

# DEPLOYING A DOCKER FILE/CONTAINER ON AZURE

## Problem Statement –

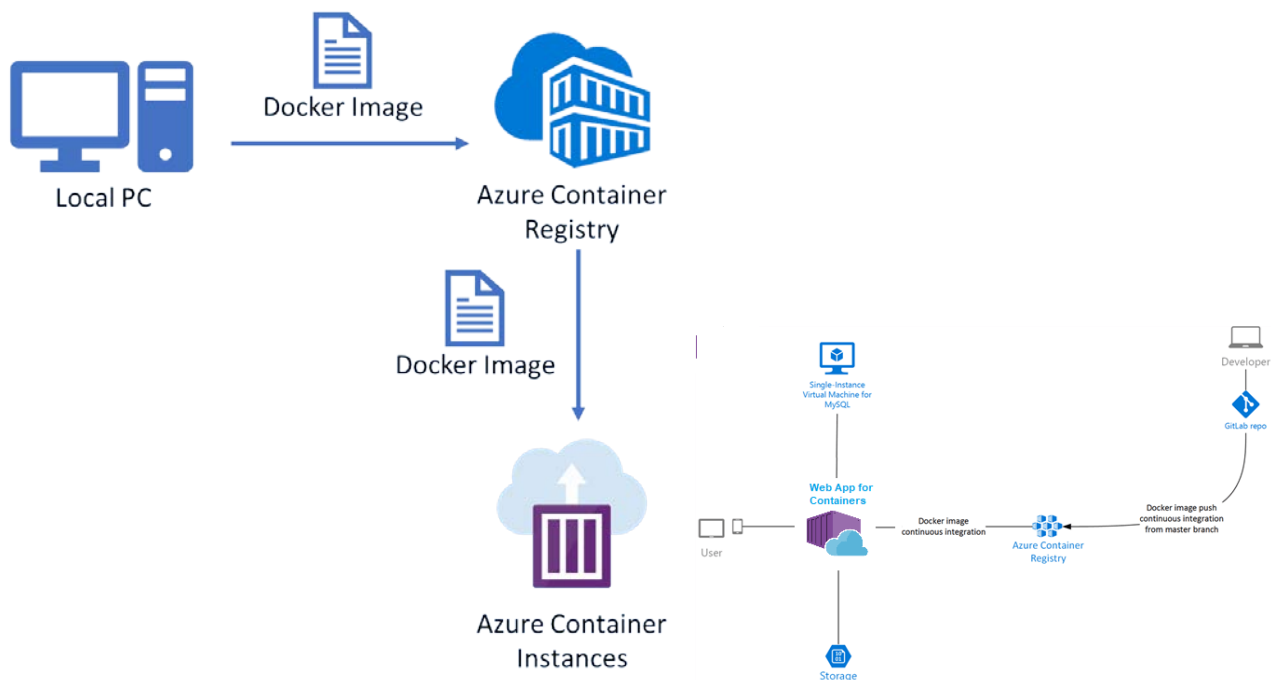
Rather than using VM which are Very slow and hard to manage also not cost efficient

Docker came into picture

Containers require less system resources than traditional or hardware virtual machine environments because they don't include operating system images. Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.

## Introduction-

Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.



Docker images have multiple layers, each one originates from the previous layer but is different from it. The layers speed up Docker builds while increasing reusability and decreasing disk use. Image layers are also read-only files.

In this article we will look into how to create a docker application or image as well as deploy it on Microsoft Azure.

## What is a Container?

A container is basically a unit of software which packages up the actual required code for application with all of its respective dependencies as a standalone image. This image runs very quickly and works on different operating systems and environments. It is lightweight and all the libraries are installed inside the container itself so that there is no need for any other installations apart from Docker itself. It also ensures that it is secure and highly available.

### **Why Use Docker?**

- Fast, consistent delivery of your applications –  
With Docker you can quickly create and deploy several containers within seconds and hence several applications with it.
- Responsive deployment and scaling  
The Docker container-based platform allows containers to be run locally, on cloud or in a mixture of environments easily with scaling.
- Running more workloads on the same hardware is possible with Docker where smaller containers can be run on the same hardware parallelly easily.

### **Benefits of using Azure to integrate with Docker containers:**

1. Support for cross platform like Windows and Linux server containers
2. Higher reliability and availability
3. Simplification of single and multi-container deployment to seamlessly deploy them on Azure's cloud services.
4. Integrated Graphical User Management panel for developers
5. Low-Cost Deployment and pay as you go services.

