



[Infra_Optimization]

[Project 3]

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

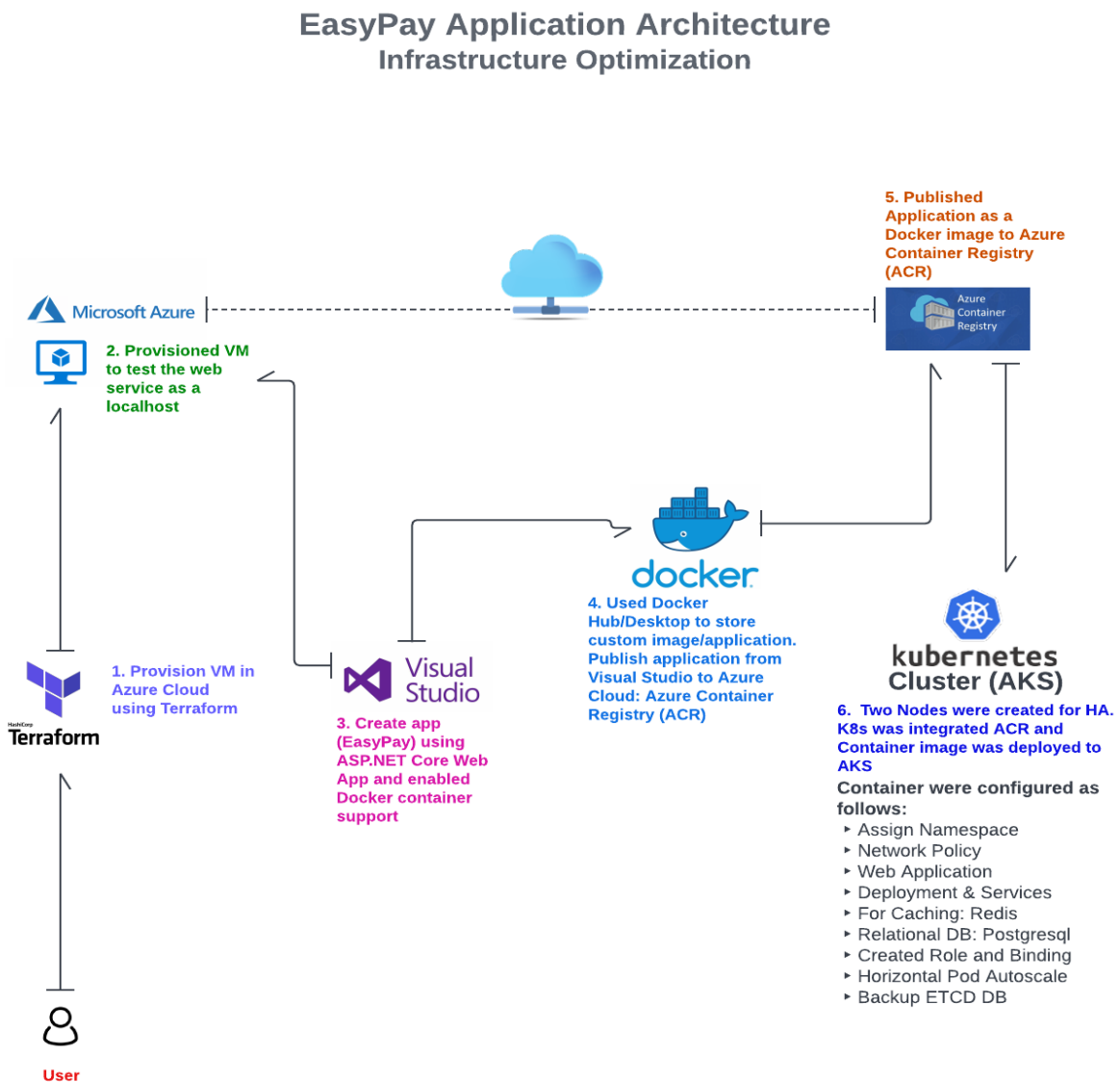
The following things to be kept in check:

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

DESCRIPTION.....	0
EasyPay App Architecture Diagram.....	1
Introduction.....	3
The following specific tools were used.....	4
1. Automating the provisioning of an Azure VM using Terraform. Created using local git and using VS Code to edit the script.....	4
▪ Provider.tf for provisioning a VM in Azure.....	5
▪ Main.tf for provisioning a VM in Azure.....	6
▪ A provisioned VM in Azure (hostname easypay-app01).....	8
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
▪ Naming the project as EasyPay.....	9
▪ Selecting ".NET 6.0 (Long-term support)" as the language, "Enable Docker" to enable container support, and "Linux" as the container language.....	10
▪ EasyPay application created.....	10
▪ Modify this page by changing the text on the screen to have a visual look of the app.	11
App was Built successfully.	11
▪ 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
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
▪ Select Azure Container Registry. Click Next.....	12
▪ Signing in with my credential.....	13
▪ Create a new Azure Container Registry using same Resource Group as infra_optimization.....	13
▪ Click Publish. Monitor the output console for messages.	14
▪ Successfully pushed to Docker Hub/Desktop and in Visual Studio.....	15
4. Validate Azure Container Registry.....	16
5. Create a Kubernetes on the cluster to run on high-availability (HA) mode.....	17
▪ Integrating Container Registry.....	18
▪ Completion of Kubernetes Cluster.....	18
6. Deploy the Container Image to the AKS (Azure Kubernetes Services) Cluster.....	19
▪ Open the Azure Cloud Shell.....	19
▪ Type "kubectl get nodes" to see the running nodes.....	19
▪ Create all yaml files and upload to the /home/joshua directory.....	20
▪ Create a custom Namespace "infra-optimize-ns".....	20
▪ Deployment of "easypay20221111100704.azurecr.io/easypay:latest" custom image and LoadBalancer service also in custom namespace "infra-optimize-ns"apiVersion: apps/v1.....	20
▪ Make a note of the public IP address under "external-IP <20.104.142.166>" to be use.....	21
▪ 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
7. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod.....	22
▪ Network Policy created. "kubectl get networkpolicy --namespace=infra-optimize-ns".....	22
▪ Created a postgresql db pod and redis caching to custom namespace=infra-optimize-ns".....	22
▪ Create a new user (joshua-demebo) with permissions to create, list, get, update, and delete pods.....	23
▪ Create a Role Binding.....	24
▪ Created a Redis for caching.....	25
▪ Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured.....	25
8. Take snapshot of ETCD database.....	26
▪ kubectl get all -o wide.....	26
▪ kubectl get all -o wide.....	26
Conclusion.....	26

EasyPay App Architecture Diagram



Introduction

This documentation contains all steps and procedures in successfully managing a DevOps infrastructure for an eCommerce 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.
4. **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.
5. **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.

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

```
jdemebo@ubuntumachine:~/capstone_project/infra_optimization/terraform_az_vm$ ls -l
total 24
-rw-rw-r-- 1 jdemebo jdemebo 1666 Nov  8 06:15 main.tf
-rw-rw-r-- 1 jdemebo jdemebo 161 Nov  8 06:17 provider.tf
-rw-rw-r-- 1 jdemebo jdemebo 9766 Nov  8 06:21 terraform.tfstate
-rw-rw-r-- 1 jdemebo jdemebo 178 Nov  8 06:19 terraform.tfstate.backup
jdemebo@ubuntumachine:~/capstone_project/infra_optimization/terraform_az_vm$
```

