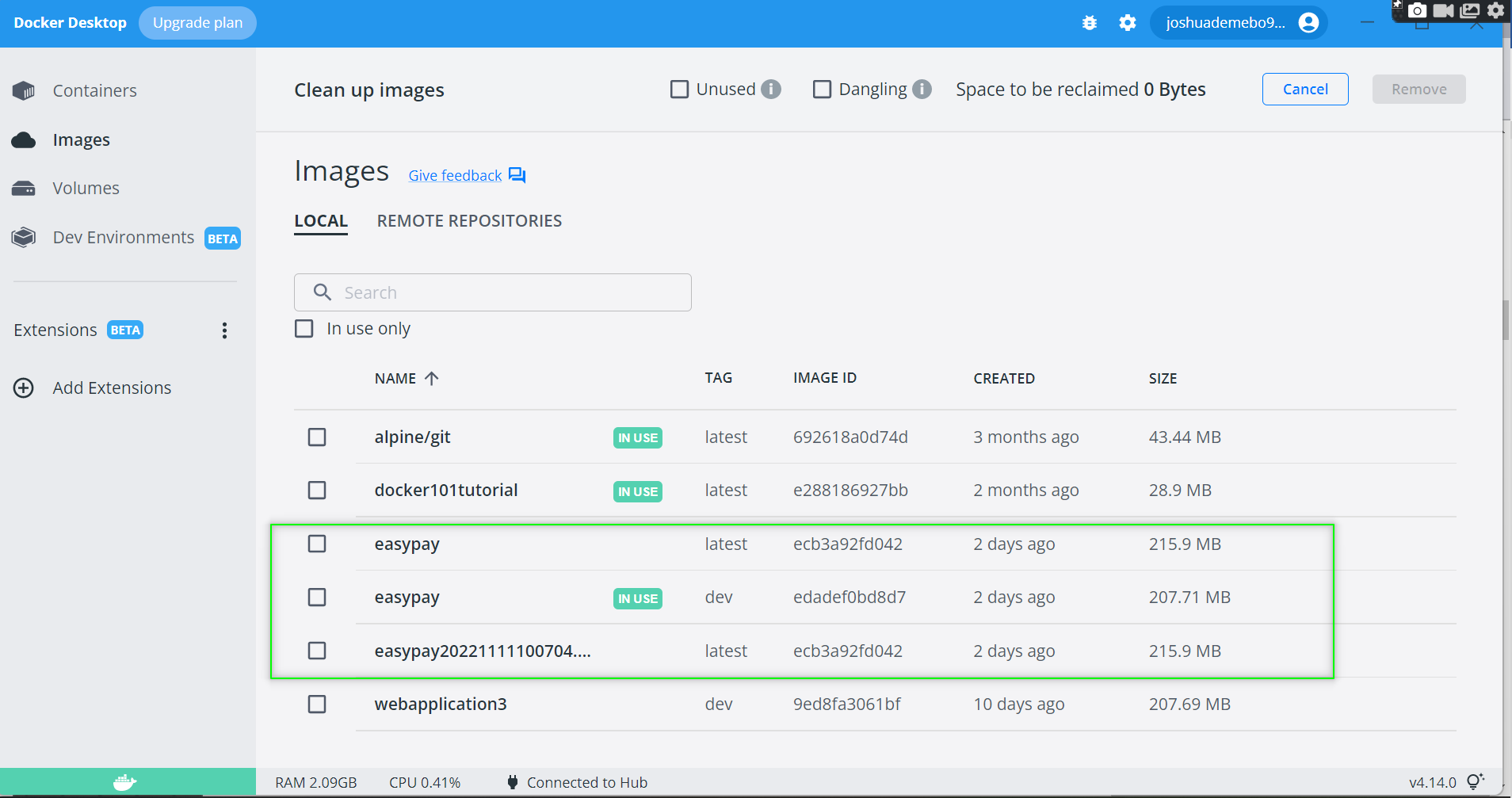


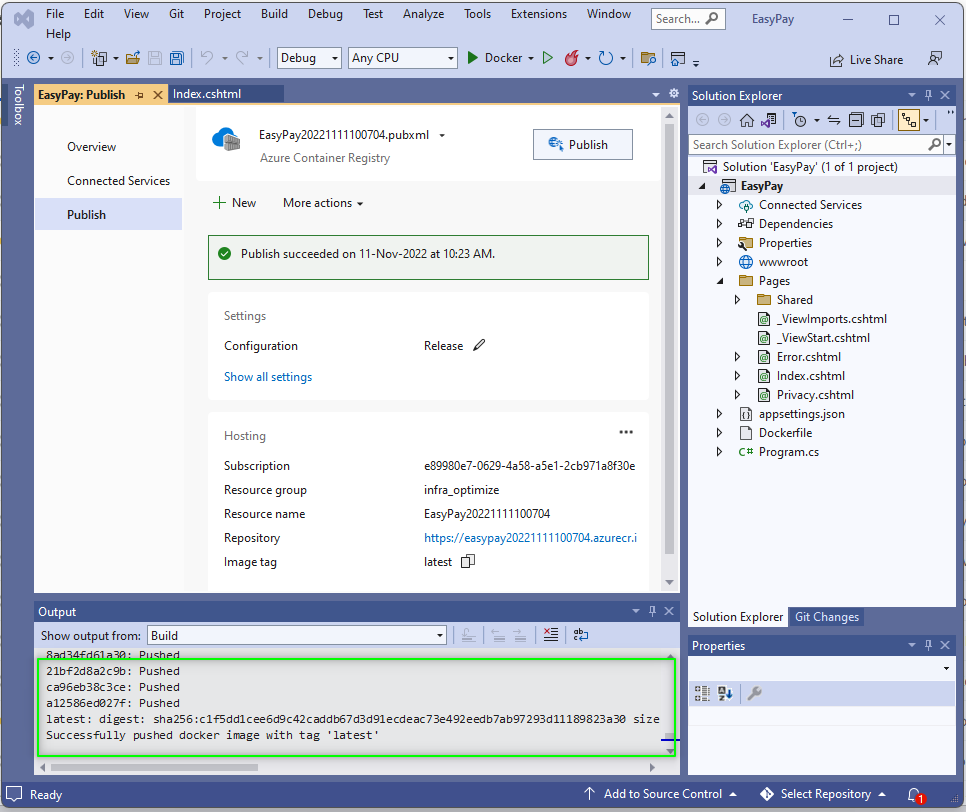


## **Click Publish. Monitor the output console for messages.