### **Solution**

#### **Prerequisites and tools–**

1. An active Docker Hub Account
2. Basic Knowledge about Docker
3. Basic Knowledge about Microsoft Azure
4. Installed Visual Studio 2022
5. Microsoft Azure CLI installed
6. VS Code or any Text editor
7. Basic NodeJS or Javascript

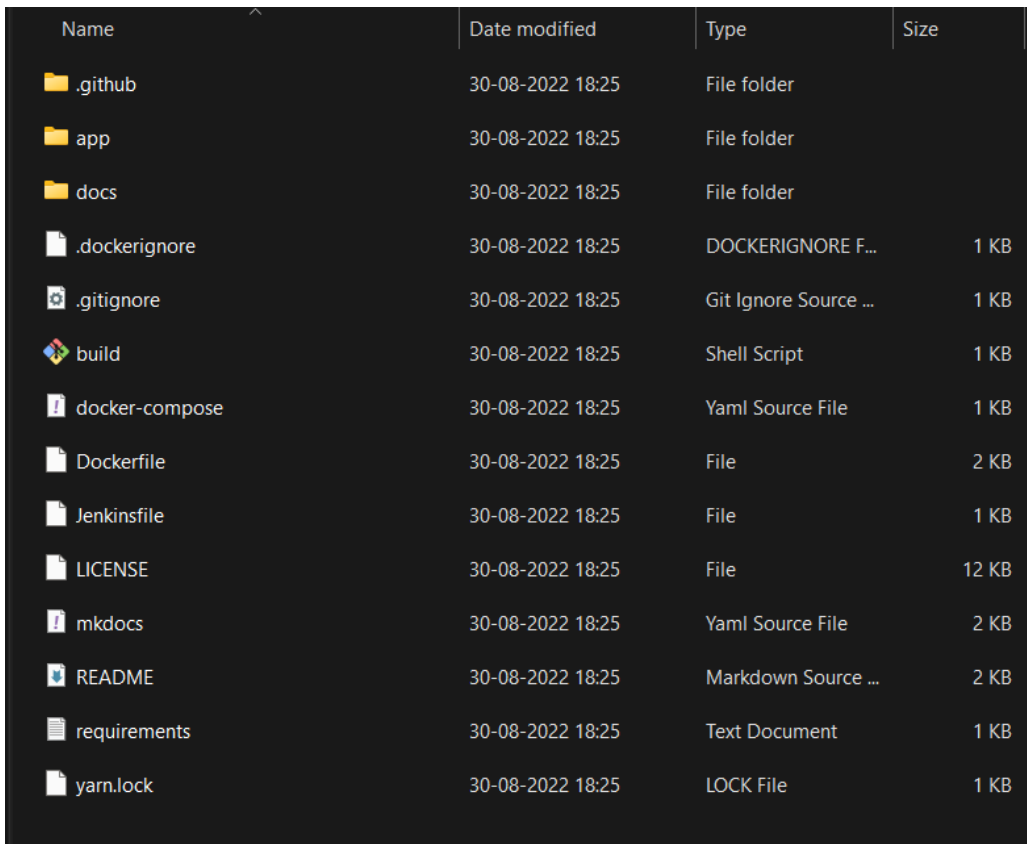
#### **Steps: -**

1. Create a Microsoft Azure account and fill up the basic details or have an active Microsoft Azure account.

2. Download the sample repository or application from the given link or use your own container: -

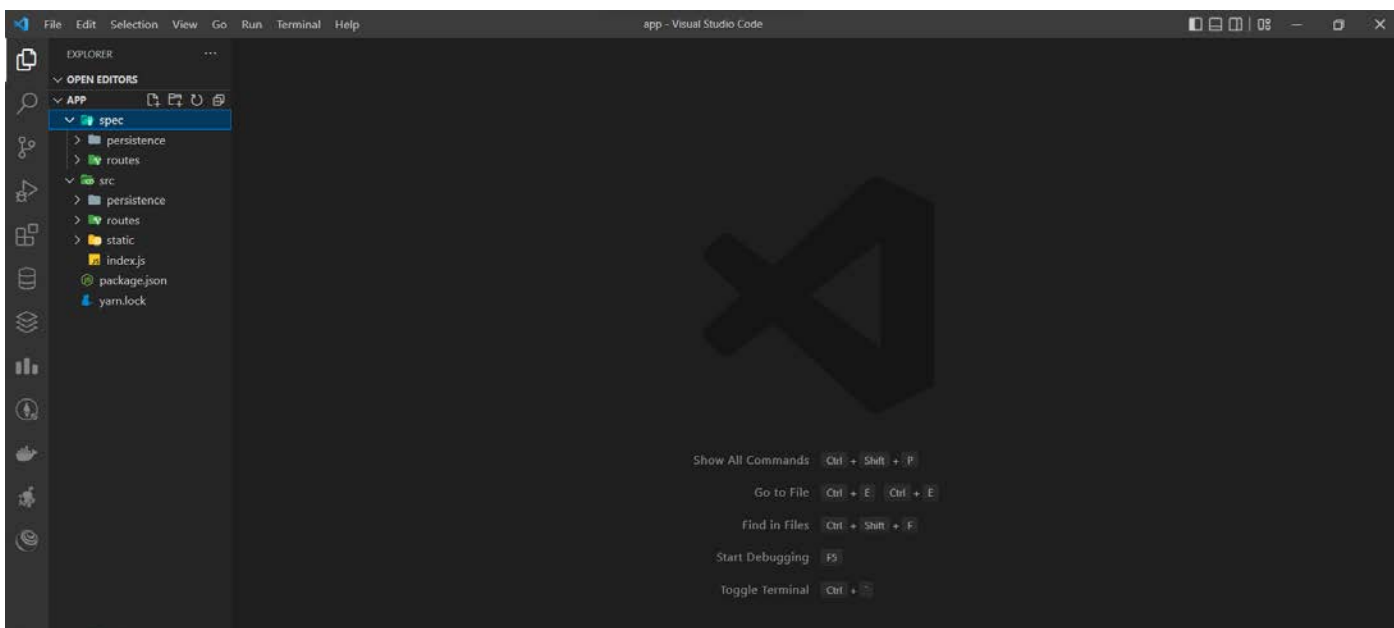
<https://github.com/docker/getting-started/archive/refs/heads/master.zip>

3. Extract the zip file to get started to see the contents as shown –