- Provider.tf for provisioning a VM in Azure

```

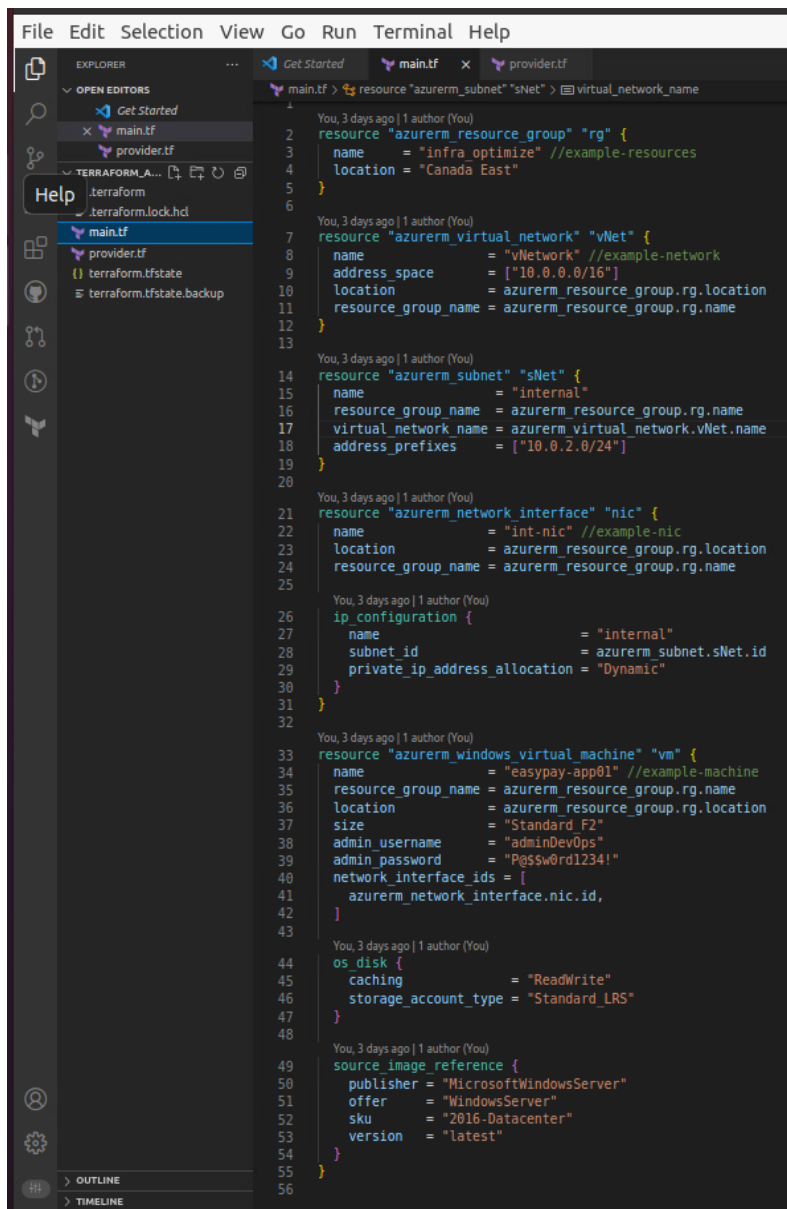
1 terraform {
2   required_providers {
3     azure = {
4       source = "hashicorp/azurerm"
5       version = "3.30.0"
6     }
7   }
8 }
9
10 provider "azurerm" {
11   features {}
12 }
13
```

```

terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "3.30.0"
    }
  }
}

provider "azurerm" {
  features {}
}
```

- Main.tf for provisioning a VM in Azure



```

1  resource "azurerm_resource_group" "rg" {
2    name     = "infra_optimize" //example-resources
3    location = "Canada East"
4  }
5
6  resource "azurerm_virtual_network" "vNet" {
7    name                = "vNetwork" //example-network
8    address_space       = ["10.0.0.0/16"]
9    location             = azurerm_resource_group.rg.location
10   resource_group_name = azurerm_resource_group.rg.name
11 }
12
13 resource "azurerm_subnet" "sNet" {
14   name                = "internal"
15   resource_group_name = azurerm_resource_group.rg.name
16   virtual_network_name = azurerm_virtual_network.vNet.name
17   address_prefixes    = ["10.0.2.0/24"]
18 }
19
20 resource "azurerm_network_interface" "nic" {
21   name                = "int-nic" //example-nic
22   location             = azurerm_resource_group.rg.location
23   resource_group_name = azurerm_resource_group.rg.name
24 }
25
26 ip_configuration {
27   name                = "internal"
28   subnet_id           = azurerm_subnet.sNet.id
29   private_ip_address_allocation = "Dynamic"
30 }
31
32 resource "azurerm_windows_virtual_machine" "vm" {
33   name                = "easypay-app01" //example-machine
34   resource_group_name = azurerm_resource_group.rg.name
35   location             = azurerm_resource_group.rg.location
36   size                = "Standard F2"
37   admin_username       = "adminDevOps"
38   admin_password       = "Pq$Sw0rd1234!"
39   network_interface_ids = [
40     azurerm_network_interface.nic.id,
41   ]
42 }
43
44 os_disk {
45   caching              = "ReadWrite"
46   storage_account_type = "Standard_LRS"
47 }
48
49 source_image_reference {
50   publisher = "MicrosoftWindowsServer"
51   offer     = "WindowsServer"
52   sku       = "2016-Datacenter"
53   version   = "latest"
54 }
55 }
56

```

```

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"
  }
}

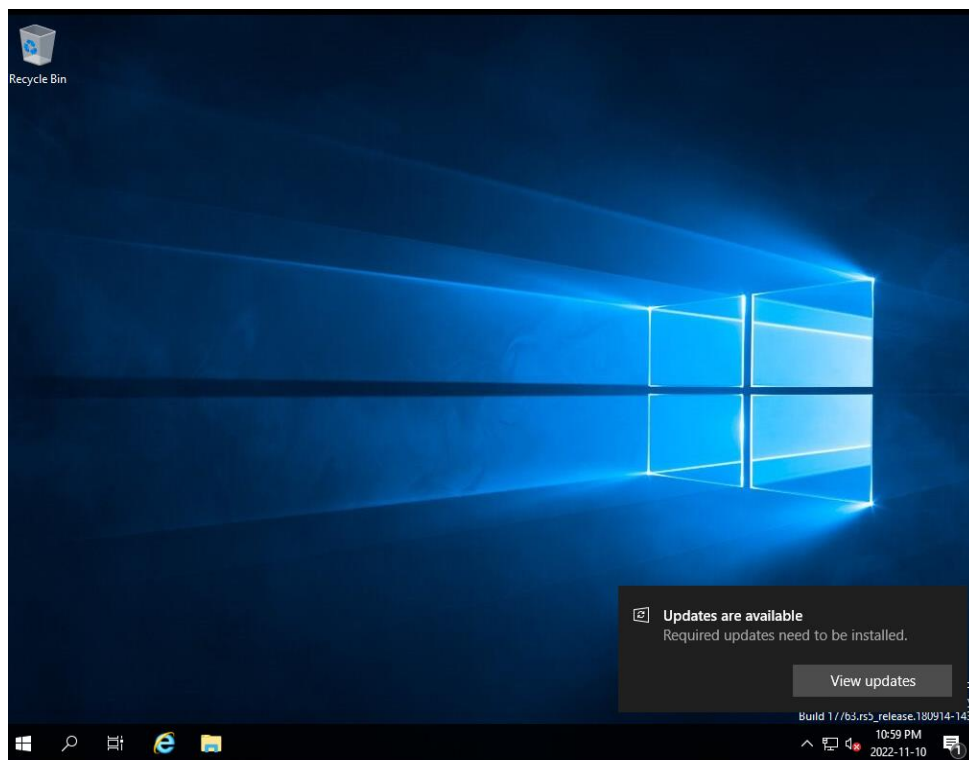
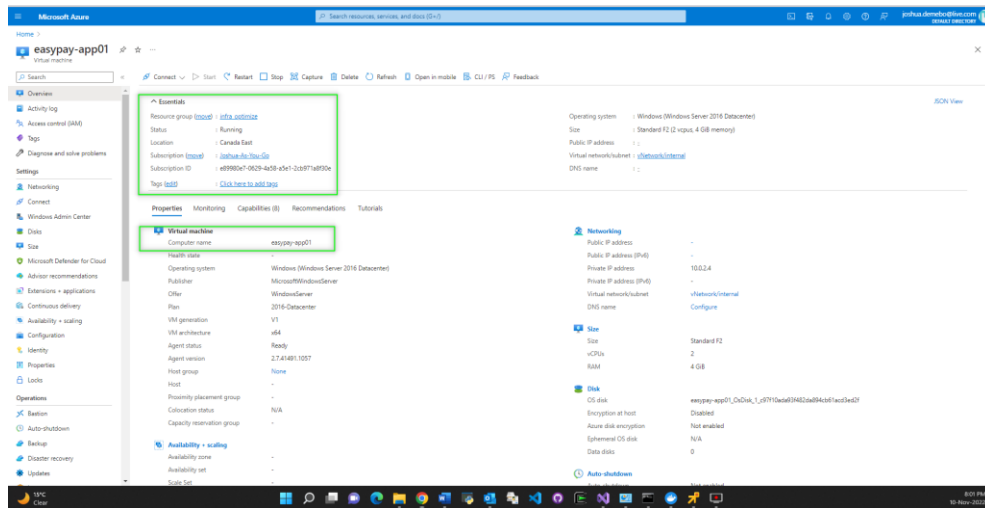
resource "azurerm_windows_virtual_machine" "vm" {
  name            = "easypay-app01" //example-machine
  resource_group_name = azurerm_resource_group.rg.name
  location        = azurerm_resource_group.rg.location
  size            = "Standard_F2"
  admin_username  = "adminDevOps"
  admin_password  = "P@$w0rd1234!"
  network_interface_ids = [
    azurerm_network_interface.nic.id,
  ]

  os_disk {
    caching          = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

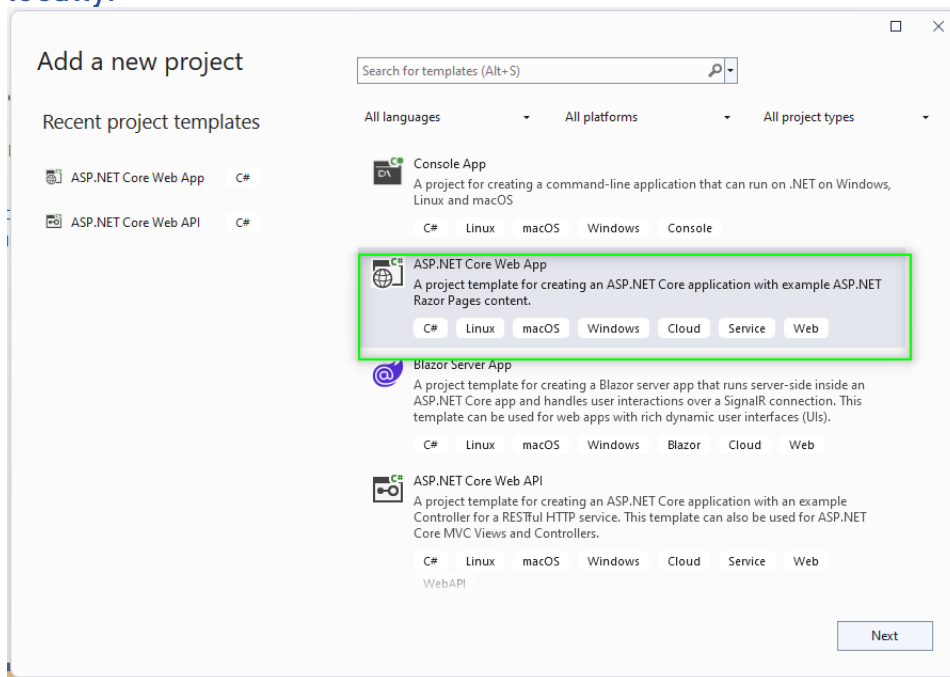
  source_image_reference {
    publisher = "MicrosoftWindowsServer"
    offer     = "WindowsServer"
    sku       = "2016-Datacenter"
    version   = "latest"
  }
}