**

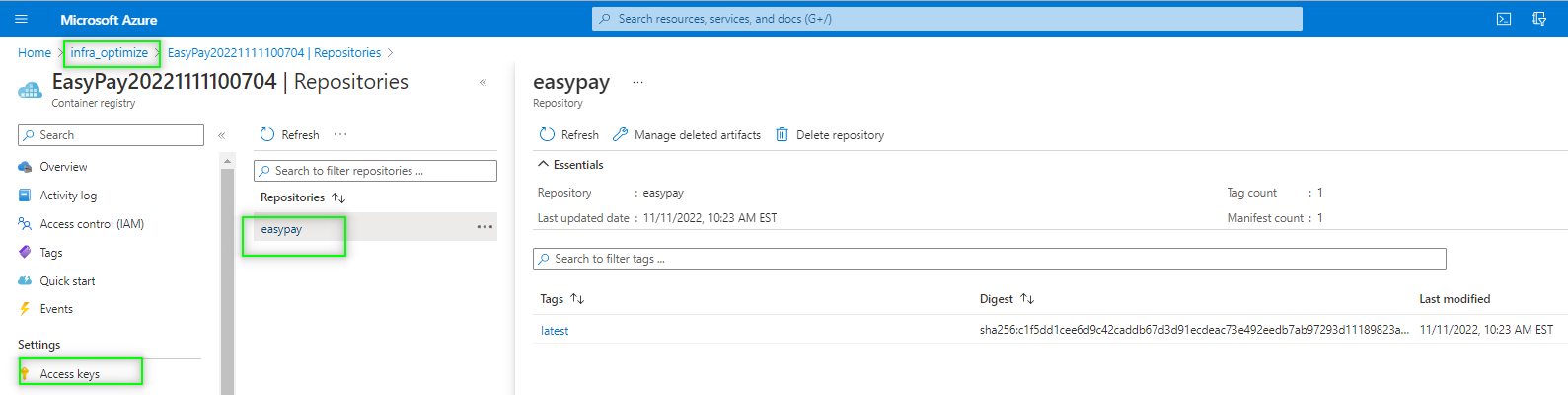


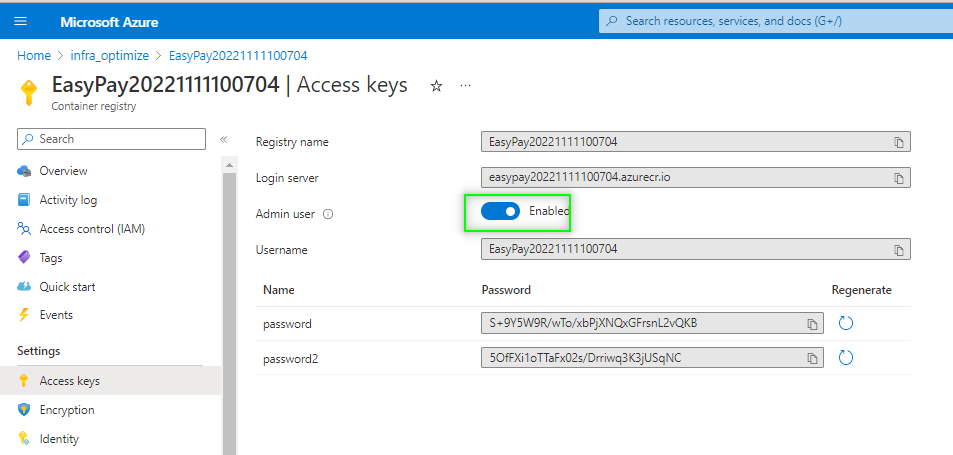
## **Successfully pushed to Docker Hub/Desktop and in Visual Studio**



