

FINAL USE CASE

DEPLOYING A STATIC “HELLO WORLD” WEB APPLICATION USING AZURE FREE TIER

Introduction:

This project offers a comprehensive, hands-on exploration of the modern DevOps ecosystem, with a strong emphasis on cloud infrastructure, containerization, and automation techniques. It's designed to guide learners through real-world practices, allowing them to gain a deeper understanding of how DevOps tools and workflows integrate to streamline application deployment and maintenance.

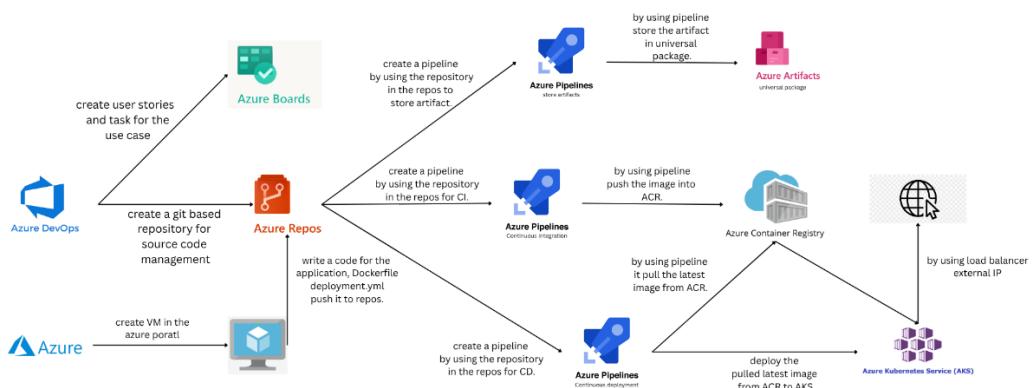
As part of this journey, you'll undertake a practical task: deploying a simple "Hello World" web application using basic HTML and CSS. This app will be hosted on Microsoft Azure's free tier services, giving you firsthand experience with cloud deployment while keeping costs at zero. The exercise not only builds foundational cloud skills but also lays the groundwork for more advanced DevOps applications.

Objective:

This project aims to equip you with *practical, real-world experience* in establishing a complete DevOps workflow from start to finish. Through each stage of this immersive exercise, you'll gain firsthand knowledge of deploying applications in a cloud environment using containers and orchestration tools—essential skills for today's DevOps engineer.

By the project's conclusion, you'll grasp the core principles of continuous integration and continuous delivery (CI/CD), while learning how to manage cloud resources within Azure's free-tier constraints to maintain a budget-friendly learning experience. This experiential journey is crafted to deepen your expertise in cloud-native development and contemporary deployment methodologies.

Architecture diagram:



Create azure boards:

The screenshot shows the Azure DevOps Boards interface for the 'inter-project' team. On the left, a sidebar lists various project management sections like Overview, Boards, Work items, Backlogs, Sprints, Queries, Delivery Plans, Analytics views, Repos, Pipelines, and Test Plans. The 'Boards' section is currently selected. The main area displays a backlog board titled 'inter-project Team'. The backlog is organized into columns: New (8 items), Active (0/5), Resolved (0/5), and Closed (0/5). A modal window titled '+ New item' is open, showing two sections: '8 required setup' and '12 creation of VM, NSG, storage account'. Both sections have a 'New' status and are assigned to 'Dineshkumar Selvan'. Under 'Required setup', there are three tasks: 'create resource group', 'create virtual network', and 'create subnet for Vnet', all marked as '0/3' completed.

Using Azure Boards, create user stories to track the workflow of the use case.

Required Setup and Installations for CI:

1. Azure Resource Group:

The screenshot shows the 'Create a resource group' wizard in the Azure Portal. At the top, there's a header bar with 'Microsoft Azure', 'Upgrade', a search bar, and user information. Below the header, the URL 'https://portal.azure.com/#create/Microsoft.ResourceGroup' is visible. The main page title is 'Create a resource group'. The 'Review + create' tab is selected. The 'Basics' section contains fields for 'Subscription' (Azure subscription 1), 'Resource group name' (DevEnvironment-RG), and 'Region' (East US). The 'Tags' section has a single tag 'None'. At the bottom, there are 'Previous', 'Next', and 'Create' buttons.

In the Azure Portal, search for "**Resource groups**" and click **+ Create**. Select your desired **subscription** and enter **DevEnvironment-RG** as the name. Choose **East US** as the region to host the group. Click **Review + Create**, then click **Create**.

"All resources for this use case are created under the resource group 'DevEnvironment-RG' using the Azure UI."

2. Virtual Network (VNet):

The screenshot shows the 'Create virtual network' wizard in the Azure portal. The 'Review + create' tab is selected. The configuration includes:

- Subscription:** Azure subscription 1
- Resource Group:** DevEnvironment-RG
- Name:** Dev-Vnet
- Region:** East US
- Security:** Azure Bastion: Disabled, Azure Firewall: Disabled, Azure DDoS Network Protection: Disabled
- IP addresses:** Address space: 10.0.0.0/16 (65,536 addresses), Subnet: default (10.0.0.0/24) (256 addresses)

At the bottom, there are 'Previous', 'Next', and 'Create' buttons, along with a 'Give feedback' link.

To create a virtual network named **Dev-Vnet**, log in to the Azure portal. Navigate to "Virtual Networks" and click **Create**. Under the "Basics" tab, set the name to *Dev-Vnet*, choose a resource group and region. In the "IP Addresses" tab, enter 10.0.0.0/16 as the address space. Review your settings, then click **Create** to deploy the network.

3. Subnet:

The screenshot shows the 'Edit subnet' dialog in the Azure portal. The left sidebar shows the 'Subnets' tab is selected. The 'Edit subnet' dialog contains the following fields:

- Subnet ID:** /subscriptions/440b8095-3209-4d1a-b715-d090abdfb487/resourceGroups/DevEnvironment-RG/providers/Microsoft.Network/virtualNetworks/Dev-Vnet/subnets/Default
- Subnet purpose:** Default
- Name:** Dev-subnet
- IPv4:** Include an IPv4 address space (checked), Starting address: 10.0.1.0, Size: /24 (256 addresses), Subnet address range: 10.0.1.0 - 10.0.1.255
- IPv6:** Include an IPv6 address space (unchecked)
- Private subnet:** This virtual network has no IPv6 address ranges.

At the bottom, there are 'Save' and 'Cancel' buttons, along with a 'Give feedback' link.

In the Azure Portal, go to **Virtual Networks** and select **Dev-Vnet**. Click on the **Subnets** tab in the left menu, then choose **+ Subnet**. Enter **Dev-subnet** as the name and 10.0.1.0/24 as the address range. Click **Save** to add the subnet to your virtual network.

4. Storage Account:

Subscription: Azure subscription 1
Resource group: DevEnvironment-RG
Location: East US
Storage account name: devstorage2398173
Primary service: Standard
Replication: Locally-redundant storage (LRS)

Advanced

Enable hierarchical namespace: Disabled
Enable SFTP: Disabled
Enable network file system v3: Disabled
Allow cross-tenant replication: Disabled
Access tier: Hot
Enable large file shares: Enabled

Previous Next Create Give feedback

In the Azure Portal, search for "**Storage accounts**" and click **+ Create**. Choose your subscription and resource group, then enter **devstorage2398173** as the name. Select **Standard** for performance and **Locally-redundant storage (LRS)** for redundancy. Review your settings and click **Create** to deploy the storage account.

5. Virtual Machine:

Validation passed

Help me create a low cost VM Help me create a VM optimized for high availability Help me choose the right VM size for my workload

Subscription: Azure subscription 1
Resource group: DevEnvironment-RG
Virtual machine name: Dev-vm
Region: East US
Availability options: Availability zone
Zone options: Self-selected zone
Availability zone: 2
Security type: Trusted launch virtual machines
Enable secure boot: Yes
Enable vTPM: Yes
Integrity monitoring: No
Image: Ubuntu Server 24.04 LTS - Gen2
VM architecture: x64
Size: Standard B1s (1 vcpu, 1 GiB memory)
Enable Hibernation: No

... Initializing deployment...
Initializing template deployment to resource group 'DevEnvironment-RG'.

< Previous Next > Create Download a template for automation Give feedback

Creating a virtual machine named "**Dev-vm**" using the Azure portal,

In **availability zone: 2**,

With the storage size: **Standard B1s (1 vcpu, 1 GiB memory)**,

Image: **Ubuntu 24.04LTS – Gen2**,

located in East US, connected to the **Dev-Vnet** virtual network and the **Dev-subnet** subnet, and adding **tags**: Project=DevEnvironment, Owner=azure-dev-user. For development environment.

6. Network Security Group (NSG):

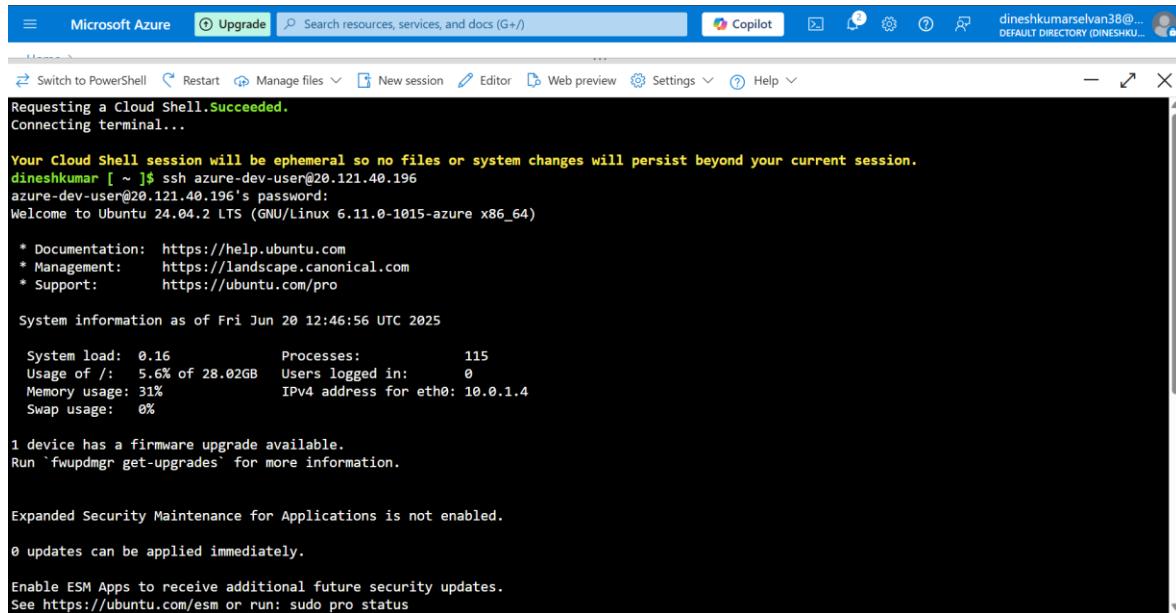
The screenshot shows the Azure portal interface for a Network Security Group named "Dev-NSG". The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Settings, Monitoring, Automation, and Help. The main content area displays the "Essentials" section with details such as Resource group (move) to "DevEnvironment-RG", Location as East US, Subscription (move) to "Azure subscription_1", and Subscription ID "440b8095-3209-4d1a-b715-d090abdfb487". It also shows custom security rules: 0 inbound, 0 outbound, associated with 0 subnets, and 0 network interfaces. A table lists three inbound security rules with priorities 65000, 65001, and 65500, all set to Allow. A table for outbound security rules is partially visible.