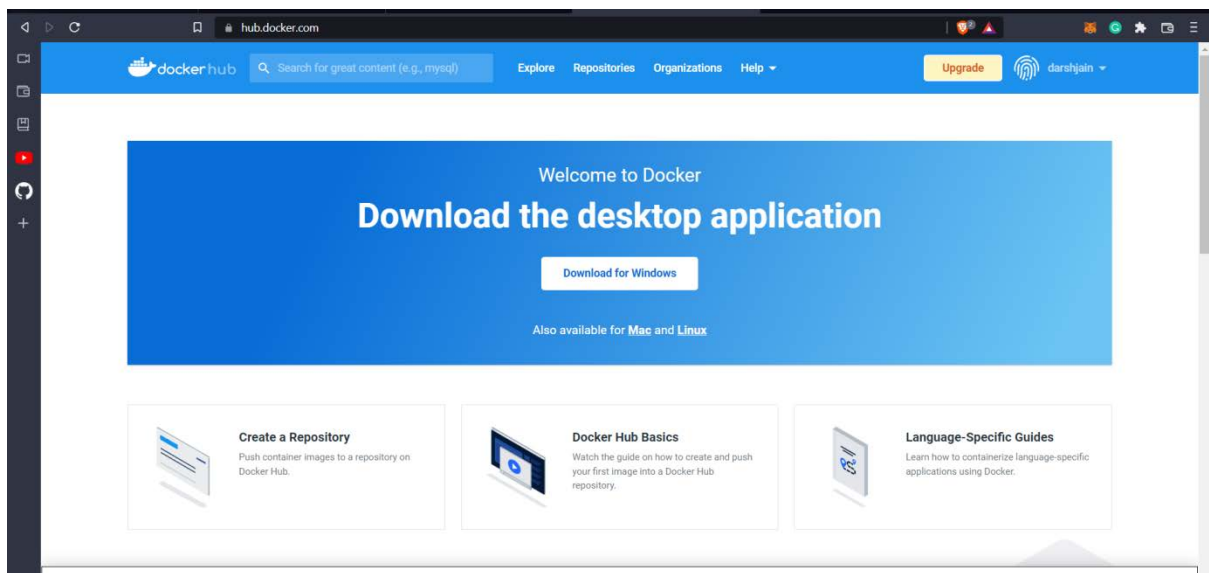
Name	Date modified	Type	Size
.github	30-08-2022 18:25	File folder	
app	30-08-2022 18:25	File folder	
docs	30-08-2022 18:25	File folder	
.dockerignore	30-08-2022 18:25	DOCKERIGNORE F...	1 KB
.gitignore	30-08-2022 18:25	Git Ignore Source ...	1 KB
build	30-08-2022 18:25	Shell Script	1 KB
docker-compose	30-08-2022 18:25	Yaml Source File	1 KB
Dockerfile	30-08-2022 18:25	File	2 KB
Jenkinsfile	30-08-2022 18:25	File	1 KB
LICENSE	30-08-2022 18:25	File	12 KB
mkdocs	30-08-2022 18:25	Yaml Source File	2 KB
README	30-08-2022 18:25	Markdown Source ...	2 KB
requirements	30-08-2022 18:25	Text Document	1 KB
yarn.lock	30-08-2022 18:25	LOCK File	1 KB