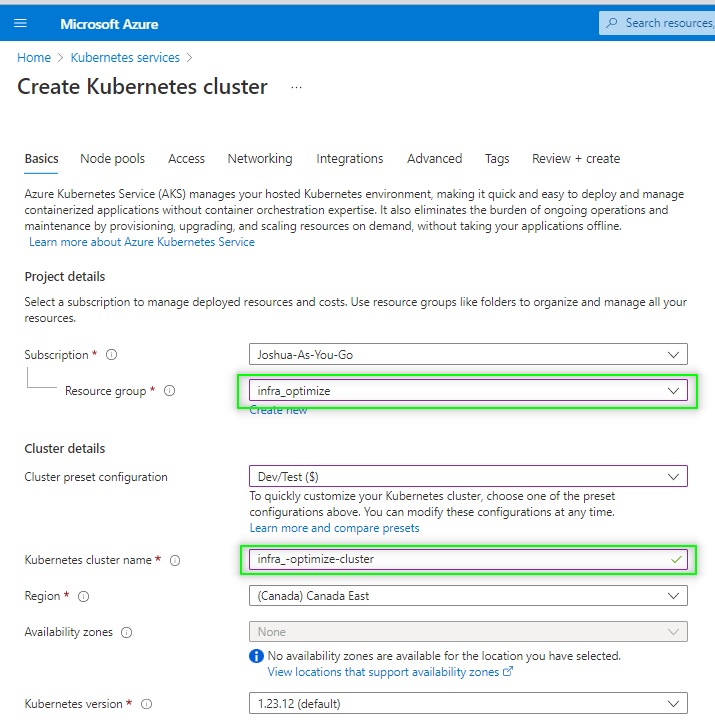


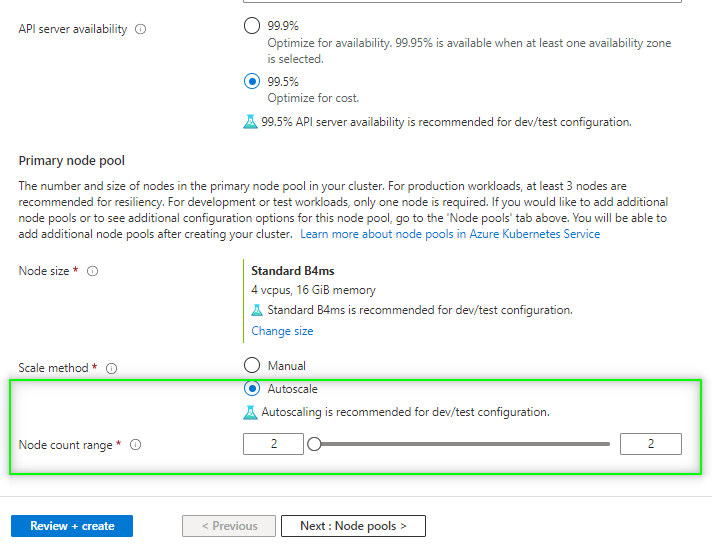
# Validate Azure Container Registry



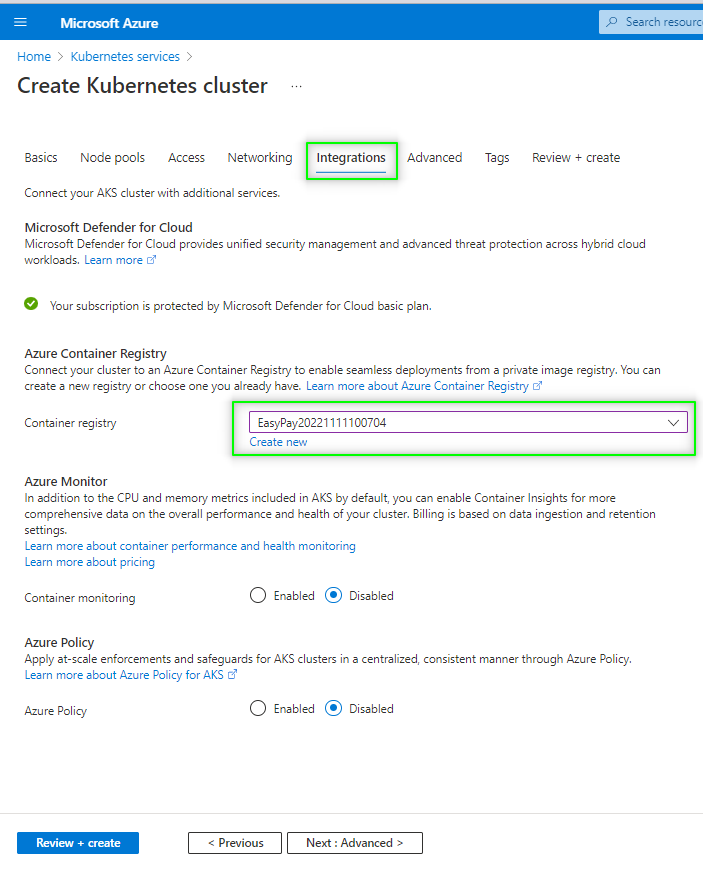


# Create a **Kubernetes** on the cluster to run on high-availability (HA) mode

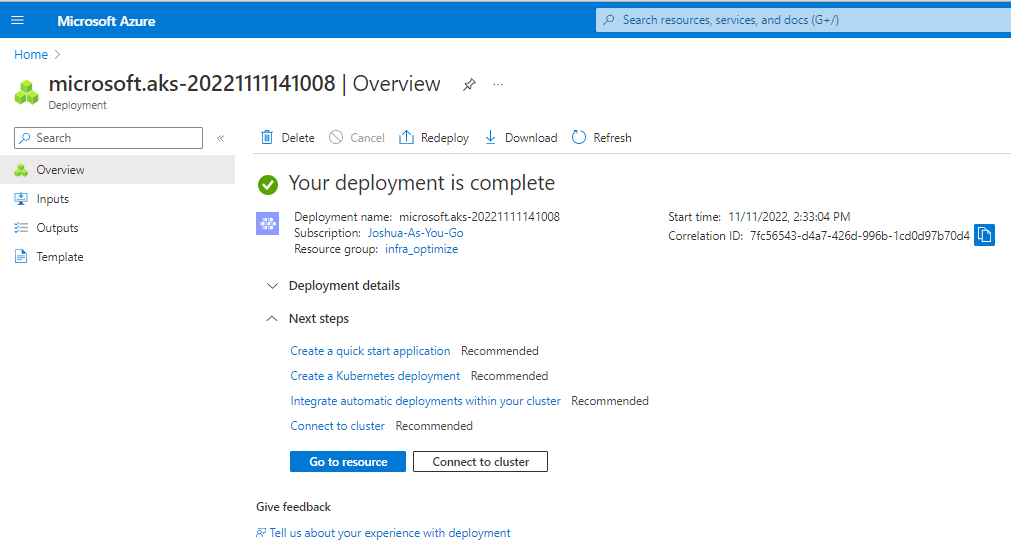


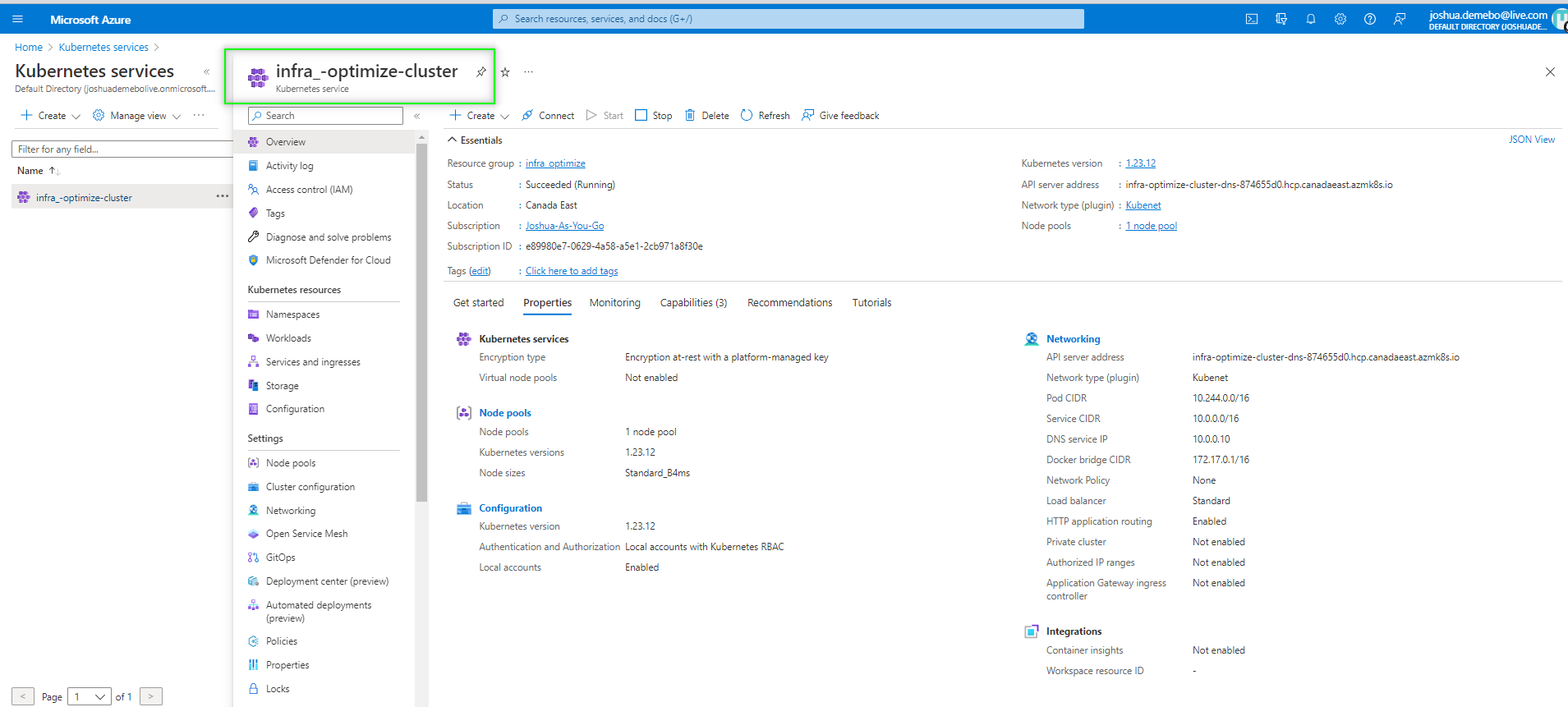


## **Integrating Container Registry**



## **Completion of Kubernetes Cluster**



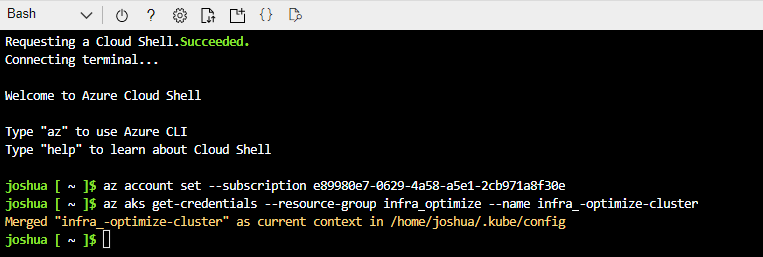


# **Deploy the Container Image to the** **AKS (Azure Kubernetes Services) Cluster**

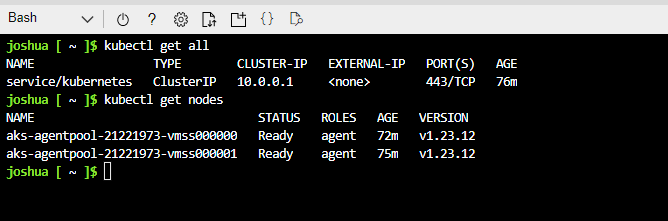