In the Azure Portal, search for "**Network Security Groups**" and click **+ Create**. Choose your subscription and resource group, then name it **Dev-NSG**. Select a region that matches your other resources (e.g., East US). Click **Review + Create**, then hit **Create** to deploy the NSG.

The screenshot shows the Azure portal interface for creating an inbound security rule for the "Dev-NSG" network security group. The left sidebar shows the "Inbound security rules" section. The right pane is titled "Add inbound security rule" for "Dev-NSG". It allows selecting protocol (Any, TCP, UDP, ICMPv4, ICMPv6) and action (Allow, Deny). A priority field is set to 100, and the name is specified as "port-80". The "Description" field is empty. At the bottom are "Add" and "Cancel" buttons, and a "Give feedback" link.

In the Azure Portal, go to **Network Security Groups** and select **Dev-NSG**. Click on **Inbound security rules**, then select **+ Add** to create a rule. Add a rule for **port 80** with priority **100**, and another for **port 20** with priority **110**. Set both to **Allow** and click **Add** to apply the rules.

7. Docker Installation on VM:



The screenshot shows a Microsoft Azure Cloud Shell interface. At the top, there's a header bar with 'Microsoft Azure', 'Upgrade', a search bar, 'Copilot', and user information ('dineshkumarselvan38@...'). Below the header is a toolbar with icons for 'Switch to PowerShell', 'Restart', 'Manage files', 'New session', 'Editor', 'Web preview', 'Settings', and 'Help'. The main area is a terminal window displaying a Cloud Shell session. The session starts with a success message: 'Requesting a Cloud Shell. Succeeded.' followed by 'Connecting terminal...'. It then displays system information for Ubuntu 24.04.2 LTS, including documentation links, system load, memory usage, and swap usage. A note about a firmware upgrade is shown, along with instructions to run 'fwupdmanager get-upgrades'. Security maintenance status is checked, and ESM Apps are enabled. The session ends with a note to see https://ubuntu.com/esm or run 'sudo pro status'.

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

Your Cloud Shell session will be ephemeral so no files or system changes will persist beyond your current session.
dineshkumar [ ~ ]$ ssh azure-dev-user@20.121.40.196
azure-dev-user@20.121.40.196's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-1015-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Fri Jun 20 12:46:56 UTC 2025

 System load: 0.16      Processes:          115
 Usage of /: 5.6% of 28.02GB  Users logged in:   0
 Memory usage: 31%           IPv4 address for eth0: 10.0.1.4
 Swap usage:  0%

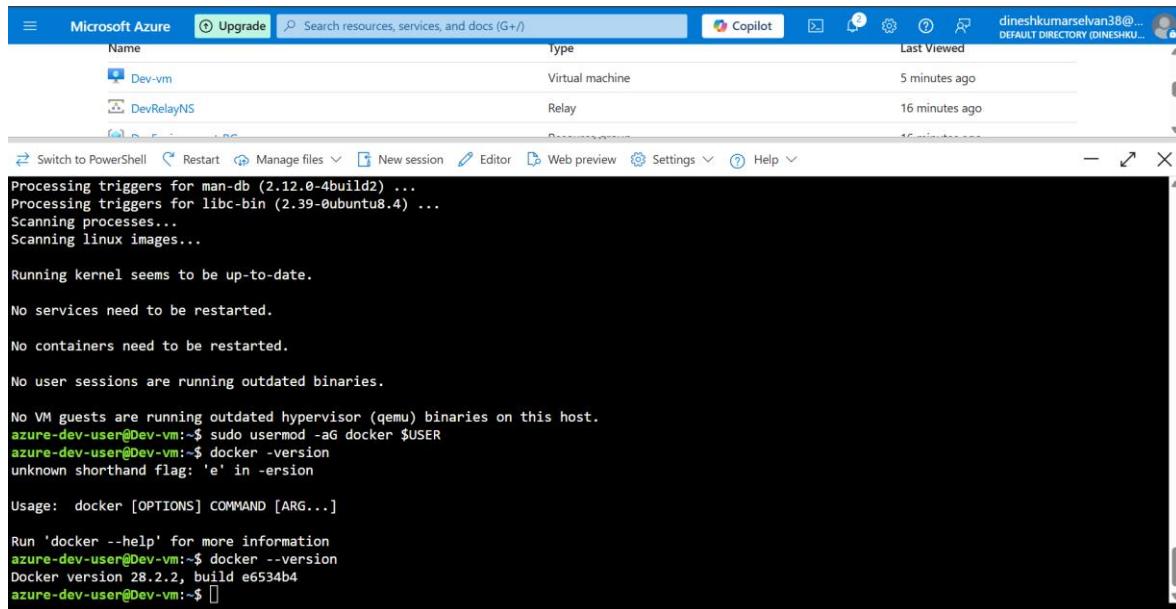
1 device has a firmware upgrade available.
Run `fwupdmanager get-upgrades` for more information.

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
```

In the Cloud Shell, ensure you have SSH access enabled on the VM. Use the command: ssh azure-dev-user@20.121.40.196 to start the connection. If prompted, type "yes" to accept the fingerprint and enter the password if required. Once authenticated, you'll be logged into the **Dev-vm** via SSH.



This screenshot shows a Microsoft Azure Cloud Shell interface connected to a virtual machine named 'Dev-vm'. The top header bar and toolbar are identical to the previous screenshot. The main terminal window shows a series of commands being run on the 'Dev-vm' host. It starts with processing triggers for man-db and libc-bin, followed by scanning processes and linux images. It then checks the running kernel, services, and containers, all of which are reported as up-to-date and restarted. It lists user sessions and VM guests. Finally, it runs the 'docker --version' command, which outputs the Docker version 28.2.2 and build e6534b4.

```
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

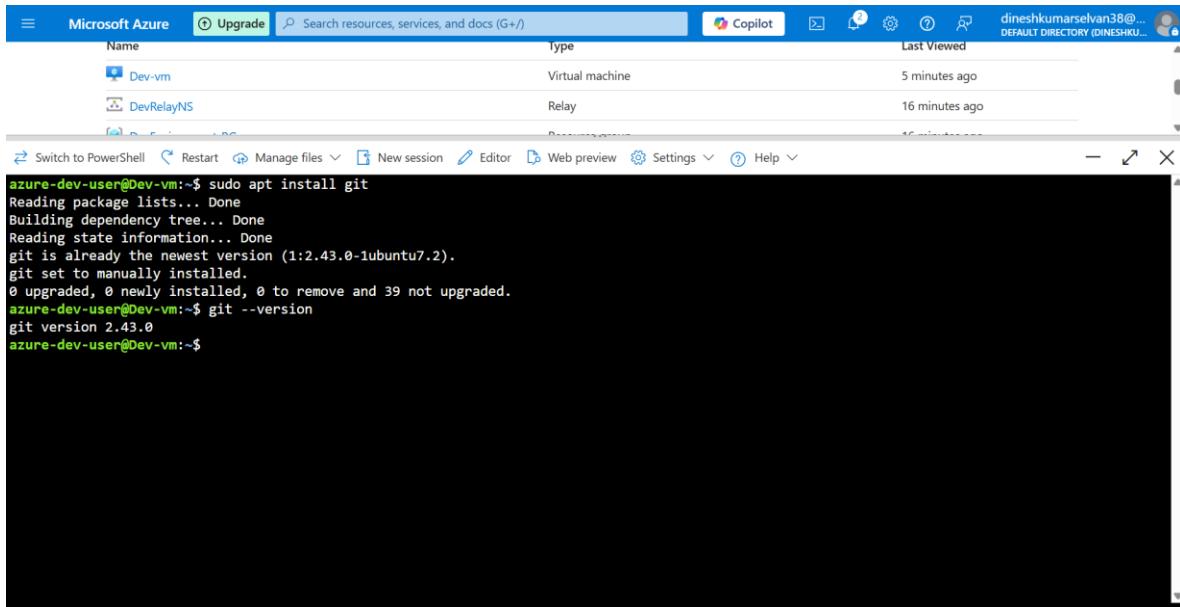
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
azure-dev-user@Dev-vm:~$ sudo usermod -aG docker $USER
azure-dev-user@Dev-vm:~$ docker --version
unknown shorthand flag: 'e' in -ersion
Usage: docker [OPTIONS] COMMAND [ARG...]

Run 'docker --help' for more information
azure-dev-user@Dev-vm:~$ docker --version
Docker version 28.2.2, build e6534b4
azure-dev-user@Dev-vm:~$ 
```

In the Dev-vm terminal, start by updating package info: sudo apt update. Install required dependencies: sudo apt install ca-certificates curl gnupg. Use the official script or repository to install Docker version **28.2.2** specifically. Verify installation with docker --version to confirm it's set up correctly.

8. Git Installation on VM:

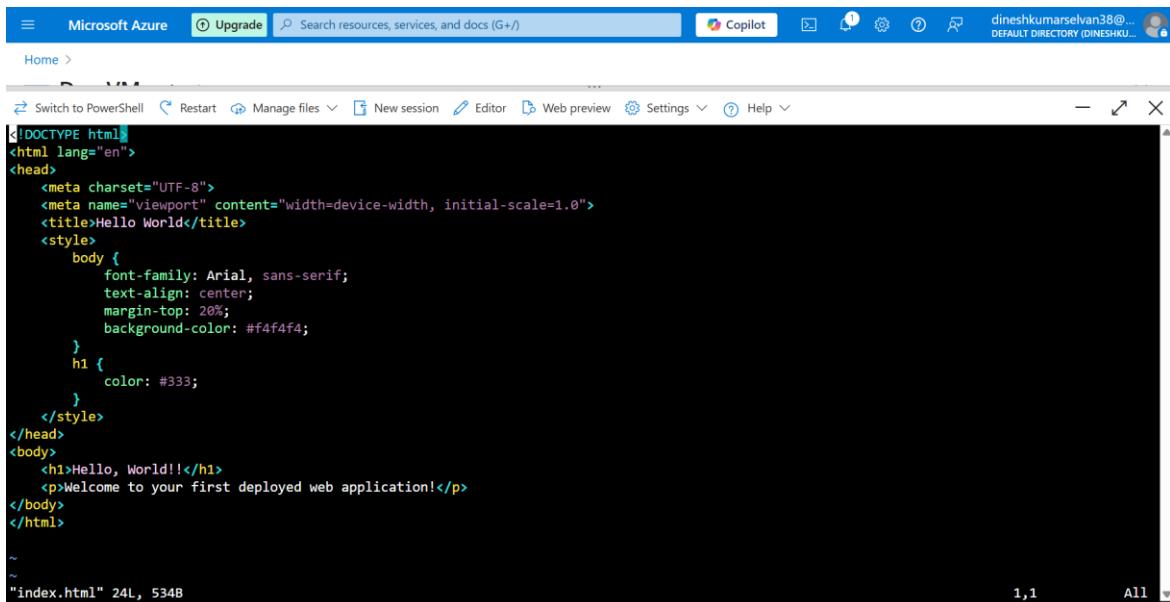


A screenshot of the Microsoft Azure Dev VM terminal window. The title bar shows "Microsoft Azure" and "az Dev-vm". The terminal content shows the output of a command to install git via apt:

```
azure-dev-user@Dev-vm:~$ sudo apt install git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.43.0-1ubuntu7.2).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
azure-dev-user@Dev-vm:~$ git --version
git version 2.43.0
azure-dev-user@Dev-vm:~$
```

In the Dev-vm terminal, first update the package list using sudo apt update. Download Git version **2.43.0** source from the official repository or GitHub. Compile and install it with make and sudo make install after extraction. Verify the installation using git --version to confirm it's 2.43.0.

9. HTML and CSS Web Application:



A screenshot of the Microsoft Azure Dev VM Editor window. The title bar shows "Microsoft Azure" and "index.html". The editor content displays the HTML and CSS code for a "Hello World" web application:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hello World</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 20%;
            background-color: #f4f4f4;
        }
        h1 {
            color: #333;
        }
    </style>
</head>
<body>
    <h1>Hello, World!!</h1>
    <p>Welcome to your first deployed web application!</p>
</body>
</html>
```

At the bottom right, the status bar shows "1,1" and "All".

I created a 'Hello World' web application using HTML and inline CSS in the name index.html inside the hello-world-app folder. I then used Git for source code management and pushed the code to Azure Repos.

HTML and CSS:

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

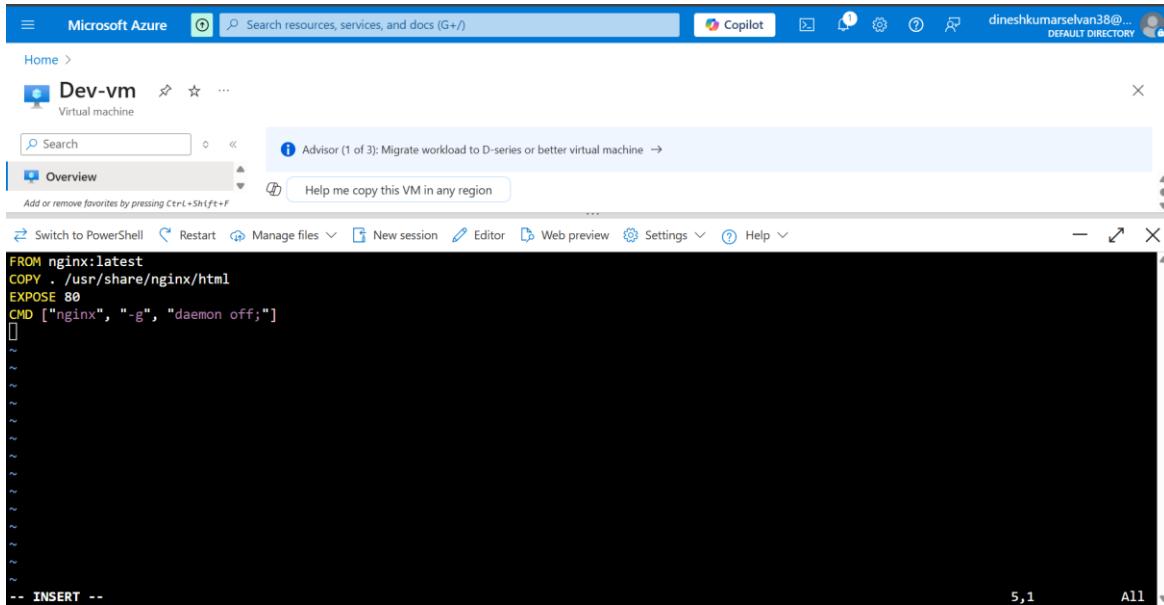
<title>Hello World</title>
```

```

<style>
body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin-top: 20%;
    background-color: #f4f4f4;
}
h1 {
    color: #333;
}
</style>
</head>
<body>
<h1>Hello, World!!</h1>
<p>Welcome to your first deployed web application!</p>
</body>
</html>

```

10. Dockerfile:



```

FROM nginx:latest
COPY . /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

Create Dockerfile:

Dockerfile:

```

FROM nginx:latest
COPY . /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

```

create mode 100644 Dockerfile
azur...@Dev-vm:~/hello-world-app$ git pull
Password for 'https://dineshkumarselvan38@dev.azure.com':
Already up to date.
azur...@Dev-vm:~/hello-world-app$ git push
Password for 'https://dineshkumarselvan38@dev.azure.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 411 bytes | 411.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Analyzing objects... (3/3) (4 ms)
remote: Validating commits... (1/1) done (0 ms)
remote: Storing packfile... done (54 ms)
remote: Storing index... done (43 ms)
remote: Updating refs... done (140 ms)
remote: We noticed you're using an older version of Git. For the best experience, upgrade to a newer version.
To https://dev.azure.com/dineshkumarselvan38/inter-project/_git/inter-project
  90360dd..1a84a75 master -> master
azur...@Dev-vm:~/hello-world-app$ 

```

Push the dockerfile in azure repos.

Name	Last change	Commits
Dockerfile	2m ago	1a84a75b add the docker f...
index.html	52m ago	90360dd4 source code cre...

After the push operation the files appeared in the azure repos.

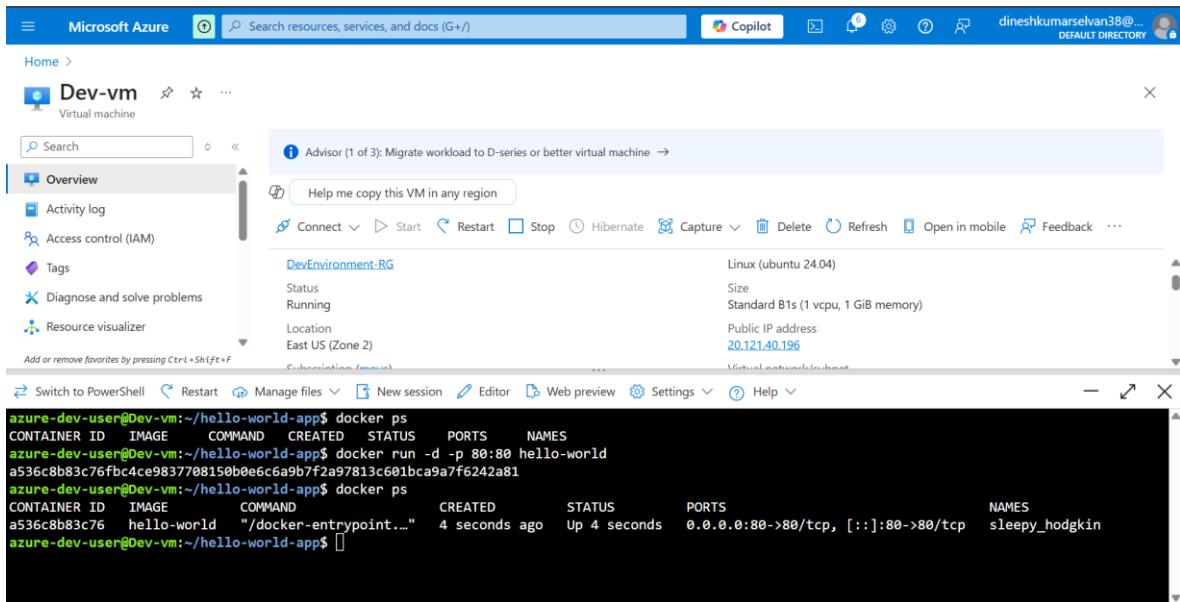
11. Docker Commands:

```

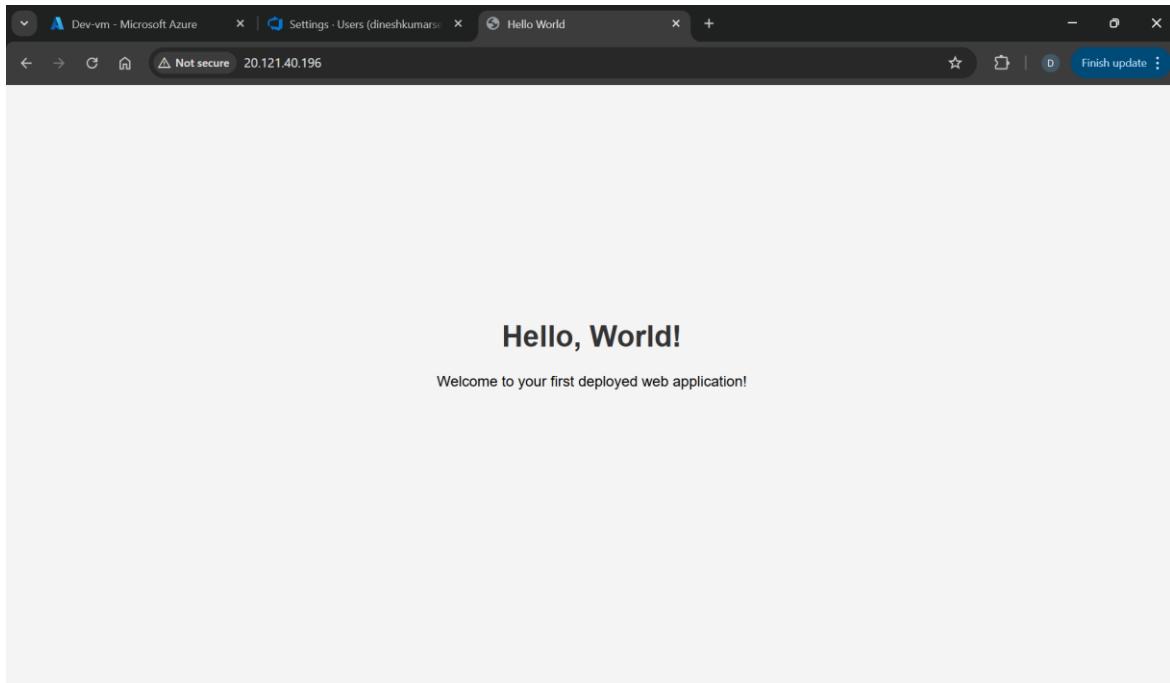
azur...@Dev-vm:~/hello-world-app$ docker build -t hello-world .
[+] Building 0.5s (7/7) FINISHED
--> [internal] load build definition from Dockerfile
--> = transferring dockerfile: 130B
--> [internal] load metadata for docker.io/library/nginx:latest
--> [internal] load .dockerignore
--> => transferring context: 2B
--> [internal] load build context
--> => transferring context: 33.14kB
--> [1/2] FROM docker.io/library/nginx:latest@sha256:6784fb0834aa7dbbe12e3d7471e69c290df3e6ba810dc38b34ae33d3c1c05f7d
--> => CACHED [2/2] COPY . /usr/share/nginx/html
--> => exporting to image
--> => => exporting layers
--> => writing image sha256:39ed0b7d3e80cfecab3b5a7c4263669c65de6cc247215c303760be78da7424011
--> => naming to docker.io/library/hello-world
azur...@Dev-vm:~/hello-world-app$ 

```

In the Dev-vm terminal, navigate to the directory containing your Dockerfile. Run docker build -t hello-world . to build the image and tag it as **hello-world**. This command reads the Dockerfile and packages the application into a container image. Once completed, check the image with docker images to confirm it's listed.



In the Dev-vm terminal, use the docker run command to start the container. Use the -d flag to run it in detached (background) mode. Use the -p flag to map a host port to a container port, like -p 8080:80. The full command looks like: docker run -d -p 8080:80 hello-world.



Open your web browser and enter the public IP of the VM followed by port 80. For example: <http://20.121.40.196> (since port 80 is the default for HTTP). If everything is configured correctly, the web page or app will load. This confirms your Docker container is serving traffic successfully.

12. Azure Container Registry (ACR):

The screenshot shows the Azure Portal interface for managing a Container Registry. The top navigation bar includes 'Microsoft Azure', 'Upgrade', a search bar, and user information ('dineshkumarselvan38@...'). The main page displays the 'devenvironmentacr38' container registry under the 'Container registry' category. The left sidebar provides navigation links for Overview, Activity log, Access control (IAM), Tags, Quick start, Resource visualizer, Events, Settings, and Properties. The 'Overview' section is selected, showing details like Resource group (move) to 'DevEnvironment-RG', Location as 'East US', Subscription (move) to 'Azure subscription 1', and Pricing plan as 'Standard'. The right panel displays the 'Essentials' section with fields for Login server, Location, Creation date, Provisioning state, Pricing plan, and Domain name label scope. A 'Get started' tab is active, followed by 'Monitoring', 'Capabilities (9)', and 'Tutorials'. A banner at the bottom states 'Simplify container lifecycle management'.

In the Azure Portal, search for "**Container Registries**" and click **+ Create**. Select your subscription and resource group, then name it **devenvironmentacr38**. Choose a region (same as your other resources), and keep the SKU as **Basic** for simplicity. Click **Review + Create**, then hit **Create** to deploy your ACR.

Step-by-Step CI Tasks:

1. Continuous Integration (CI): Build Pipeline:

Create a New Build Pipeline in Azure DevOps:

The screenshot shows the Azure DevOps pipeline configuration interface. On the left, the sidebar lists 'inter-project' under Pipelines, along with Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area is titled 'Configure your pipeline' and shows a list of available pipeline templates: Docker (Build a Docker image), Docker (Build and push an image to Azure Container Registry), Deploy to Azure Kubernetes Service (Build and push image to Azure Container Registry; Deploy to Azure Kubernetes Service), Starter pipeline (Start with a minimal pipeline that you can customize to build and deploy your code), and Existing Azure Pipelines YAML file (Select an Azure Pipelines YAML file in any branch of the repository). To the right, a detailed configuration pane for the 'Docker' template is open, showing fields for 'Container registry' (set to 'devenvironmentacr38'), 'Image Name' (set to 'finalusecase'), and 'Dockerfile' (set to '\$(Build.SourcesDirectory)/Dockerfile'). At the bottom of the pane are 'Back' and 'Validate and configure' buttons.

In your Azure DevOps repo, create a YAML file under `.azure-pipelines/`. Define steps to docker build your app and docker push to ACR using service connection. Include loginServer from your ACR and use variables for image name and tag. Run the pipeline to build the image and push it securely to **devenvironmentacr38**.

The screenshot shows the Azure DevOps interface with the following details:

- Project Path:** inter-project / final-usecase
- File Type:** azure-pipelines.yml
- Content Preview:**

```

trigger:
- master

resources:
- repo: self

variables:
# Container registry service connection established during pipeline creation
dockerRegistryServiceConnection: '3b481201-8d73-4a80-9bfa-a6374b789bbb'
imageRepository: 'finalusecase'
containerRegistry: 'devenvironmentacr38.azurecr.io'
dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
tag: '$(Build.BuildId)'

# Agent VM image name
vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: Build and push stage
  jobs:
    - job: Build
      displayName: Build
      pool:
        vmImage: $(vmImageName)
      steps:
        - task: Docker@2
          displayName: Build and push an image to container registry
          inputs:
            command: buildAndPush
            repository: $(imageRepository)
            image: $(dockerfilePath)

```
- Actions:** Edit, ...
- Bottom Bar:** https://dev.azure.com/dineshkumarselvan38/inter-project/_git/final-usecase?path=/azure-pipelines.yml&version=GBmaster

After the creation of the YAML pipeline it add azure-pipeline.yml file in the azure repos.

azure-pipeline.yml:

```

# Docker

# Build and push an image to Azure Container Registry

# https://docs.microsoft.com/azure/devops/pipelines/languages/docker

trigger:
- master

resources:
- repo: self

variables:
# Container registry service connection established during pipeline creation
dockerRegistryServiceConnection: '3b481201-8d73-4a80-9bfa-a6374b789bbb'
imageRepository: 'finalusecase'
containerRegistry: 'devenvironmentacr38.azurecr.io'
dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
tag: '$(Build.BuildId)'

# Agent VM image name

vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: Build and push stage
  jobs:
    - job: Build
      displayName: Build
      steps:
        - task: Docker@2
          displayName: Build and push an image to container registry
          inputs:
            command: buildAndPush
            repository: $(imageRepository)
            image: $(dockerfilePath)

```

pool:

vmImage: \$(vmImageName)

steps:

- task: Docker@2

displayName: Build and push an image to container registry

inputs:

command: buildAndPush

repository: \$(imageRepository)

dockerfile: \$(dockerfilePath)

containerRegistry: \$(dockerRegistryServiceConnection)

tags: |

\$(Build.BuildId)

- task: PublishBuildArtifacts@1

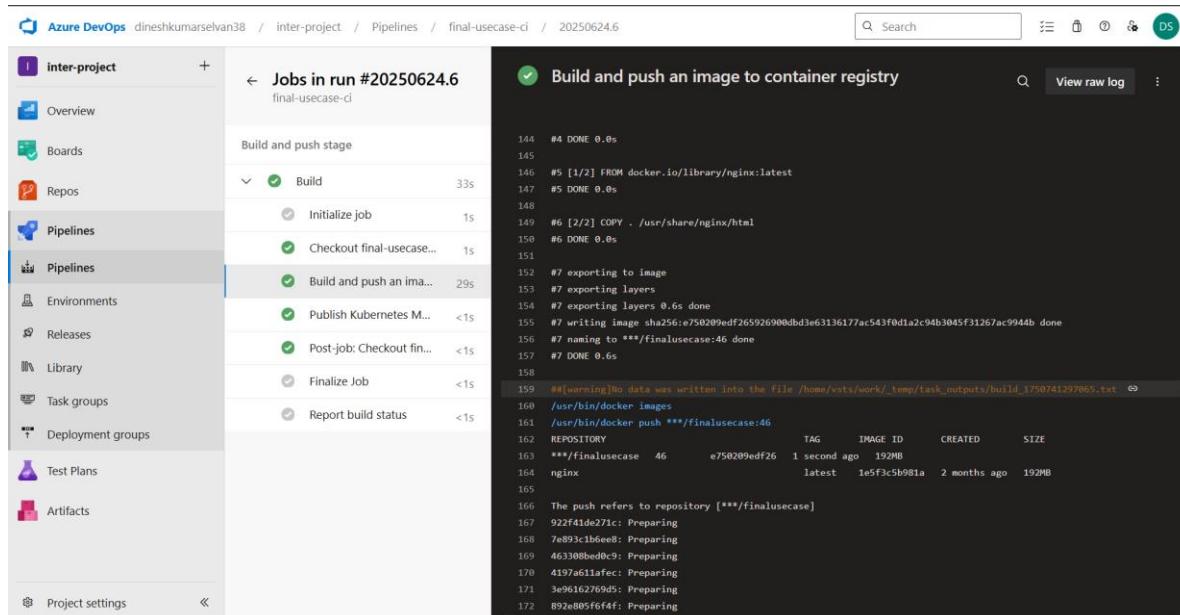
displayName: 'Publish Kubernetes Manifest'

inputs:

PathToPublish: '\$(Build.SourcesDirectory)/deployment.yaml'

ArtifactName: 'drop'

publishLocation: 'Container'



The screenshot shows the Azure DevOps interface for a pipeline named 'final-usecase-ci'. The left sidebar is visible with various project management and pipeline-related sections like Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area displays the log for a specific job in a run. The job title is 'Build and push an image to container registry'. The log output shows the following steps and their execution times:

- Build: 33s
- Initialize job: 1s
- Checkout final-usecase...: 1s
- Build and push an ima...: 29s
- Publish Kubernetes M...: <1s
- Post-job: Checkout fin...: <1s
- Finalize job: <1s
- Report build status: <1s

The log also includes several Docker commands and their outputs, such as:

- #4 DONE 0.0s
- #5 [1/2] FROM docker.io/library/nginx:latest
- #5 DONE 0.0s
- #6 [2/2] COPY . /usr/share/nginx/html
- #6 DONE 0.0s
- #7 exporting to image
- #7 exporting layers
- #7 exporting layers 0.6s done
- #7 writing image sha256:e750209edf265926900dbd3e63136177ac543f3045f31267ac994eb done
- #7 naming to ***/finalusecase:46 done
- #7 DONE 0.6s
- #8 [warning]No data was written into the file /home/vsts/work/_temp/task_outputs/build_1750741297965.txt
- /usr/bin/docker images
- /usr/bin/docker push ***/finalusecase:46
- REPOSITORY TAG IMAGE ID CREATED SIZE
- ***/finalusecase 46 e750209edf26 1 second ago 192MB
- nginx latest 1e5f3c5b981a 2 months ago 192MB
- The push refers to repository [***/finalusecase]
- 922f41de271c: Preparing
- 7e093c1b6ee8: Preparing
- 463308bed0d9: Preparing
- 4197a611afec: Preparing
- 3e9616226945: Preparing
- 892e805f6f4f: Preparing

The build was success.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure', 'Upgrade', and a search bar. On the right, it shows the user 'dineshkumarselvan38@...' and 'DEFAULT DIRECTORY (DINESH...)'.

The main area is titled 'devenvironmentacr38 | Repositories'. On the left, there's a sidebar with various options like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Quick start', 'Resource visualizer', 'Events', 'Settings', 'Services', and 'Repositories' (which is selected). Below the sidebar is a message: 'Add or remove favorites by pressing Ctrl+Shift+F'.

In the center, there's a search bar and a list of repositories. One repository, 'finalusecase', is highlighted. The details for 'finalusecase' are shown on the right, including its tag count (6) and manifest count (6), with a table listing the tags and their details.

After the successful build, the Docker image was tagged as **finalusecase**. It was then pushed securely to the Azure Container Registry (ACR). Navigate to your ACR and open the **Repositories** section. You'll find **finalusecase** listed, confirming the image is stored and ready.

2. store the artifact in azure artifact using pipeline:

The screenshot shows the Azure DevOps Pipelines interface. The left sidebar has 'inter-project' selected under 'Pipelines', along with other options like 'Overview', 'Boards', 'Repos', 'Artifacts', and 'Project settings'.

The main area shows a pipeline named 'store-artifact' for the 'master' branch of the 'final-usecase / azure-pipelines-2.yml' repository. The pipeline YAML code is displayed on the left, and the right side shows a list of available tasks:

- .NET Core**: Build, test, package, or publish a .NET application.
- Android signing**: Sign and align Android APK files.
- Ant**: Build with Apache Ant.
- App Center distribute**: Distribute app builds to testers and users via Visual Studio App Center.
- App Center test**: Test app packages with Visual Studio App Center.
- Archive files**: Compress files into .7z, tar.gz, or zip.
- ARM template deployment**: Deploy an Azure Resource Manager (ARM) template.
- Azure App Configuration Export**: Export key-values from Azure App Configuration.

Create a starter pipeline and add the file copy and universal package task using assistance using YAML.

azure-pipeline-2.yml:

```
# Starter pipeline
```

```
# Start with a minimal pipeline that you can customize to build and deploy your code.
```

```
Add steps that build, run tests, deploy, and more:
```

```
# https://aka.ms/yaml
```

```
trigger:
```

```
- master
```

```
pool:
```

```
vmImage: ubuntu-latest
```

steps:

- script: echo Hello, world!

displayName: 'Run a one-line script'

- script: |

echo Add other tasks to build, test, and deploy your project.

echo See <https://aka.ms/yaml>

displayName: 'Run a multi-line script'

- task: CopyFiles@2

inputs:

SourceFolder: '\$(Build.SourcesDirectory)/artifact'

Contents: '**'

TargetFolder: 'artifact'

- task: UniversalPackages@0

inputs:

command: 'publish'

publishDirectory: '\$(Build.ArtifactStagingDirectory)'

feedsToUsePublish: 'internal'

vstsFeedPublish: 'f63b2108-f532-4e01-a7f3-af84defac2a8/3a5af7b7-c299-4e88-988a-e9622664a6d8'

vstsFeedPackagePublish: 'my-package'

versionOption: 'patch'

The screenshot shows the Azure DevOps interface for a pipeline named 'inter-project'. The left sidebar lists various project sections like Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The 'Pipelines' section is currently selected. The main area displays a list of jobs for a specific run. The first job, 'Job', is expanded, showing its steps: Initialize job, Checkout final-usecase..., Run a one-line script, Run a multi-line script, CopyFiles, UniversalPackages, Post-job: Checkout fin..., Finalize Job, and Report build status. The total duration for this job was 19 seconds. Other collapsed jobs include 'Initialize job' and 'Checkout final-usecase...'. At the top right, there is a search bar and several icons for managing the pipeline.

The build was success.

The screenshot shows the Azure DevOps interface for managing artifacts. On the left, there's a sidebar with links for Overview, Boards, Repos, Pipelines, Test Plans, and Artifacts. The Artifacts link is highlighted. The main area shows a feed named 'app-artifact'. A search bar at the top has 'app-artifact' selected. Below it are buttons for 'Connect to Feed', '+ Create Feed', 'Search Upstream Sources', 'Recycle Bin', and settings. A table lists packages, with one entry: 'my-package' (Version 0.0.3) listed under 'This feed'. The table has columns for Type, Package, Views, and Source.

After a successful build, the pipeline publishes the artifact to **Azure Artifacts**. Navigate to **Azure DevOps > Artifacts**, and select your feed. You'll find the artifact listed with its version (e.g., 1.0.0 or custom tag). This confirms versioned artifacts are stored and available for use or deployment.

Required Setup and Installations for CD:

1. create AKS:

This screenshot shows the 'Create Kubernetes cluster' wizard in the Azure Portal. It's step 2 of 3, titled 'Configure cluster settings'. The configuration includes:

- Subscription:** Azure subscription 1
- Resource group:** DevEnvironment-RG
- Region:** Central US
- Kubernetes cluster name:** Dev-AKS
- Kubernetes version:** 1.31.8
- Automatic upgrade:** patch
- Automatic upgrade scheduler:** Every week on Sunday (recommended)
- Node security channel type:** NodlImage
- Security channel scheduler:** Every week on Sunday (recommended)

Node pools:

- Node pools:** 1
- Enable virtual nodes:** Disabled

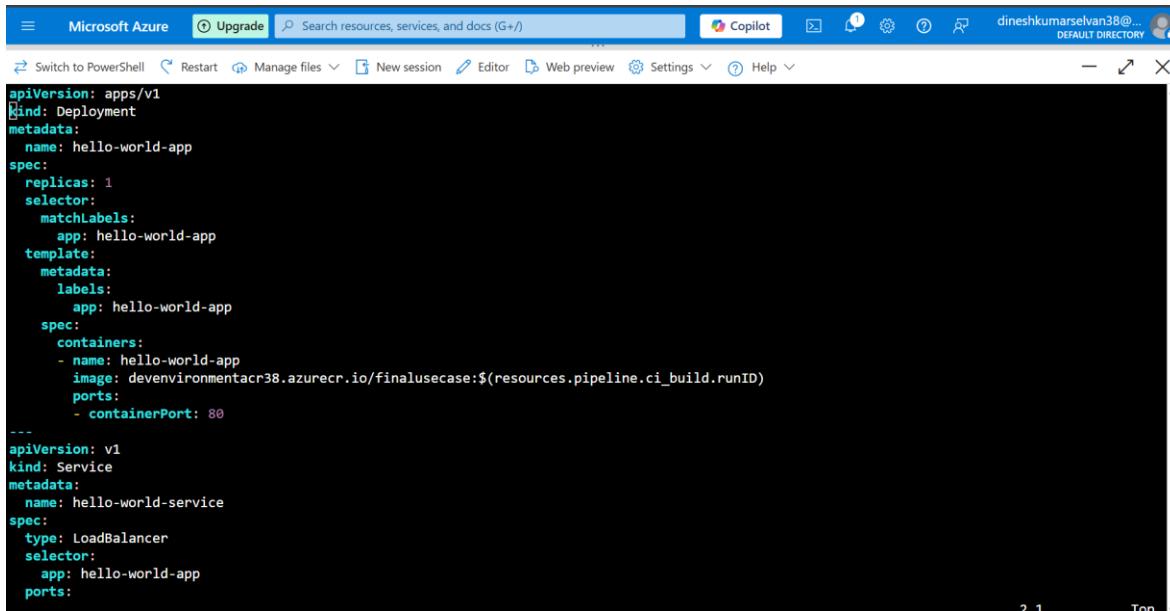
Access:

- Resource identity:** System-assigned managed identity

At the bottom, there are 'Previous', 'Next', and 'Create' buttons, along with a 'Give feedback' link.

In the Azure Portal, search for "**Kubernetes services**" and click **+ Create**. Choose your subscription, resource group, and name the cluster **Dev-AKS**. Set the node pool to have **1 node** and choose the **Standard_D2ps_v6** VM size. Click **Review + Create**, then hit **Create** to deploy the AKS cluster.

2. Kubernetes Deployment Manifest:



The screenshot shows the Microsoft Azure Cloud Shell interface. The title bar says "Microsoft Azure" and "Upgrade". The search bar says "Search resources, services, and docs (G+ /)". The top right shows the user "dineshkumarselvan38@..." and "DEFAULT DIRECTORY". Below the title bar are standard cloud shell navigation icons: "Switch to PowerShell", "Restart", "Manage files", "New session", "Editor", "Web preview", "Settings", "Help", and a Copilot icon. The main area of the window displays a YAML file named "deployment.yml". The file defines a Deployment and a Service. The Deployment "hello-world-app" has one replica, selects pods with "app: hello-world-app", and runs a container "hello-world-app" with port 80. The Service "hello-world-service" uses a LoadBalancer type and selects the same pods.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world-app
  template:
    metadata:
      labels:
        app: hello-world-app
    spec:
      containers:
        - name: hello-world-app
          image: devenvironmentacr38.azurecr.io/finalusecase:${resources.pipeline.ci_build.runID}
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: hello-world-service
spec:
  type: LoadBalancer
  selector:
    app: hello-world-app
  ports:
```

Create the deployment.yml file with service for the load balancer type in it and push it to the azure repos.

deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world-app
  template:
    metadata:
      labels:
        app: hello-world-app
    spec:
      containers:
        - name: hello-world-app
          image: devenvironmentacr38.azurecr.io/finalusecase:${resources.pipeline.ci_build.runID}
          ports:
            - containerPort: 80
---
apiVersion: v1
```

kind: Service

metadata:

name: hello-world-service

spec:

type: LoadBalancer

selector:

app: hello-world-app

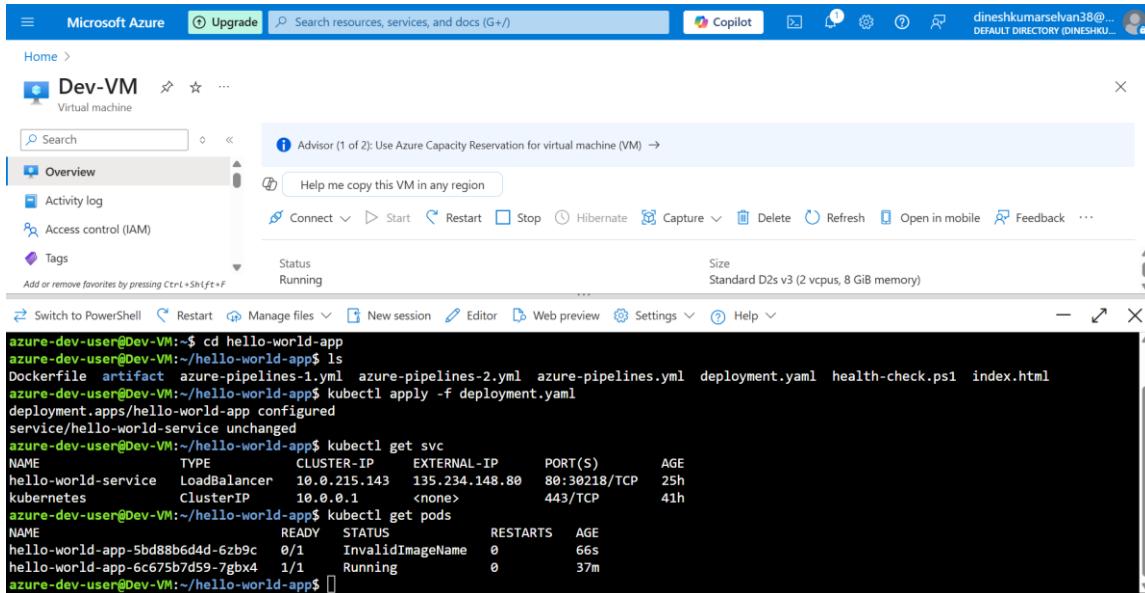
ports:

- **protocol: TCP**

port: 80

targetPort: 80

3. run the cluster in VM:

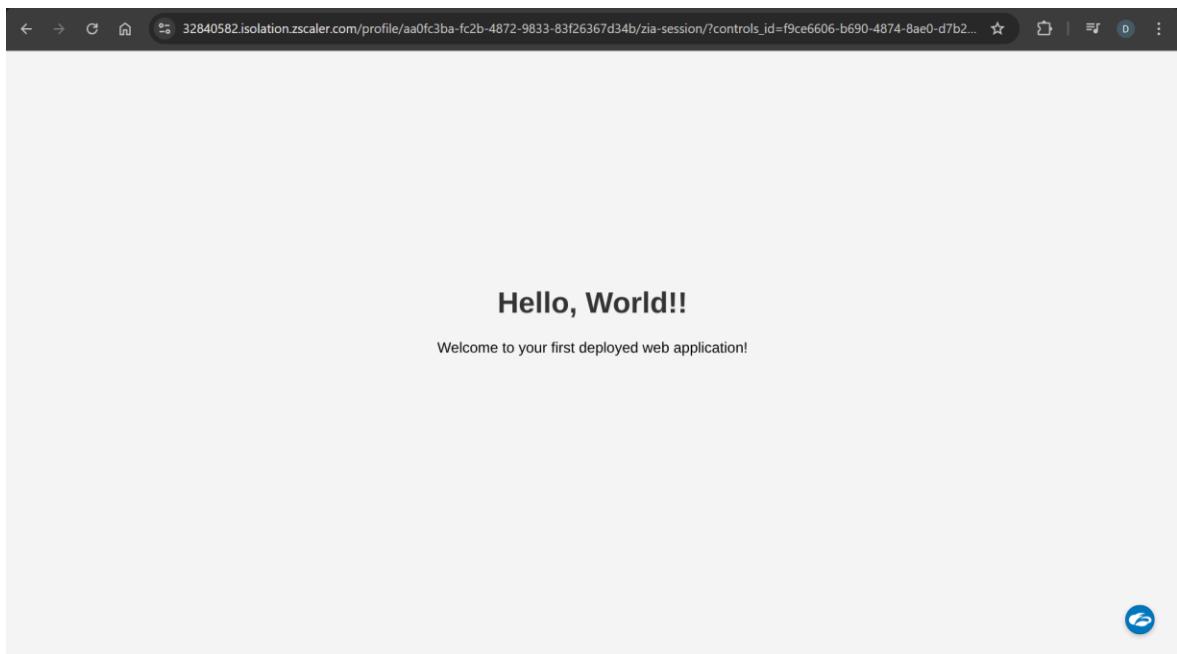


```
azur...@Dev-VM:~$ cd hello-world-app
azur...@Dev-VM:~/hello-world-app$ ls
Dockerfile artifact azure-pipelines-1.yml azure-pipelines-2.yml azure-pipelines.yml deployment.yaml health-check.ps1 index.html
azur...@Dev-VM:~/hello-world-app$ kubectl apply -f deployment.yaml
deployment.apps/hello-world-app configured
service/hello-world-service unchanged
azur...@Dev-VM:~/hello-world-app$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP     PORT(S)        AGE
hello-world-service   LoadBalancer   10.0.215.143  135.234.148.80  80:30218/TCP  25h
kubernetes       ClusterIP    10.0.0.1     <none>          443/TCP       41h
azur...@Dev-VM:~/hello-world-app$ kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
hello-world-app-5bd88b6d4d-6zb9c  0/1    InvalidImageName  0          66s
hello-world-app-6c675b7d59-7gbx4  1/1    Running   0          37m
azur...@Dev-VM:~/hello-world-app$
```

By using **kubectl apply** command we can able apply a configuration to a resource by file, **-f** by using this tag we can mention filename, **deployment.yaml** is the file we have the instruction in YAML format.

By using **kubectl get pods** to show the running pods.

By using **kubectl get svc** to show the running services.



In the Azure Portal, go to **Kubernetes services** and select your AKS cluster. Navigate to **Services and Ingress** under the **Workloads** menu. Locate the service exposing your app and note the **External IP** assigned. Open a web browser and enter that IP—your deployed web app should appear!

4. connection between Azure Kubernetes Service (AKS) and Azure Container Registry (ACR):

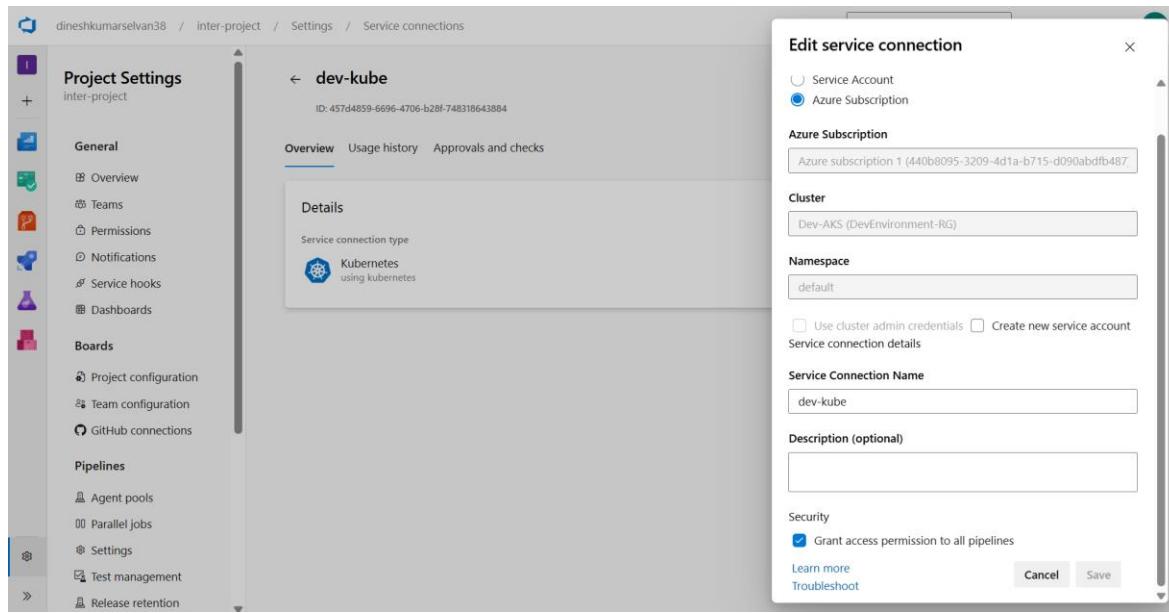
A screenshot of the Microsoft Azure portal. On the left, the navigation sidebar shows 'Dev-AKS' as the selected Kubernetes service. Under the 'Essentials' section, the 'Container registries' field is highlighted with the value 'devenvironmentacr38'. To the right, a 'Notifications' panel is open, displaying a single green checkmark notification: 'Successfully attached the container registry' with the subtext 'Successfully attached the container registry 'devenvironmentacr38''. The timestamp 'an hour ago' is also visible.

In the Azure Portal, go to your **Dev-AKS** Kubernetes cluster. Under **Settings**, select **Integrations**, then click on **Container registry**. Choose your registry, such as **devenvironmentacr38**, from the dropdown list. Click **Apply** to establish the connection between AKS and your ACR.

Step-by-Step CD Tasks:

1. Continuous Deployment (CD): Build Pipeline:

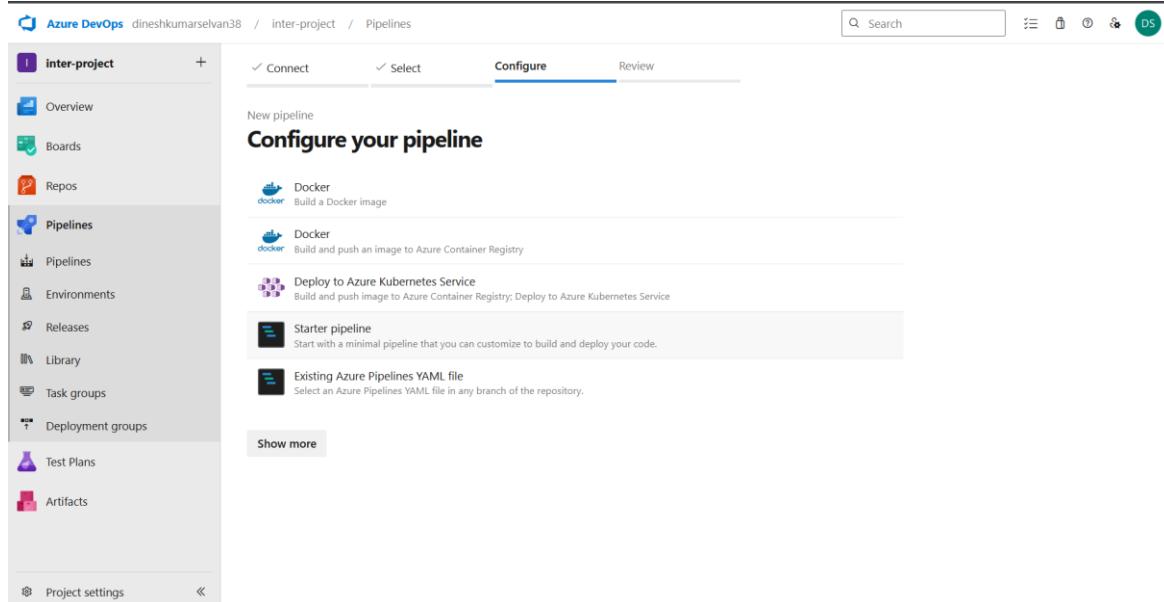
1.1 Create a service connection for Kubernetes:



The screenshot shows the Azure Portal interface. On the left, there's a sidebar with 'Project Settings' for 'inter-project'. In the main area, a 'Service connections' blade is open, showing a 'dev-kube' entry. The 'Overview' tab is selected, displaying the ID: 457d4859-6696-4706-b28f-748310643884. A detailed configuration modal is overlaid on the page, titled 'Edit service connection'. It shows the 'Service connection type' as 'Kubernetes using kubernetes'. Under 'Cluster', 'Dev-AKS (DevEnvironment-RG)' is selected. The 'Namespace' field contains 'default'. There are checkboxes for 'Use cluster admin credentials' and 'Create new service account', both of which are unchecked. The 'Service Connection Name' is set to 'dev-kube'. A 'Description (optional)' field is present but empty. At the bottom, there are 'Cancel' and 'Save' buttons, with 'Save' being highlighted.

In the Azure Portal, navigate to **Kubernetes services** from the homepage or search bar. Choose the correct **subscription** from the dropdown at the top of the page. From the list of clusters, locate and click on the **Dev-AKS** cluster you created. This opens the overview and configuration settings for managing your AKS cluster.

1.2 Create a New Build Pipeline in Azure DevOps:



The screenshot shows the Azure DevOps Pipelines interface. On the left, there's a sidebar with 'Pipelines' selected. The main area shows a 'Configure your pipeline' screen with a 'New pipeline' section. It lists several pipeline templates: 'Docker' (Build a Docker image), 'Docker' (Build and push an image to Azure Container Registry), 'Deploy to Azure Kubernetes Service' (Build and push image to Azure Container Registry; Deploy to Azure Kubernetes Service), 'Starter pipeline' (Start with a minimal pipeline that you can customize to build and deploy your code), and 'Existing Azure Pipelines YAML file' (Select an Azure Pipelines YAML file in any branch of the repository). The 'Starter pipeline' template is currently selected. At the bottom, there's a 'Show more' button.

In Azure DevOps, go to Pipelines and click "**New Pipeline**". Choose your repository and select "**Starter pipeline**" as the template. Azure generates a basic YAML file with predefined trigger and jobs. Customize it as needed and click **Save and Run** to execute your pipeline.