4. Open the 'app' folder in your preferred text editor (We will be using Visual Studio Code)

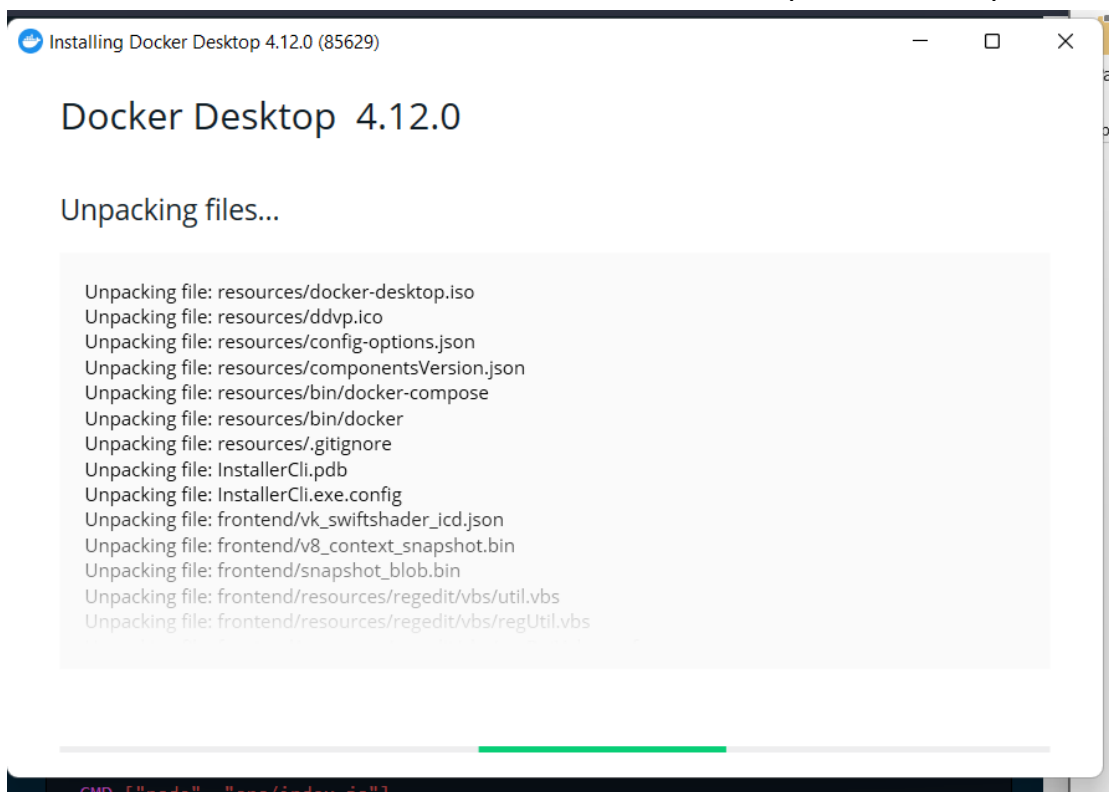


5. Login to the Docker hub page and Download Docker Desktop from the given link: -

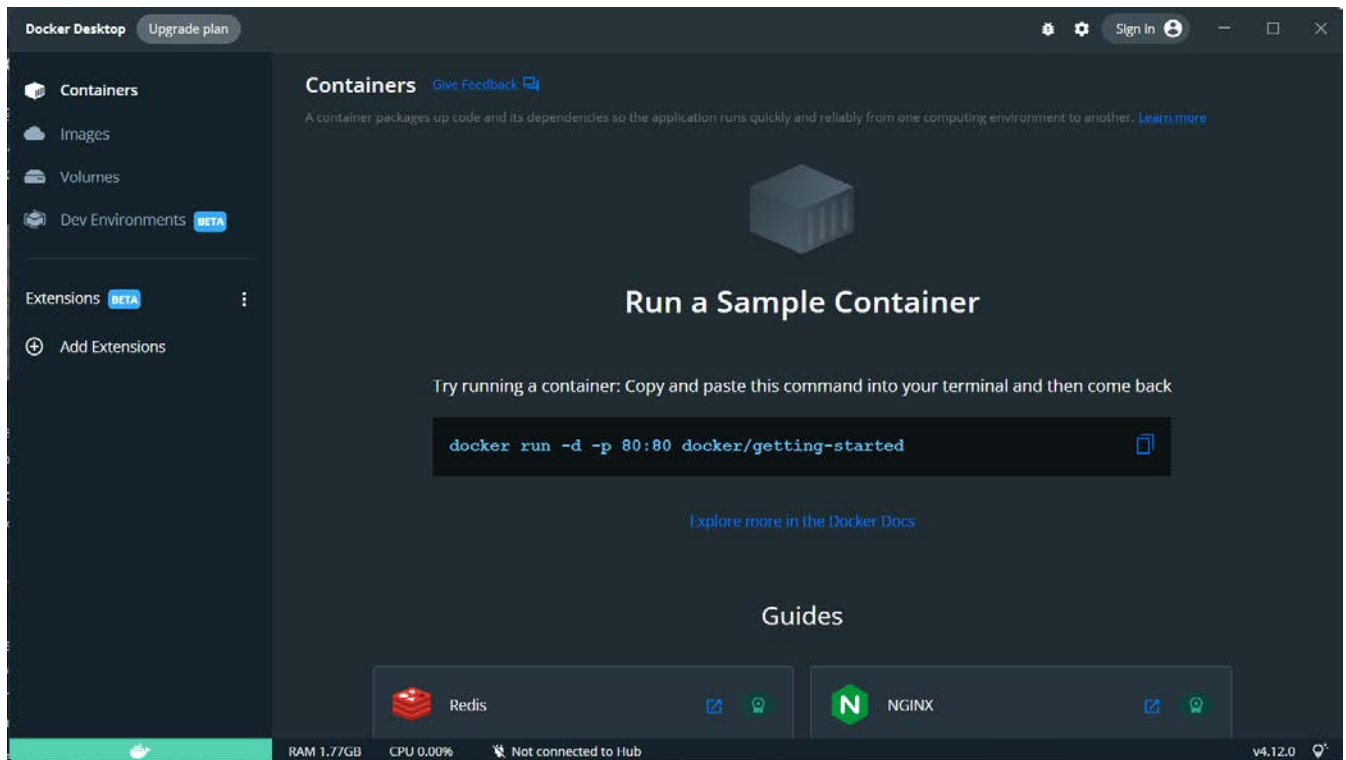
<https://hub.docker.com/>



6. Run the installer for docker and follow the basic steps. Let it complete the installation



7. Start the Docker Desktop service by launching the application. You should receive the following screen upon successful installation and is ready for work.



8. Go back to the text editor (VSCode) and create a "Dockerfile" in the root directory of the application. The Dockerfile must have no extension. A Dockerfile is basically a text-based script of instructions that is used to create a container image. Add the following contents in the same:

```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

9. Open a terminal in the root directory or use the VScode terminal to build the container image using the "docker build" command. This command uses the Dockerfile we created to build a new container image with node v12.  
\$ docker build -t getting-started .

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app> docker build -t getting-started .
[+] Building 0.9s (2/3)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> resolve image config for docker.io/docker/dockerfile:1

```

10. When the docker build completes, it will show the following screen:

```

PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app> docker build -t getting-started .
[+] Building 37.3s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> resolve image config for docker.io/docker/dockerfile:1
=> CACHED docker-image://docker.io/docker/dockerfile:1@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc
=> [internal] load build definition from Dockerfile
=> [internal] load .dockerignore
=> [internal] load metadata for docker.io/library/node:12-alpine
=> [internal] load build context
=> => transferring context: 2.49kB
=> [1/5] FROM docker.io/library/node:12-alpine@sha256:d4b15b3d48f42059a15bd659be60afe21762aae9d6cbeaf124440895c27db68
=> CACHED [2/5] RUN apk add --no-cache python2 g++ make
=> CACHED [3/5] WORKDIR /app
=> CACHED [4/5] COPY . .
=> [5/5] RUN yarn install --production
=> exporting to image
=> => exporting layers
=> => writing image sha256:b2b78b05e34a4c70c45f0db165170ccc4bc5d4caae22fa341be6c017862c902
=> => naming to docker.io/library/getting-started

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app>

```

11. We can start and test the app container we built by running the command  
**\$ docker run -dp 3000:3000 getting-started**