```

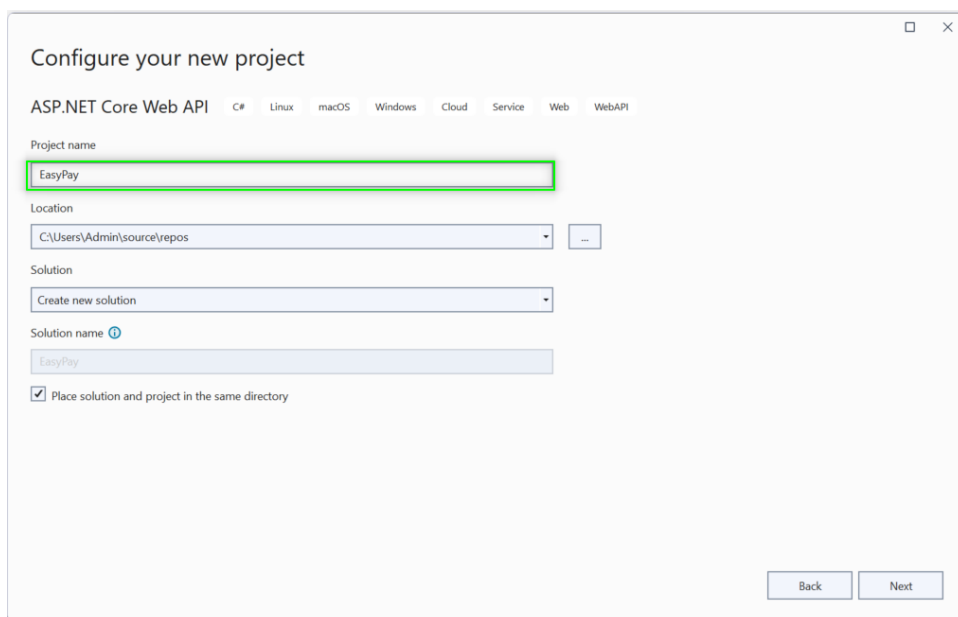

- A provisioned VM in Azure (hostname easypay-app01)



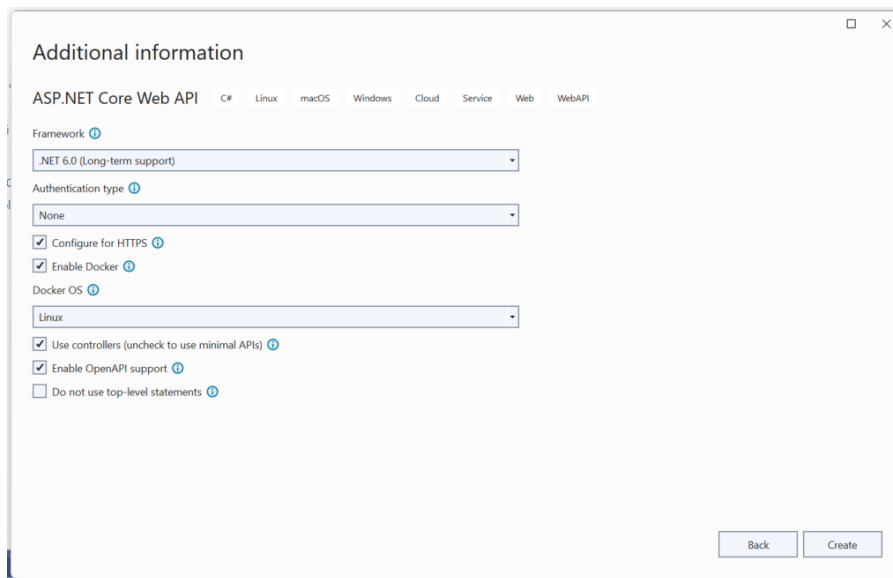
2. Creating a new app in Visual Studio as an ASP.NET Core Web App, enable Docker support, and make a small change to it. Ensure it builds and runs locally.