```

azuripipelines-1.yml
trigger: none
resources:
  pipelines:
    - pipeline: ci_build
      source: final-usecase-ci
      trigger:
        enabled: true
variables:
  acrName: 'devenvironmentacr38'
  imageName: 'finalusecase'
stages:
  - stage: Deploy
    displayName: Deploy to AKS
    jobs:
      - deployment: DeployToAKS
        displayName: Deploy Application
        environment: 'AKS-Environment'
        pool:
          vmImage: 'ubuntu-latest'
        strategy:
          runOnce:
            deploy:
              steps:
                - checkout: self
                - download: ci_build
                artifact: drop

```

After creating the YAML pipeline in Azure DevOps, it gets saved automatically. The file is named **azure-pipeline-1.yml** and added to your Azure Repos repository. You can find it under the root directory or .azure-pipelines/ folder. It serves as the blueprint for your build and deployment pipeline.

azure-pipeline-1.yml:

```

# Starter pipeline

# Start with a minimal pipeline that you can customize to build and deploy your code.

# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml

trigger: none

resources:

pipelines:
  - pipeline: ci_build
    source: final-usecase-ci

trigger:
  enabled: true

variables:
  acrName: 'devenvironmentacr38'
  imageName: 'finalusecase'

stages:
  - stage: Deploy
    displayName: Deploy to AKS
    jobs:
      - deployment: DeployToAKS
        displayName: Deploy Application
        environment: 'AKS-Environment'

pool:

```

```
vmImage: 'ubuntu-latest'

strategy:
  runOnce:
    deploy:
      steps:
        - checkout: self
        - download: ci_build

      artifact: drop

#Task for health check
- task: PowerShell@2
  displayName: 'Kubernetes Cluster Health Check'

  inputs:
    targetType: 'filePath'
    filePath: 'health-check.ps1'
    pwsh: true

# Task to apply the Kubernetes manifest to your AKS cluster
- task: KubernetesManifest@1
  displayName: Apply Kubernetes Manifest

  inputs:
    kubernetesServiceConnection: 'AKS-Environment-Dev'
    namespace: 'default'
    manifests: |
      $(Pipeline.Workspace)/ci_build/drop/deployment.yaml
    containers: |
      $(acrName).azurecr.io/$(imageName):$(resources.pipeline.ci_build.runID)
```

The screenshot shows the Azure DevOps interface for a pipeline named 'inter-project'. The pipeline has a single job named 'final-usecase-cd' with one step: 'Deploy to AKS'. This step includes sub-steps: 'Initialize job', 'Download', 'Apply Kubernetes Ma...', 'Finalize Job', and 'Report build status'. The 'Apply Kubernetes Ma...' step is highlighted. The log for this step shows the command being run: '/usr/bin/kubectl apply -f /home/vsts/work/_temp/Deployment_hello-world-app_1750740903243,/home/vsts/work/_temp/Service/service/hello-world-service unchanged'. The log concludes with 'deployment "hello-world-app" successfully rolled out'.

The build was success.

2. Kubernetes cluster health check:

The screenshot shows a Microsoft Azure Cloud Shell window. The title bar says 'Microsoft.Azure' and 'Upgrade'. The main area contains a PowerShell session with the following script content:

```
# health-check.ps1
# Kubernetes Cluster Health Check Script

function Check-Nodes {
    Write-Host "Checking Kubernetes Nodes..."
    $nodes = kubectl get nodes --no-headers | ForEach-Object {
        $fields = $_ -split '\s+'
        [PSCustomObject]@{
            NodeName = $fields[0]
            Status   = $fields[1]
        }
    }

    $allHealthy = $true
    foreach ($node in $nodes) {
        if ($node.Status -ne "Ready") {
            Write-Error "Node $($node.NodeName) is not healthy. Status: $($node.Status)"
            $allHealthy = $false
        } else {
            Write-Host "Node $($node.NodeName) is healthy."
        }
    }

    if (-not $allHealthy) {
        exit 1
    }
}
```

To check the health of the cluster by using powershell script called **health-check.ps1**.

health-check.ps1:

```
# health-check.ps1

# Kubernetes Cluster Health Check Script

function Check-Nodes {

    Write-Host "Checking Kubernetes Nodes..."

    $nodes = kubectl get nodes --no-headers | ForEach-Object {

        $fields = $_ -split '\s+'

        [PSCustomObject]@{

            NodeName = $fields[0]

            Status   = $fields[1]
        }
    }

    $allHealthy = $true
    foreach ($node in $nodes) {
        if ($node.Status -ne "Ready") {
            Write-Error "Node $($node.NodeName) is not healthy. Status: $($node.Status)"
            $allHealthy = $false
        } else {
            Write-Host "Node $($node.NodeName) is healthy."
        }
    }

    if (-not $allHealthy) {
        exit 1
    }
}
```

```

}

}

$allHealthy = $true

foreach ($node in $nodes) {

    if ($node.Status -ne "Ready") {

        Write-Error "Node $($node.NodeName) is not healthy. Status: $($node.Status)"

        $allHealthy = $false

    } else {

        Write-Host "Node $($node.NodeName) is healthy."
    }
}

if (-not $allHealthy) {

    exit 1
}
}

```

The screenshot shows the Azure DevOps interface for a pipeline named 'final-usecase-cd'. The left sidebar is visible with various project settings like Overview, Boards, Repos, Pipelines, and Test Plans. The main area displays a successful run of a task named 'Kubernetes Cluster Health Check'. The task details show it was started at 05:02:26.505197 and completed at 05:02:26.617612. The log output is as follows:

```

1 Starting: Kubernetes Cluster Health Check
2 =====
3 Task      : PowerShell
4 Description : Run a PowerShell script on Linux, macOS, or Windows
5 Version   : 2.247.1
6 Author    : Microsoft Corporation
7 Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell
8 =====
9 Generating script.
10 ===== Starting Command Output =====
11 /usr/bin/pwsh -NoLogo -NoProfile -NonInteractive -Command . /home/vsts/work/_temp/e45842ea-81b4-4289-bd04-73e62d701
12 Checking Kubernetes Nodes...
13 E0624 05:02:26.505197 1988 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
14 E0624 05:02:26.506793 1988 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
15 E0624 05:02:26.508588 1988 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
16 E0624 05:02:26.510267 1988 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
17 E0624 05:02:26.513225 1988 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
18 The connection to the server localhost:8080 was refused - did you specify the right host or port?
19 Checking Kubernetes Pods...
20 E0624 05:02:26.617612 1913 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
21 E0624 05:02:26.619322 1913 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
22 E0624 05:02:26.621199 1913 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
23 E0624 05:02:26.622817 1913 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
24 E0624 05:02:26.624796 1913 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Ge
25 The connection to the server localhost:8080 was refused - did you specify the right host or port?
26 All Kubernetes components are healthy.
27
28 Finishing: Kubernetes Cluster Health Check

```

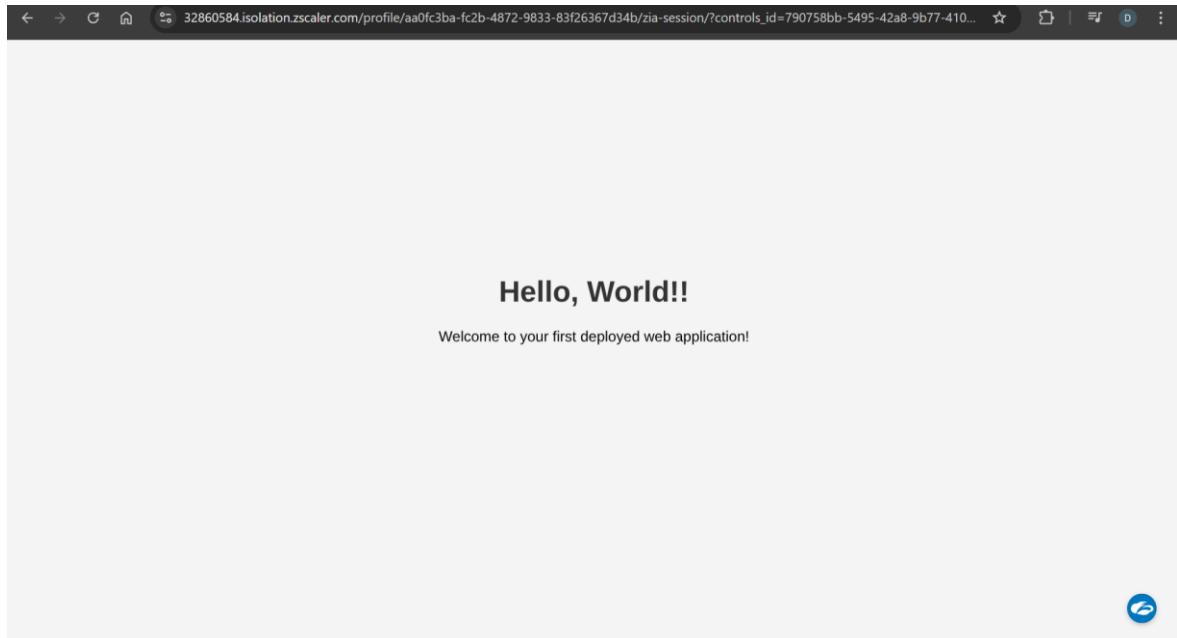
After deploying your app, perform a health check on the AKS cluster using `kubectl get nodes`. Verify all nodes show **Ready** status, and check pods with `kubectl get pods` for running state. Ensure your services are accessible and the app is responding properly via the external IP. Once everything is validated, proceed with the final build or release step in your pipeline. This confirms the Kubernetes environment is healthy and ready for production deployment.

3. final

output:

Name	Namespace	Status	Type	Cluster IP	External IP	Ports
kubernetes	default	Ok	ClusterIP	10.0.0.1		443/TCP
kube-dns	kube-system	Ok	ClusterIP	10.0.0.10		53/UDP,53/TCP
metrics-server	kube-system	Ok	ClusterIP	10.0.18.172		443/TCP
azure-wi-webhook-web...	kube-system	Ok	ClusterIP	10.0.10.45		443/TCP
ama-metrics-ksm	kube-system	Ok	ClusterIP	10.0.137.63		8080/TCP
ama-metrics-operator-t...	kube-system	Ok	ClusterIP	10.0.19.18		80/TCP
network-observability	kube-system	Ok	ClusterIP	10.0.8.67		10093/TCP
hello-world-service	default	Ok	LoadBalancer	10.0.215.143	135.234.148.80	80:30218/TCP

In the Azure Portal, go to your **AKS cluster** and open **Services and Ingress**. Find the service exposing your app and note its **External IP** address. Copy the IP and open it in your web browser (e.g., `http://<external-ip>`). If port mapping is correct (like container port 80), your web app will load. This confirms the app is live and accessible via the external IP.



Conclusion:

Through this hands-on project, you've not only built and deployed a web application using Azure's free-tier services, but also navigated the core pillars of modern DevOps—including infrastructure provisioning, containerization with Docker, orchestration using AKS, and CI/CD automation with Azure Pipelines. These steps reflect real-world workflows used by DevOps engineers in professional environments.

By implementing this end-to-end pipeline, you've gained essential experience in managing cloud-native applications efficiently and securely. More than just a technical exercise, this project lays a solid foundation for scaling your skills toward more advanced cloud architecture and DevOps automation challenges in the future. You're now better equipped to contribute confidently to modern software delivery lifecycles.

New	Active	Resolved	Closed
+ New item			<ul style="list-style-type: none">51 test the web app Closed Dineshkumar Selvan42 health check Closed Dineshkumar Selvan37 create CD pipeline Closed Dineshkumar Selvan44 create Kubernetes Deployment Manifest Closed Dineshkumar Selvan30 create AKS Closed Dineshkumar Selvan

In Azure DevOps, navigate to **Boards** and select your project. Go to the **Work Items** tab to view all user stories and tasks assigned to you. Open each item, update its status to **Closed**, and add relevant comments if needed. Use filters to confirm all assigned work items are completed and closed properly. This ensures your sprint or milestone reflects accurate progress in the board.

TROUBLESHOOTING

1. unable to access the web page through port 80.

Server error - server 20.121.40.196 is unreachable at this moment.
Please retry the request or contact your administrator.

A **504 Gateway Timeout** typically means your browser couldn't get a timely response from the containerized app. First, make sure the container is running using docker ps on your Dev-vm. Confirm the app inside the container is actively listening on **port 80** or the expected port. Check your docker run command includes correct port mapping, like -p 80:80. Ensure that **NSG inbound rules** for the VM allow traffic on port 80. Also verify the VM's public IP is being used correctly in the browser (e.g., http://<public-ip>). If all checks are correct and the issue persists, try restarting the container and VM.

This is a new experience. [Please provide feedback](#)

Network security group Dev-vm-nsg (attached to network)

Impacts 0 subnets, 1 network interfaces

Search rules Source == all Destination

Priority ↑ Name

Inbound port rules (4)

Priority	Name
300	SSH
65000	AllowVnetInBound
65001	AllowAzureLoadBalancerInBound
65500	DenyAllInBound

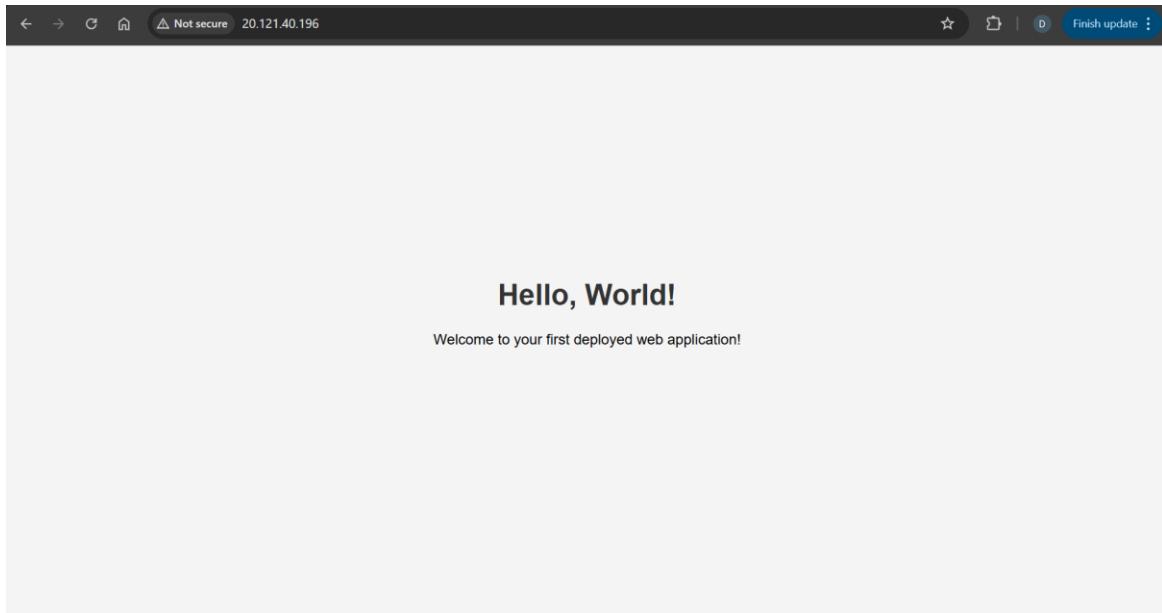
Outbound port rules (3)

Add Cancel Give feedback

By using the internet, I found the cause of this error. The inbound rule I applied to the NSG in Azure was not accepted by the VM's firewall, so when I tried to run the container on port 80, I got the error. I then visited the Azure portal and added the same inbound rule for port 80 that I had previously added to the NSG.

What is 504 Gateway Timeout?

A 504 Gateway Timeout error means a server acting as a gateway or proxy didn't receive a timely response from another server it needed to complete a request. It's a server-side issue, indicating a problem with the server's communication with another server. This often happens when a server is overloaded, down, or experiencing network connectivity issues.



After adding the inbound rule for the VM in VM networking settings.

2. try to use b series size in AKS node.

System Node Pool Requirements: Azure AKS has specific requirements for system node pools, which are responsible for running critical Kubernetes components like CoreDNS, metrics-server, and other essential services. To ensure the reliability and stability of the AKS cluster, these system node pools have minimum resource requirements.

- **Minimum vCPU and Memory:** System node pools generally require a VM SKU with at least **4 vCPUs and 4GB memory**. Many of the smaller B-series VMs might not meet these minimums.
- **Consistent Performance:** The burstable nature of B-series VMs can lead to performance throttling if CPU credits are exhausted. This inconsistency is undesirable for core Kubernetes services, which need predictable and reliable performance to maintain cluster health.

So, I use d series size which is Standard_D2ps_v6 for AKS node.

The screenshot shows the 'Node pools' tab of the 'Create Kubernetes cluster' wizard. A table lists a single node pool named 'agentpool' with the following details:

Name	Mode	Node size	OS SKU	Node count	Availability
agentpool	System	Standard_B2s (cha...)	Ubuntu	2 - 5	1,2,3

A red warning message at the bottom left of the table area states: "B-series node sizes cannot be scheduled. Select another node size." Below the table, there's a section titled "Enable virtual nodes". At the bottom of the screen, there are "Previous" and "Next" buttons, and a "Review + create" button.

3. can't be able to start the minikube:

The screenshot shows the Azure portal interface with a virtual machine named 'Dev-VM'. In the terminal window, the user runs the command 'minikube start'. The output shows that minikube failed to start because it requires at least 2 CPUs, but the current VM only has 1 CPU.

```
Last login: Wed Jun 25 06:04:47 2025 from 4.224.110.241
azure-dev-user@Dev-VM:~$ ls
azure-artifact hello-world-app myagent
azure-dev-user@Dev-VM:~$ minikube start
minikube v1.36.0 on Ubuntu 24.04
Using the docker driver based on existing profile
Exiting due to RSRC_INSUFFICIENT_CORES: has less than 2 CPUs available, but Kubernetes requires at least 2 to be available
azure-dev-user@Dev-VM:~$
```

As per the documentation I use b1s size for the VM. Due to this I can't able to start the minikube.

If the virtual machine is currently running, changing its size will cause it to be restarted. Stopping the virtual machine may reveal additional sizes.

Showing 397 VM sizes. Subscription: Azure subscription 1 Region: East US Current size: Standard_D2s_v3 Learn more about VM sizes Group by series

VM Size ↑↓	Type ↑↓	vCPUs ↑↓	RAM (GiB) ↑↓	Data disks ↑↓	Max IOPS ↑↓
B2s	General purpose	2	4	4	1280
D-Series v3					
D2s_v3	General purpose	2	8	4	3200
E-Series v3					
F-Series v2					
Previous generation sizes					
Size not available					

Prices presented are estimates in USD that include only Azure infrastructure costs and any discounts for the subscription and location. The prices don't include any applicable software costs. Final charges will appear in your local currency in cost analysis and billing views. [View Azure pricing calculator.](#)

[Give feedback](#)

In the Azure Portal, go to **Virtual Machines** and select your VM. Click on **Size** under the **Settings** section in the left menu. Browse available sizes and choose **Standard_D2s_v3** from the list. Click **Resize**, then confirm the operation to change the VM size. After resizing, restart the VM if needed to apply the changes.

```

azimuth@azimuth-OptiPlex-5090:~$ minikube start
(minikube) minikube v1.36.0 on Ubuntu 24.04
(minikube) Using the docker driver based on existing profile
(minikube) Starting "minikube" primary control-plane node in "minikube" cluster
(minikube) Pulling base image v0.0.47 ...
(minikube) Restarting existing docker container for "minikube" ...
(minikube) Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
(minikube) Verifying Kubernetes components...
  * Using image gcr.io/k8s-minikube/storage-provisioner:v5
(minikube) Enabled addons: default-storageclass, storage-provisioner

(minikube) /usr/bin/kubectl is version 1.28.15, which may have incompatibilities with Kubernetes 1.33.1.
  * Want kubectl v1.33.1? Try 'minikube kubectl -- get pods -A'
(minikube) Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
azimuth@azimuth-OptiPlex-5090:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS NAMES
8c2c1511afe4 gcr.io/k8s-minikube/kicbase:v0.0.47 "/usr/local/bin/entr..." 38 hours ago Up 39 seconds 127.0.0.1:32768->22/tcp, 127.0.0.1:32769->2376/tcp, 127.0.0.1:32770->5000/tcp, 127.0.0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp minikube
azimuth@azimuth-OptiPlex-5090:~$ 

```

After resizing the VM to a larger size like **Standard_D2s_v3**, resource constraints are resolved. Log into your VM and run `minikube start` to initialize the Kubernetes cluster. Minikube begins setting up the local cluster using the updated resources. Once completed, check cluster status with `kubectl get nodes` to confirm it's running. Your Minikube environment is now active and ready for local Kubernetes testing.