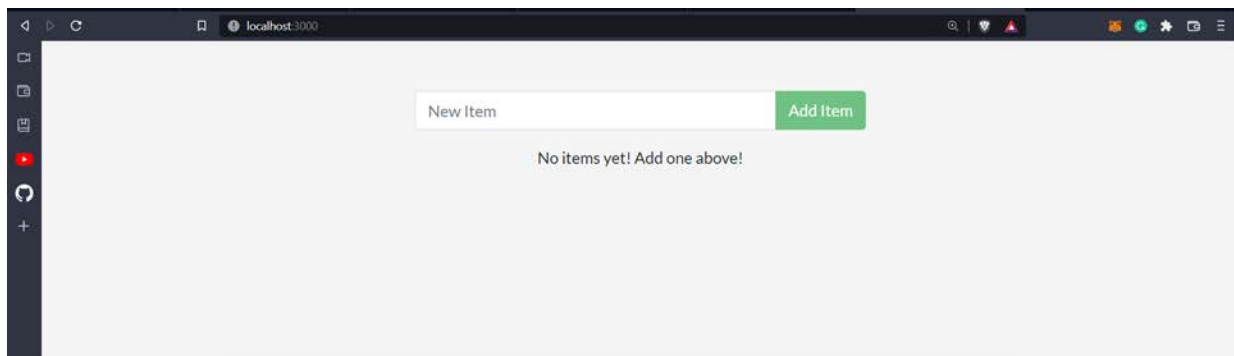
```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app> docker run -dp 3000:3000 getting-started
619991cef9d8399133a25d0ab3192c9a4e93006da7ef5f6fcf93e72c701a9b8a
PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app>

```

The string we got as output represents the ID of the container and means that the container is successfully up and running.

12. You can see and test the app container running on <http://localhost:3000>.



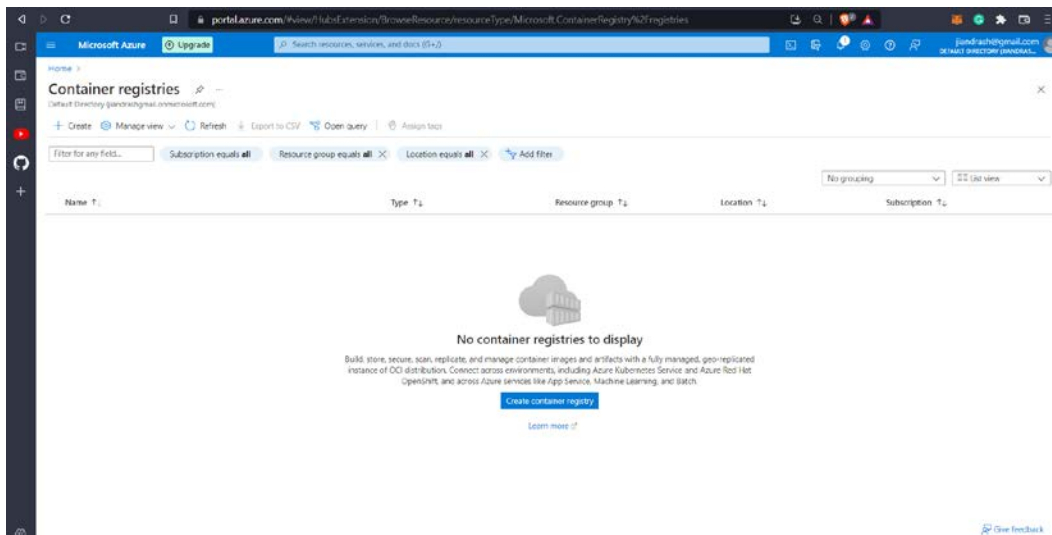
We have successfully built and tested the container now let's move onto deployment to Microsoft Azure