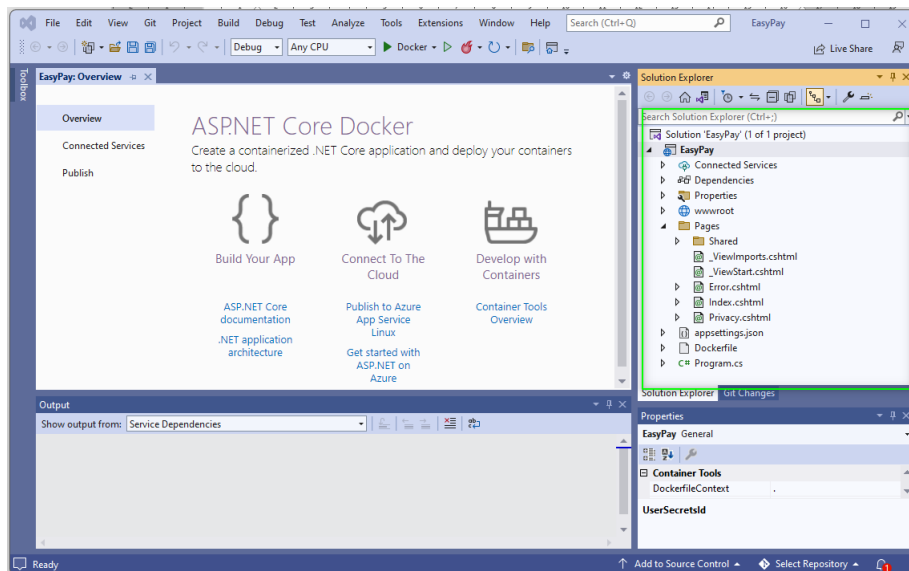
- Naming the project as EasyPay



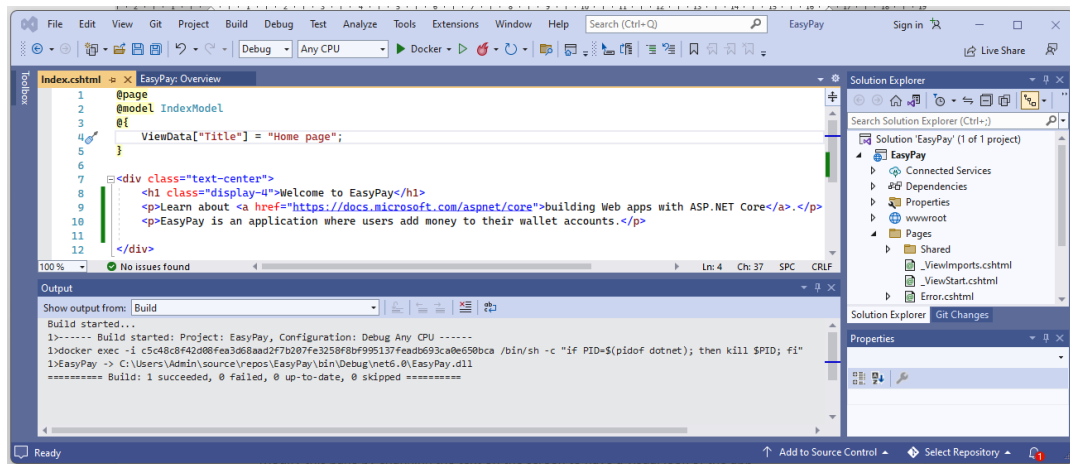
- Selecting ".NET 6.0 (Long-term support)" as the language, "Enable Docker" to enable container support, and "Linux" as the container language.



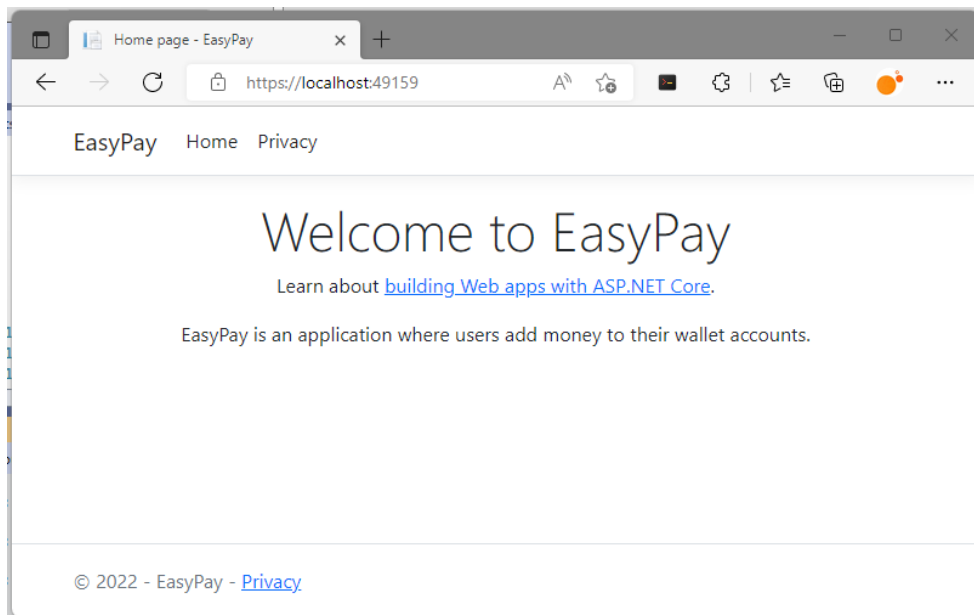
- EasyPay application created



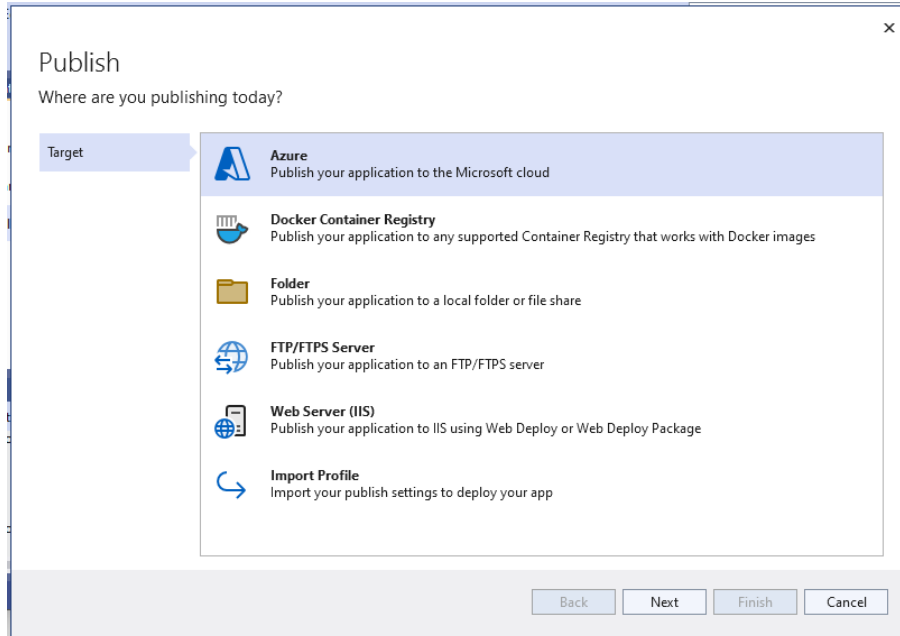
- Modify this page by changing the text on the screen to have a visual look of the app. App was Built successfully.



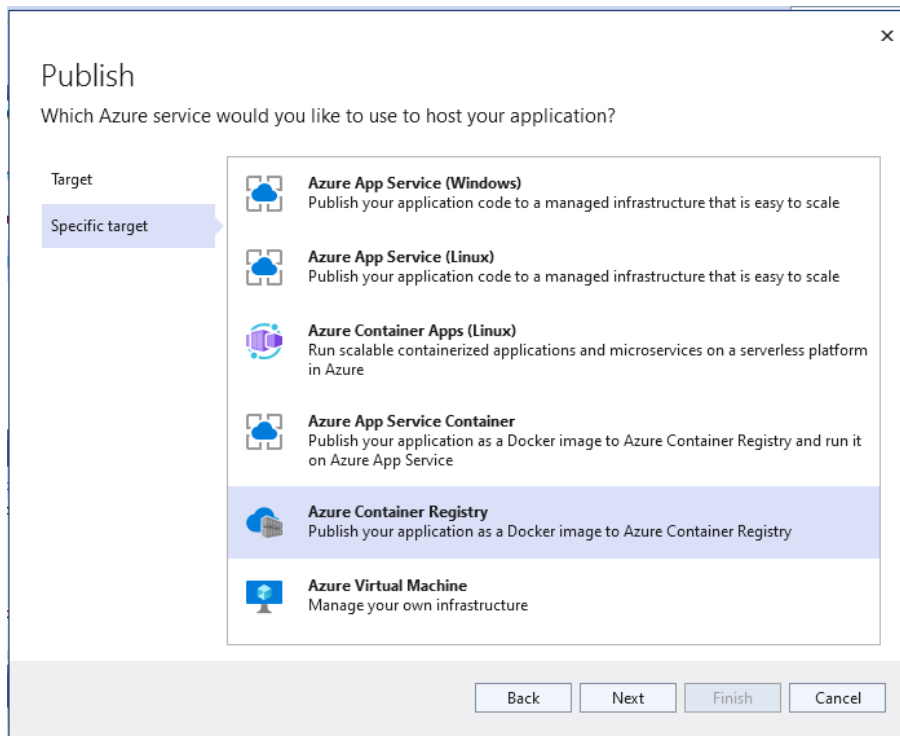
- Test application on Azure VM. Before deploying this solution to Azure, it will be nice to test the application on the tenant VM as a web app.