Deploy Azure Container Registry (ACR) in the AKS (Azure Kubernetes Services) that was just created. YAML files will be created to deploy the images within the cluster. The following steps will be followed:

* **Open the Azure Cloud Shell** and choose the **Bash/CLI** command prompt. **Azure Cloud Shell** is represented by the **">\_"** symbol at the top of the Azure Portal. Choose **CLI**.

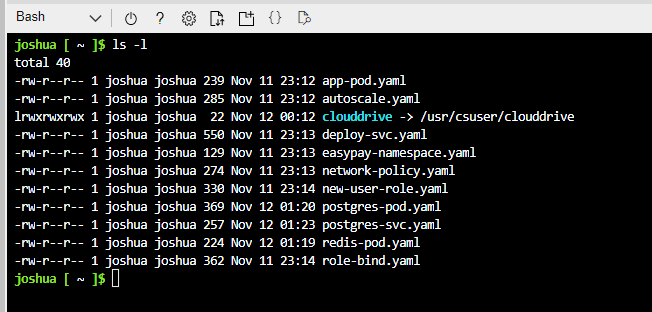
Type “**az aks get-credentials --resource-group infra\_optimize --name infra\_-optimize-cluster**”



## **Type “kubectl get nodes” to see the running nodes.**



## **Create all yaml files and upload to the /home/joshua directory**



## **Create a custom Namespace “infra-optimize-ns”**

apiVersion: v1

kind: Namespace

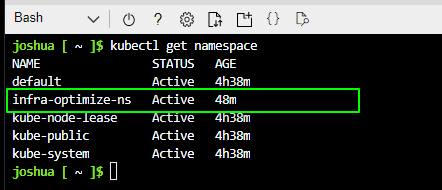
metadata:

name: infra-optimize-ns

labels:

name: easypay-namespace

app: easypay



## **Deployment of “easypay20221111100704.azurecr.io/easypay:latest” custom image and LoadBalancer service also in custom namespace “infra-optimize-ns”apiVersion: apps/v1**

kind: Deployment