13. Make sure you have installed Azure Cli or download from the official website

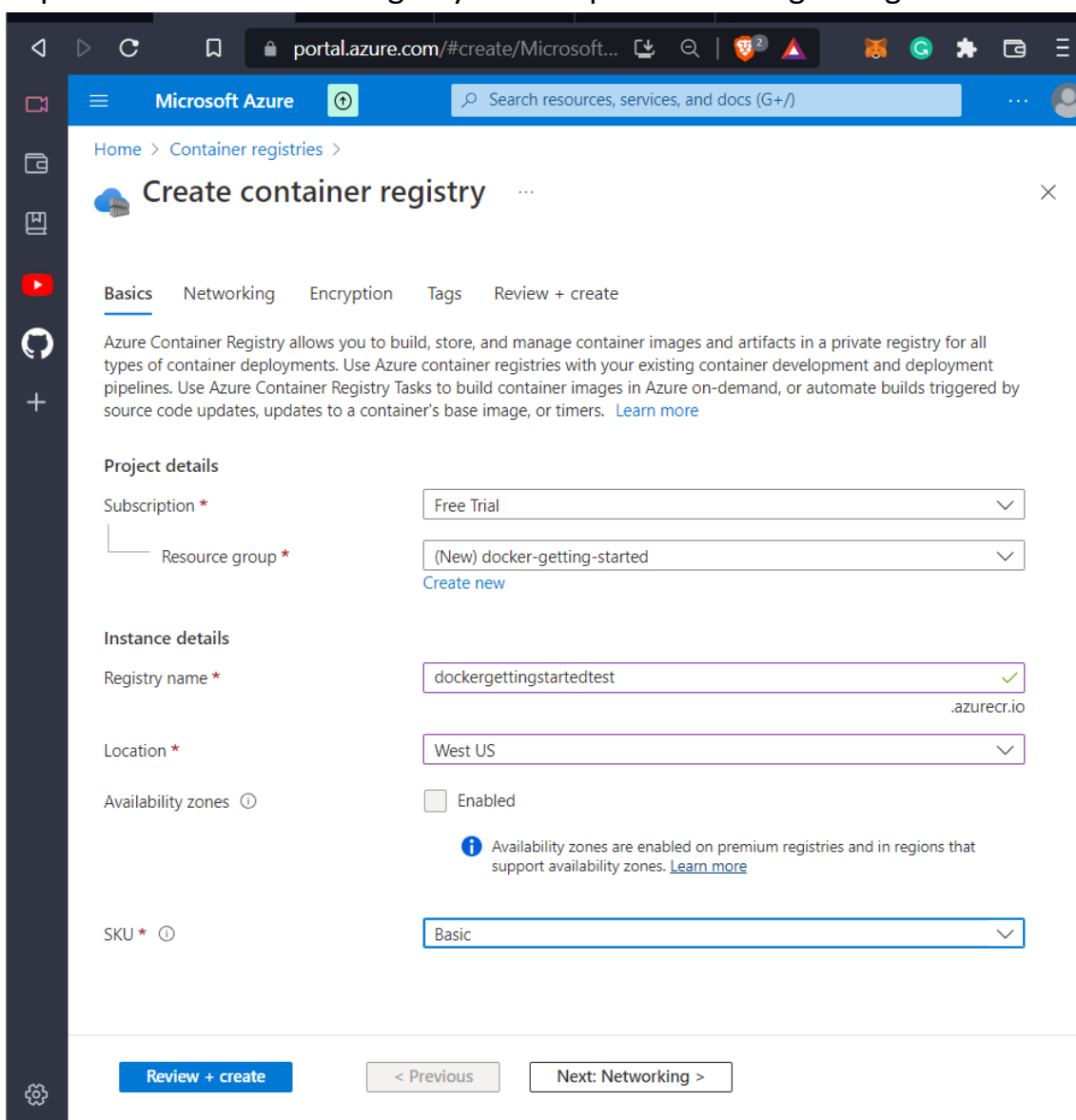
14. First, we will have to create a container registry on Microsoft Azure:

- a. Select Create a resource > Containers > Container Registry.

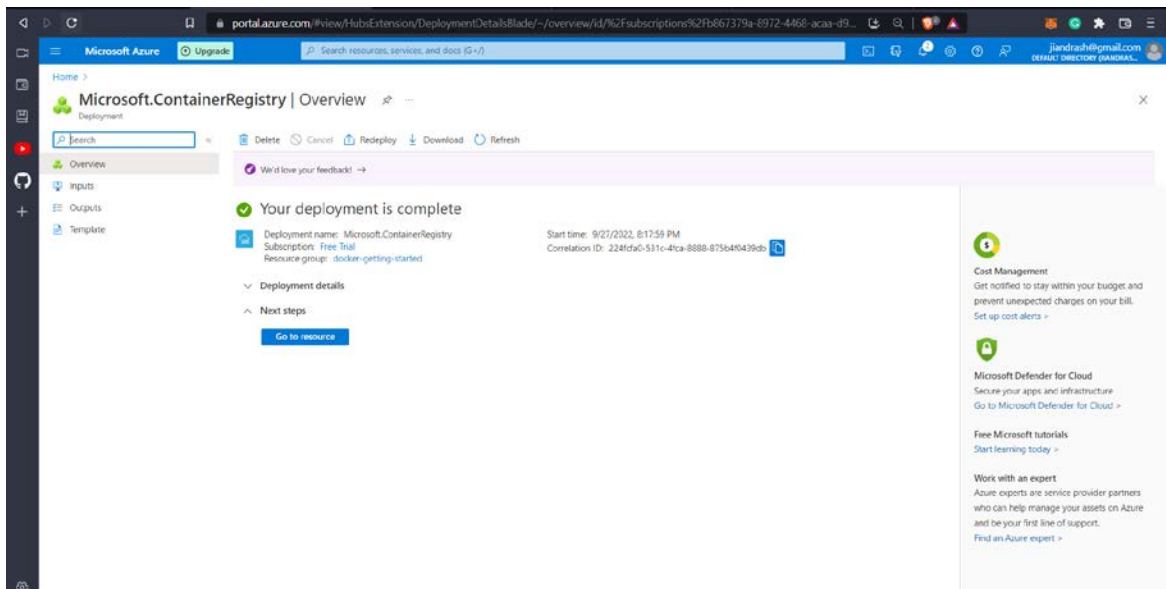
b. Navigate to “Container Registries on Azure Website”