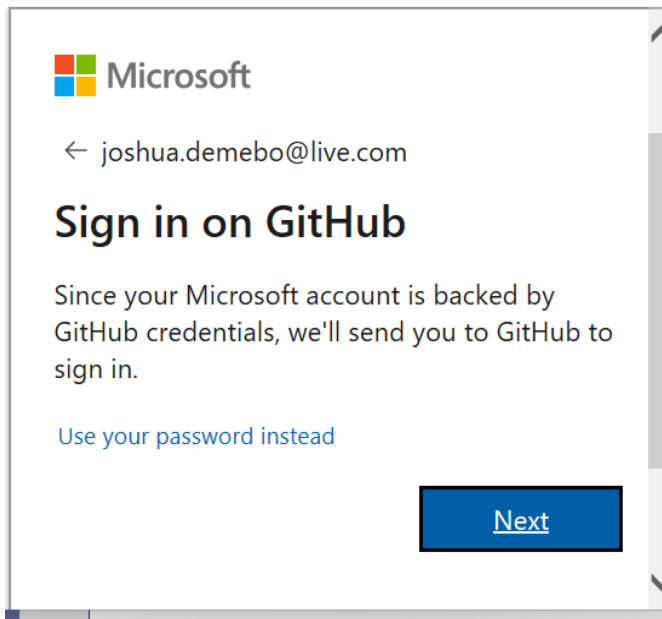
- The application will be published from Visual Studio to a new Azure Container Registry. A new Azure Container Registry will be created using Visual Studio. Under Build menu, choose Publish. And Choose Azure as the Target. Click Next.



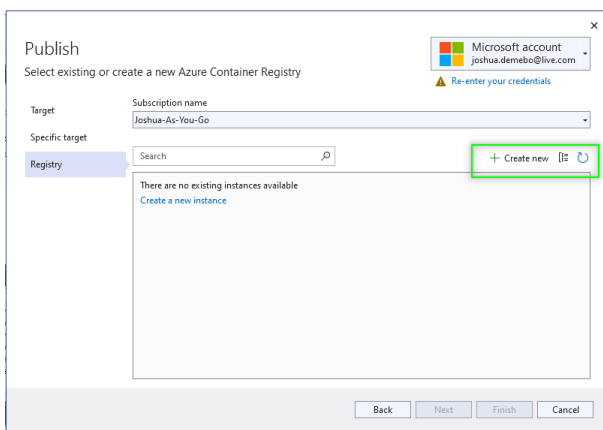
- Select Azure Container Registry. Click Next.

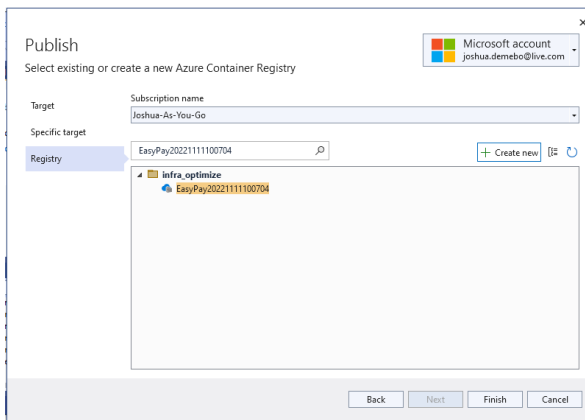
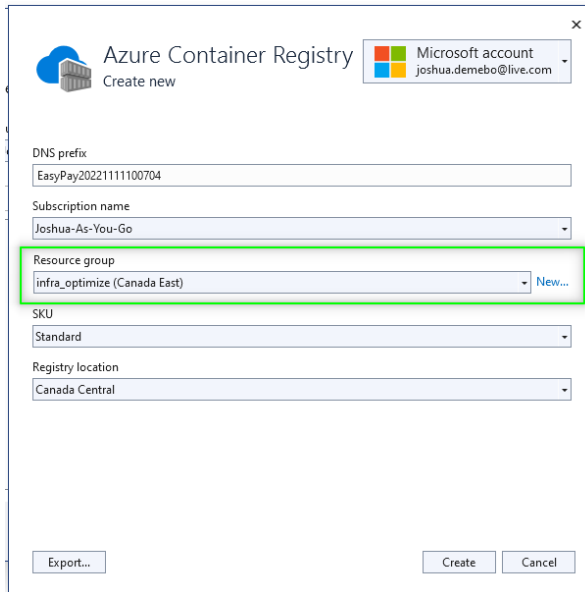


- Signing in with my credential

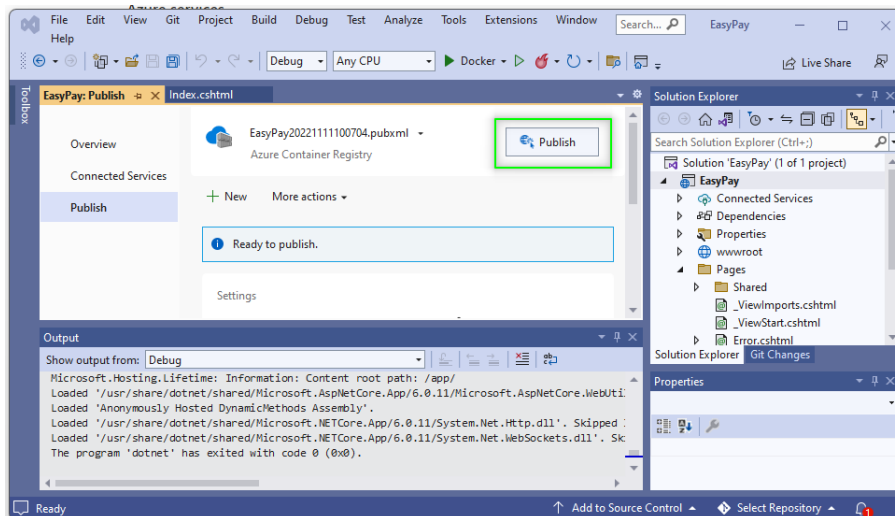


- Create a new Azure Container Registry using same Resource Group as infra_optimization