metadata:

labels:

app: easypay

name: easypay

namespace: infra-optimize-ns

spec:

replicas: 1

selector:

matchLabels:

app: easypay

template:

metadata:

labels:

app: easypay

spec:

containers:

- image: easypay20221111100704.azurecr.io/easypay:latest

name: easypay-app

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: easypay

namespace: infra-optimize-ns

spec:

type: LoadBalancer

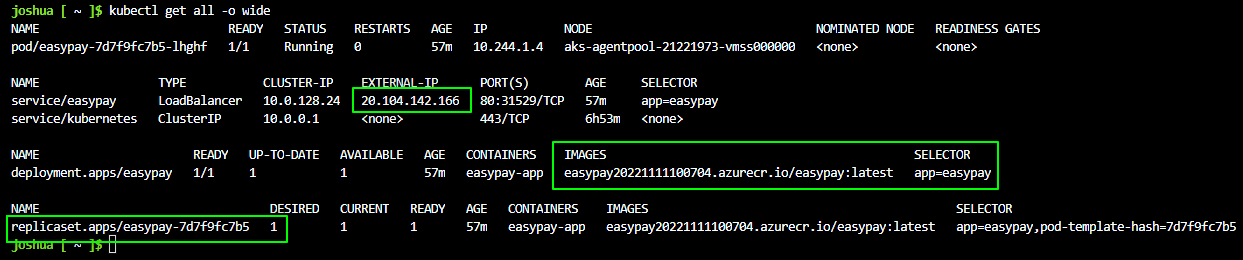
ports:

- port: 80

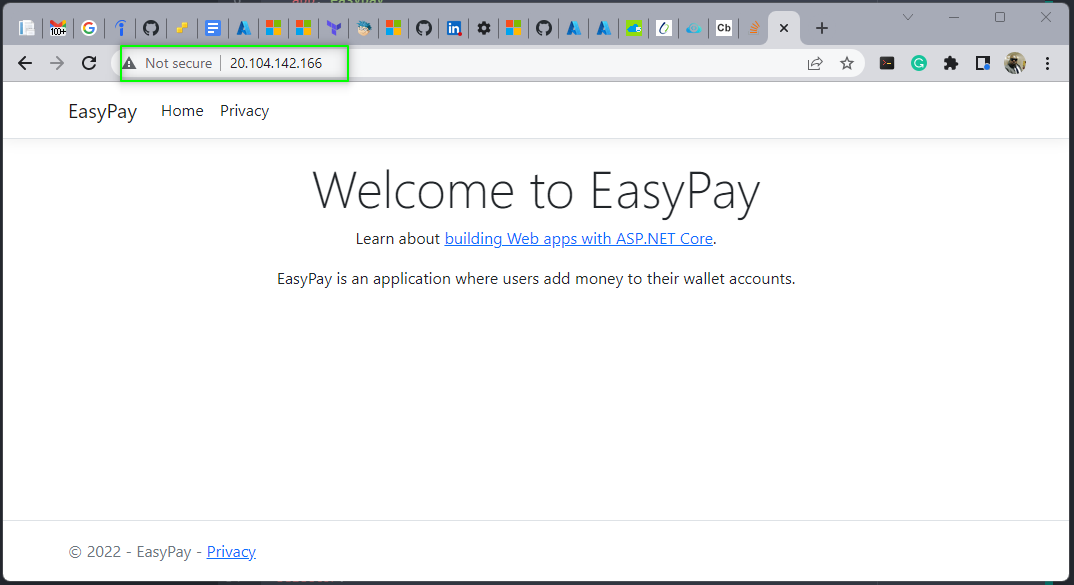
selector:

app: easypay

## **Make a note of the public IP address under “external-IP <20.104.142.166>” to be use**

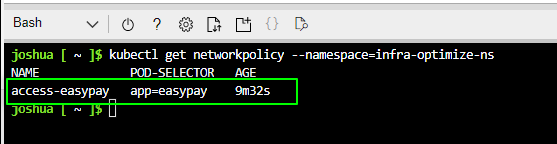


## **Copy the IP address to your clipboard. Open a browser and go to that IP address, see that your application is successfully running in Azure Kubernetes Service.**



# Implement the **network policies** at the **database** pod to allow ingress traffic from the front-end **application pod**

## **Network Policy created. “kubectl get networkpolicy --namespace=infra-optimize-ns”**



## **Created a postgresql db pod and redis caching to custom namespace=infra-optimize-ns”**

apiVersion: v1

kind: Pod

metadata:

name: postgres-pod

namespace: infra-optimize-ns

labels:

app: easypay

spec:

containers:

- name: postgres

image: postgres

ports:

- containerPort: 5432

env:

- name: POSTGRES\_USER

value: "postgres"

- name: POSTGRES\_PASSWORD

value: "postgres"

---

apiVersion: v1

kind: Service

metadata:

name: db

namespace: infra-optimize-ns

labels:

name: postgres-service

app: easypay

spec:

ports:

- port: 5432

targetPort: 5432

selector:

name: postgres-pod

app: easypay

---

apiVersion: v1

kind: Pod

metadata:

name: redis-pod

namespace: infra-optimize-ns

labels:

app: easypay

spec:

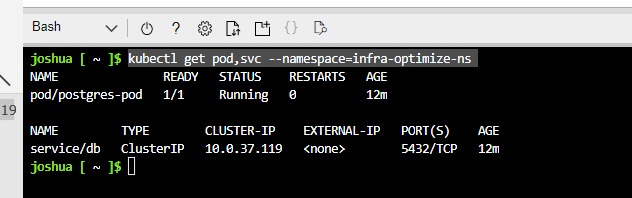
containers:

- name: redis

image: redis

ports:

- containerPort: 6379



## **Create a new user (joshua-demebo) with permissions to create, list, get, update, and delete pods**

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

namespace: infra-optimize-ns

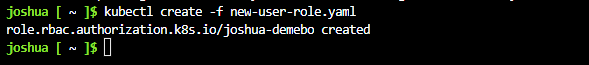
name: joshua-demebo

rules:

- apiGroups: ["", "extensions", "apps"]

resources: ["deployments", "pods", "services","namespaces"]

verbs: ["create", "list", "get", "update", "Pod", "delete"]



## **Create a Role Binding**

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: role-bind

namespace: infra-optimize-ns

subjects:

- kind: ServiceAccount

name: joshua-demebo

namespace: infra-optimize-ns

roleRef:

kind: Role

name: joshua-demebo

apiGroup: ""

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: role-bind

namespace: infra-optimize-ns

subjects:

- kind: ServiceAccount

name: joshua-demebo

namespace: infra-optimize-ns

roleRef:

kind: Role

name: joshua-demebo

apiGroup: ""



## **Created a Redis for caching**

apiVersion: v1

kind: Pod

metadata:

name: redis-pod

namespace: infra-optimize-ns

labels:

app: easypay

spec:

containers:

- name: redis

image: redis

ports:

- containerPort: 6379

apiVersion: v1

kind: Pod

metadata:

name: redis-pod

namespace: infra-optimize-ns

labels:

app: easypay

spec:

containers:

- name: redis

image: redis

ports:

- containerPort: 6379



## **Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured**

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: auto-pod

namespace: infra-optimize-ns

spec:

scaleTargetRef:

apiVersion: apps/v1

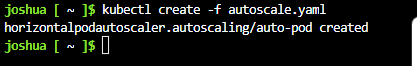
kind: Deployment

name: easypay

minReplicas: 1

maxReplicas: 5

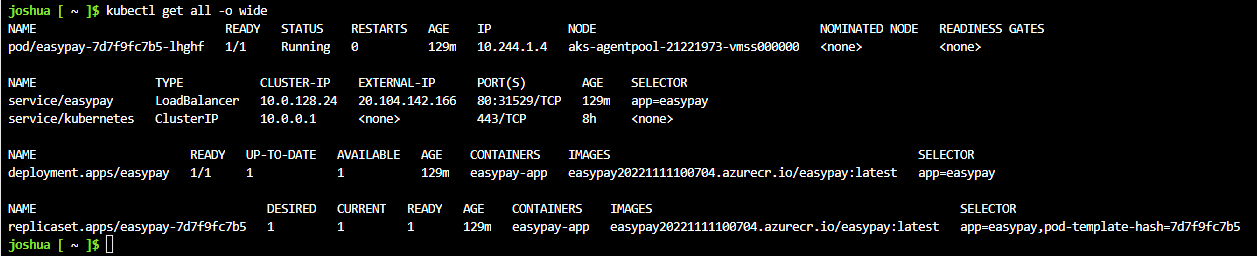
targetCPUUtilizationPercentage: 50



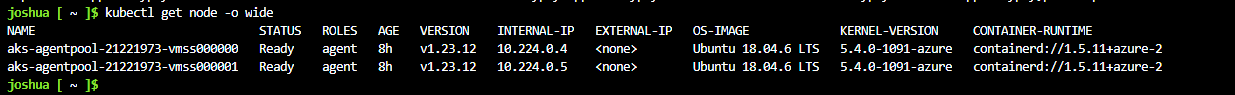
# Take snapshot of ETCD database

**sudo ETCDCTL\_API=3 etcdctl snapshot save snapshot.db --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key**

## **kubectl get all -o wide**



## **kubectl get all -o wide**



# **Conclusion**

To improve the DevOps infrastructure for an eCommerce application to run on high-availability state, which EasyPay app was used in this case, the above configuration was tested and deployed using various tools to accomplish the result. All source code was uploaded to GitHub for continues improvement either vertical or horizontal autoscaling.

I strongly believe all being done and with the documentation, it will improve the quality and performance of the eCommerce and prevent connectivity downtime.

This infra optimization will improve the app and give an online shoppers more reliable, fast, and secure improved performance of the EasyPay app.