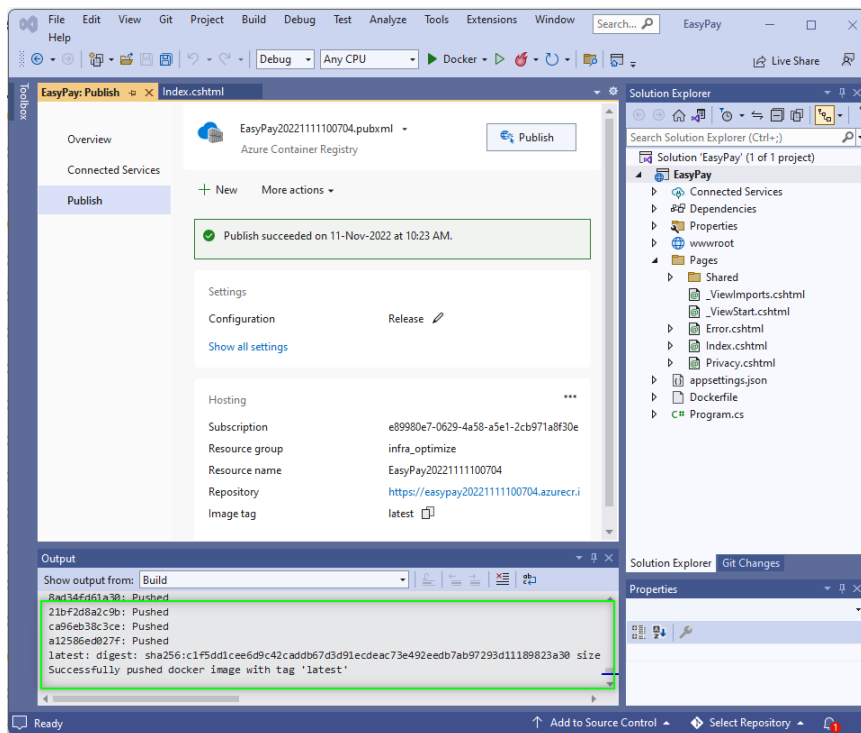
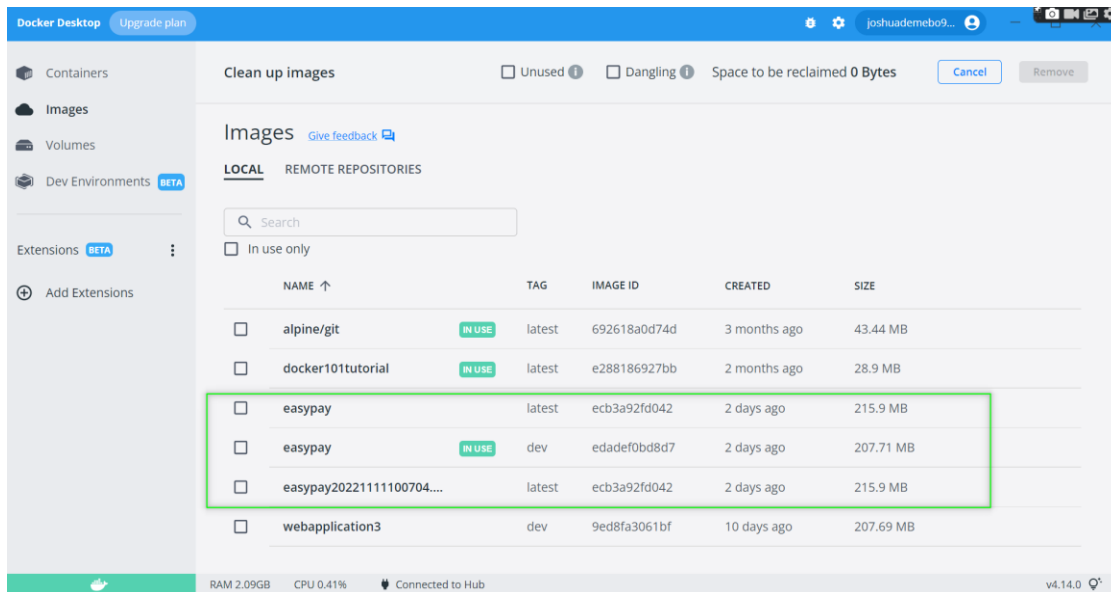




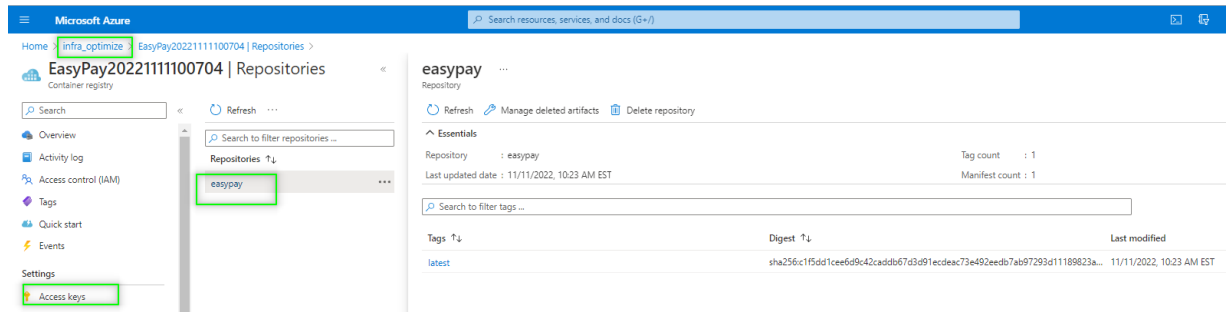
- Click Publish. Monitor the output console for messages.



- Successfully pushed to Docker Hub/Desktop and in Visual Studio



4. Validate Azure Container Registry



Microsoft Azure

Home > infra_optimize > EasyPay20221111100704 | Repositories >

EasyPay20221111100704 | Repositories

Search

Refresh

Search to filter repositories ...

Repositories 1

easypay

Repository

Refresh Manage deleted artifacts Delete repository

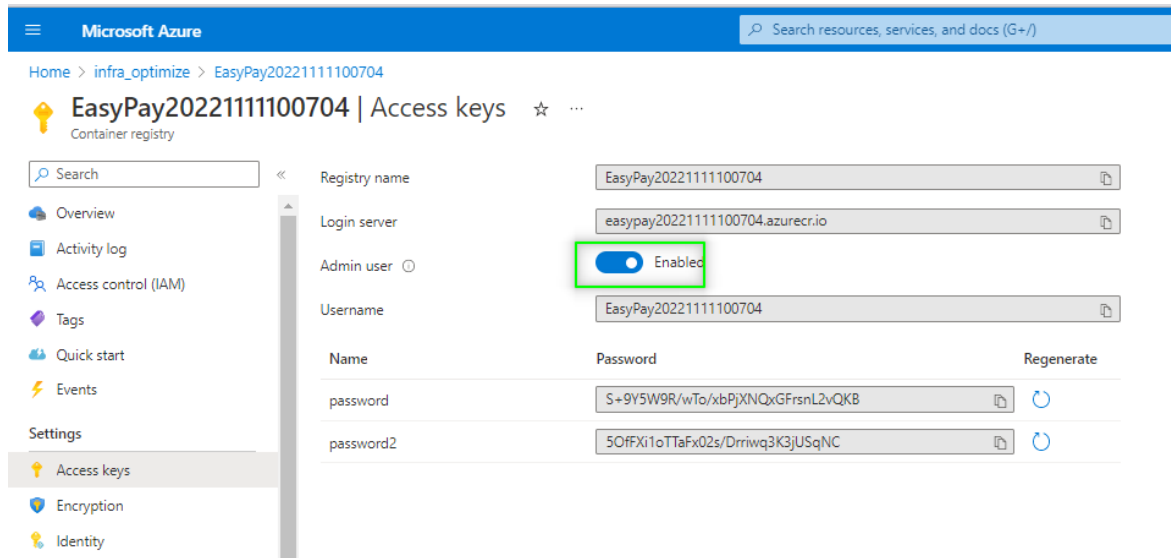
Essentials

Repository : easypay Tag count : 1

Last updated date : 11/11/2022, 10:23 AM EST Manifest count : 1

Search to filter tags ...

Tags	Digest	Last modified
latest	sha256:c1f5dd1cee6b9c42caddb67d3d91ecdeac73e492eedb7ab97293d11189823a...	11/11/2022, 10:23 AM EST



Microsoft Azure

Home > infra_optimize > EasyPay20221111100704

EasyPay20221111100704 | Access keys

Container registry

Search

Registry name EasyPay20221111100704

Login server easypay20221111100704.azurecr.io

Admin user

Enabled

Username EasyPay20221111100704

Name	Password	Regenerate
password	S+9Y5W9R/wTo/xbPJXNQxGFrSnL2vQKB	
password2	5OfFXi1oTTaFx02s/Drriwq3K3jUSqNC	

5. Create a Kubernetes on the cluster to run on high-availability (HA) mode

Microsoft Azure

Search resources

[Home](#) > [Kubernetes services](#) >

Create Kubernetes cluster

Basics

Node pools

Access

Networking

Integrations

Advanced

Tags

Review + create

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline.

[Learn more about Azure Kubernetes Service](#)

Project details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Joshua-As-You-Go

Resource group *

infra_optimize

Create new

Cluster details

Cluster preset configuration

Dev/Test (\$)

To quickly customize your Kubernetes cluster, choose one of the preset configurations above. You can modify these configurations at any time.

[Learn more and compare presets](#)

Kubernetes cluster name *

infra-optimize-cluster

Region *

(Canada) Canada East

Availability zones

None

No availability zones are available for the location you have selected.

[View locations that support availability zones](#)

Kubernetes version *

1.23.12 (default)

API server availability

99.9%

Optimize for availability. 99.95% is available when at least one availability zone is selected.

99.5%

Optimize for cost.

99.5% API server availability is recommended for dev/test configuration.

Primary node pool

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. If you would like to add additional node pools or to see additional configuration options for this node pool, go to the 'Node pools' tab above. You will be able to add additional node pools after creating your cluster. [Learn more about node pools in Azure Kubernetes Service](#)

Node size *

Standard B4ms

4 vcpus, 16 GiB memory

Standard B4ms is recommended for dev/test configuration.

[Change size](#)

Scale method *

Manual

Autoscale

Autoscaling is recommended for dev/test configuration.

Node count range *

2

2

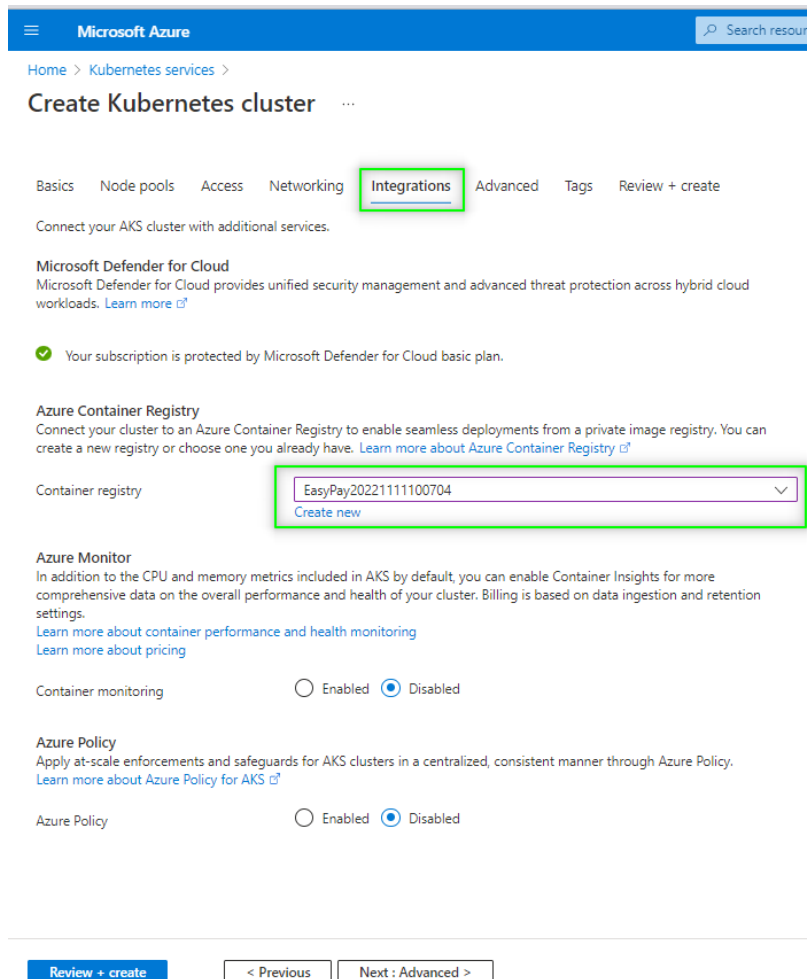
Review + create

< Previous

Next : Node pools >

17 | Page

■ Integrating Container Registry



Microsoft Azure

Home > Kubernetes services >

Create Kubernetes cluster

Basics Node pools Access Networking **Integrations** Advanced Tags Review + create

Connect your AKS cluster with additional services.

Microsoft Defender for Cloud
Microsoft Defender for Cloud provides unified security management and advanced threat protection across hybrid cloud workloads. [Learn more](#)

✓ Your subscription is protected by Microsoft Defender for Cloud basic plan.

Azure Container Registry
Connect your cluster to an Azure Container Registry to enable seamless deployments from a private image registry. You can create a new registry or choose one you already have. [Learn more about Azure Container Registry](#)

Container registry: EasyPay20221111100704 [Create new](#)

Azure Monitor
In addition to the CPU and memory metrics included in AKS by default, you can enable Container Insights for more comprehensive data on the overall performance and health of your cluster. Billing is based on data ingestion and retention settings. [Learn more about container performance and health monitoring](#)
[Learn more about pricing](#)

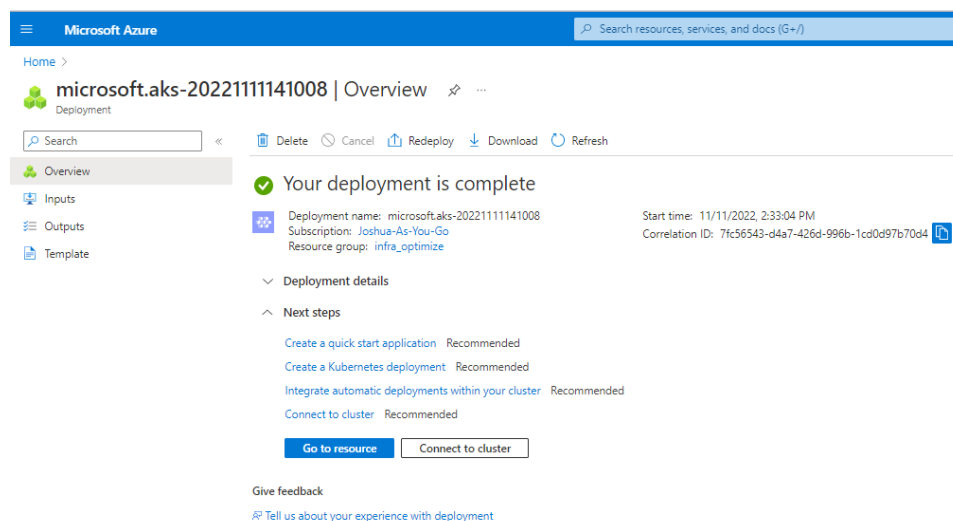
Container monitoring: ☐ Enabled ☒ Disabled

Azure Policy
Apply at-scale enforcements and safeguards for AKS clusters in a centralized, consistent manner through Azure Policy. [Learn more about Azure Policy for AKS](#)

Azure Policy: ☐ Enabled ☒ Disabled

[Review + create](#) [< Previous](#) [Next : Advanced >](#)

■ Completion of Kubernetes Cluster



Microsoft Azure

Home >

microsoft.aks-20221111141008 | Overview

Deployment

Search

Delete Cancel Redeploy Download Refresh

Overview

Inputs

Outputs

Template

✓ Your deployment is complete

Deployment name: microsoft.aks-20221111141008 Start time: 11/11/2022, 2:33:04 PM
Subscription: Joshua-As-You-Go
Resource group: infra_optimize Correlation ID: 7fc56543-d4a7-426d-996b-1cd0d97b70d4

Deployment details

Next steps

Create a quick start application Recommended

Create a Kubernetes deployment Recommended

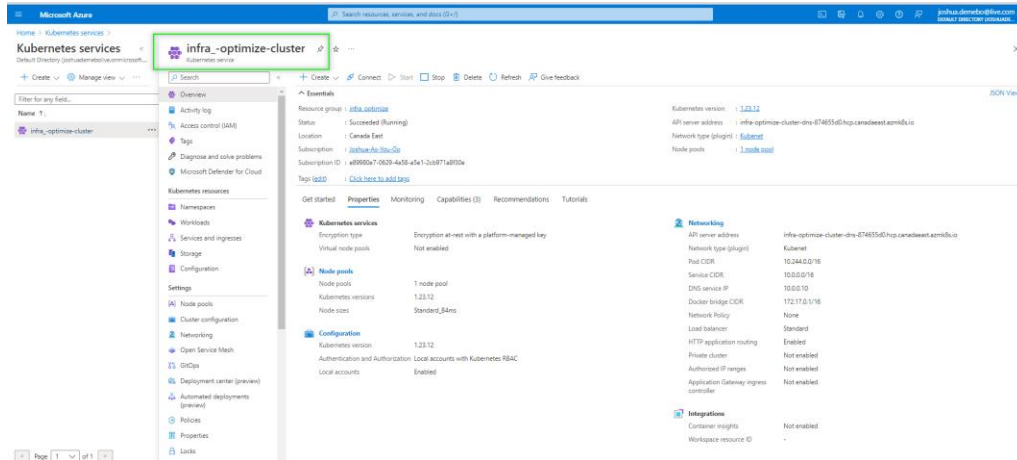
Integrate automatic deployments within your cluster Recommended

Connect to cluster Recommended

[Go to resource](#) [Connect to cluster](#)

Give feedback

Tell us about your experience with deployment



6. 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"

```
Bash
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

joshua [ ~ ]$ az account set --subscription e89980e7-0629-4a58-a5e1-2cb971a8f30e
joshua [ ~ ]$ az aks get-credentials --resource-group infra_optimize --name infra_-optimize-cluster
Merged "infra_-optimize-cluster" as current context in /home/joshua/.kube/config
joshua [ ~ ]$
```

- Type "kubectl get nodes" to see the running nodes.

```
Bash
joshua [ ~ ]$ kubectl get all
NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
service/kubernetes                  ClusterIP     10.0.0.1      <none>         443/TCP    76m
joshua [ ~ ]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
aks-agentpool-21221973-vmss000000 Ready     agent    72m    v1.23.12
aks-agentpool-21221973-vmss000001 Ready     agent    75m    v1.23.12
joshua [ ~ ]$
```

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

```
Bash
joshua [ ~ ]$ ls -l
total 40
-rw-r--r-- 1 joshua joshua 239 Nov 11 23:12 app-pod.yaml
-rw-r--r-- 1 joshua joshua 285 Nov 11 23:12 autoscale.yaml
lrwxrwxrwx 1 joshua joshua 22 Nov 12 00:12 clouddrive -> /usr/csuser/clouddrive
-rw-r--r-- 1 joshua joshua 550 Nov 11 23:13 deploy-svc.yaml
-rw-r--r-- 1 joshua joshua 129 Nov 11 23:13 easypay-namespace.yaml
-rw-r--r-- 1 joshua joshua 274 Nov 11 23:13 network-policy.yaml
-rw-r--r-- 1 joshua joshua 330 Nov 11 23:14 new-user-role.yaml
-rw-r--r-- 1 joshua joshua 369 Nov 12 01:20 postgres-pod.yaml
-rw-r--r-- 1 joshua joshua 257 Nov 12 01:23 postgres-svc.yaml
-rw-r--r-- 1 joshua joshua 224 Nov 12 01:19 redis-pod.yaml
-rw-r--r-- 1 joshua joshua 362 Nov 11 23:14 role-bind.yaml
joshua [ ~ ]$
```

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

```
apiVersion: v1
kind: Namespace
metadata:
  name: infra-optimize-ns
  labels:
    name: easypay-namespace
    app: easypay
```

```
Bash
joshua [ ~ ]$ kubectl get namespace
NAME                STATUS  AGE
default             Active  4h38m
infra-optimize-ns   Active  48m
kube-node-lease     Active  4h38m
kube-public         Active  4h38m
kube-system         Active  4h38m
joshua [ ~ ]$
```

- 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: easypay202211111100704.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

```

joshua [ ~ ]$ kubectl get all -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
pod/easypay-7d7f9fc7b5-lhghf	1/1	Running	0	57m	10.244.1.4	aks-agentpool1-21221973-vmss000000	<none>		<none>	

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
service/easypay	LoadBalancer	10.0.128.24	20.104.142.166	80:31529/TCP	57m	app=easypay
service/kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	6h53m	<none>

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
deployment.apps/easypay	1/1	1	1	57m	easypay-app	easypay202211111100704.azurecr.io/easypay:latest	app=easypay

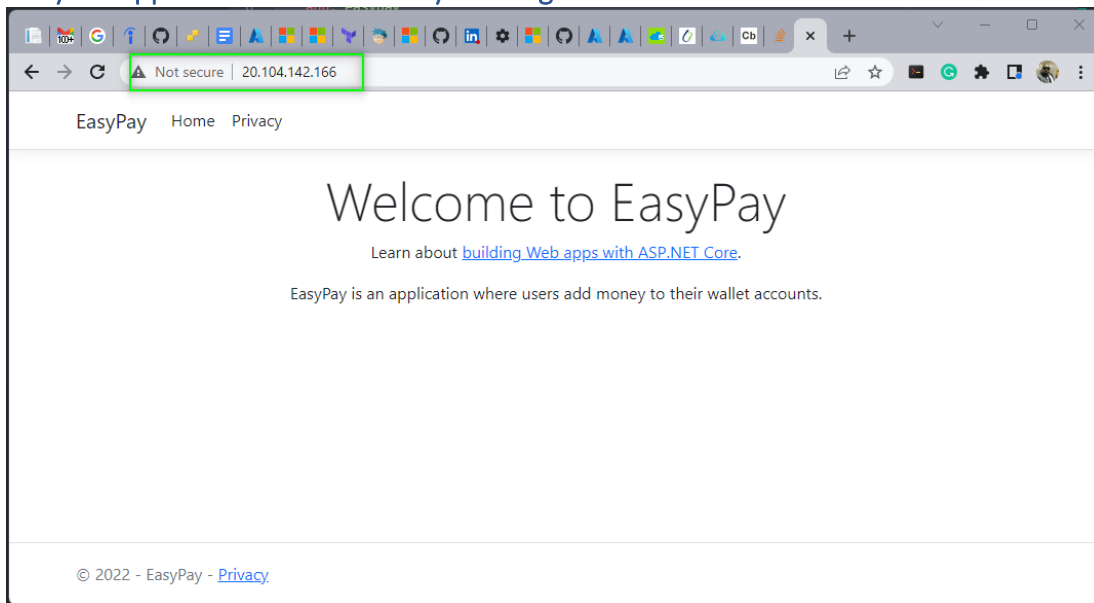
NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/easypay-7d7f9fc7b5	1	1	1	57m	easypay-app	easypay202211111100704.azurecr.io/easypay:latest	app=easypay,pod-template-hash=7d7f9fc7b5

```

joshua [ ~ ]$

```

- 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.



7. 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"

```
Bash
joshua [ ~ ]$ kubectl get networkpolicy --namespace=infra-optimize-ns
NAME          POD-SELECTOR  AGE
access-easypay  app=easypay  9m32s
```

- 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

```

```

joshua [ ~ ]$ kubectl get pod,svc --namespace=infra-optimize-ns
NAME                READY   STATUS    RESTARTS   AGE
pod/postgres-pod    1/1     Running   0           12m

NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/db          ClusterIP   10.0.37.119  <none>        5432/TCP   12m
joshua [ ~ ]$

```

- 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"]

```

```

joshua [ ~ ]$ kubectl create -f new-user-role.yaml
role.rbac.authorization.k8s.io/joshua-demebo created
joshua [ ~ ]$ 

```

■ 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: ""

```

```

rolebinding.rbac.authorization.k8s.io/role-bind created
joshua [ ~ ]$ 

```

- 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
```

```
joshua [ ~ ]$ kubectl create -f redis-pod.yaml
pod/redis-pod created
joshua [ ~ ]$ ls
```

- 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
```

```
joshua [ ~ ]$ kubectl create -f autoscale.yaml
horizontalpodautoscaler.autoscaling/auto-pod created
joshua [ ~ ]$
```

8. 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

```
joshua [ ~ ]$ kubectl get all -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
pod/easypay-7d7f9fc7b5-1hghf	1/1	Running	0	129m	10.244.1.4	aks-agentpool-21221973-vmss000000	<none>	<none>


```
joshua [ ~ ]$ kubectl get all -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
service/easypay	LoadBalancer	10.0.128.24	20.104.142.166	80:31529/TCP	129m	app=easypay
service/kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	8h	<none>


```
joshua [ ~ ]$ kubectl get all -o wide
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
deployment.apps/easypay	1/1	1	1	129m	easypay-app	easypay20221111100704.azurecr.io/easypay:latest	app=easypay


```
joshua [ ~ ]$ kubectl get all -o wide
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/easypay-7d7f9fc7b5	1	1	1	129m	easypay-app	easypay20221111100704.azurecr.io/easypay:latest	app=easypay,pod-template-hash=7d7f9fc7b5

■ kubectl get node -o wide

```
joshua [ ~ ]$ kubectl get node -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
aks-agentpool-21221973-vmss000000	Ready	agent	8h	v1.23.12	10.224.0.4	<none>	Ubuntu 18.04.6 LTS	5.4.0-1091-azure	containerd://1.5.11+azure-2
aks-agentpool-21221973-vmss000001	Ready	agent	8h	v1.23.12	10.224.0.5	<none>	Ubuntu 18.04.6 LTS	5.4.0-1091-azure	containerd://1.5.11+azure-2

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.