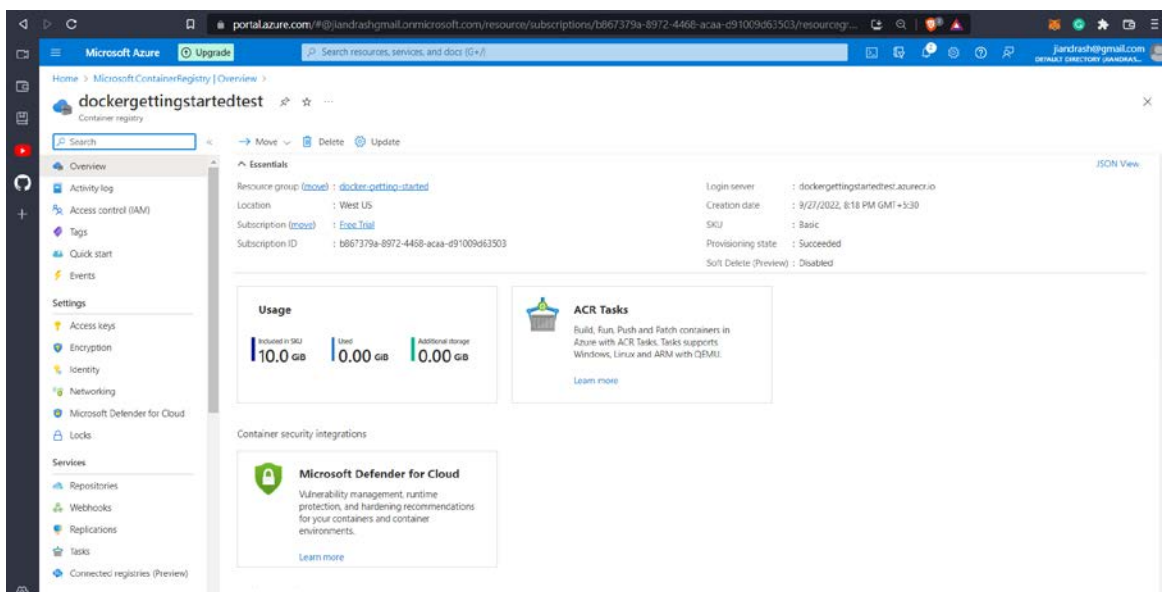
c. Tap on create container registry and keep the following configuration



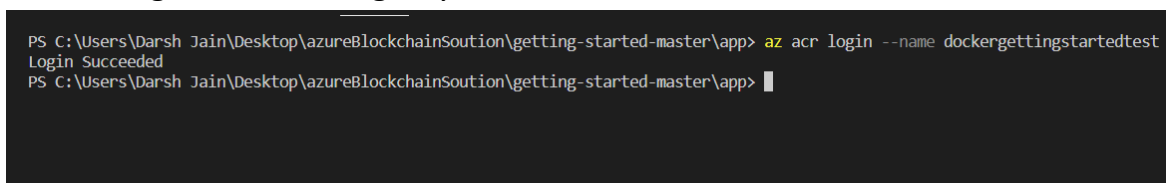
d. Click on review and create > create and you should receive a deployment successful message on screen with container registry showing up



e. Go to the resource to see the details as shown



f. Return to the terminal and run the following command  
`$ az acr login --name <registry-name>`



This sets the container registry on docker up on local machine or app directory so that we can push the container.

g. The setup is completed as you can see

15. Let's push our container image and run the container. Let's push the image to the registry. Use the following command. Replace <login-server> with the name available on the container details webpage

`$ docker push <login-server>/getting-started:latest`



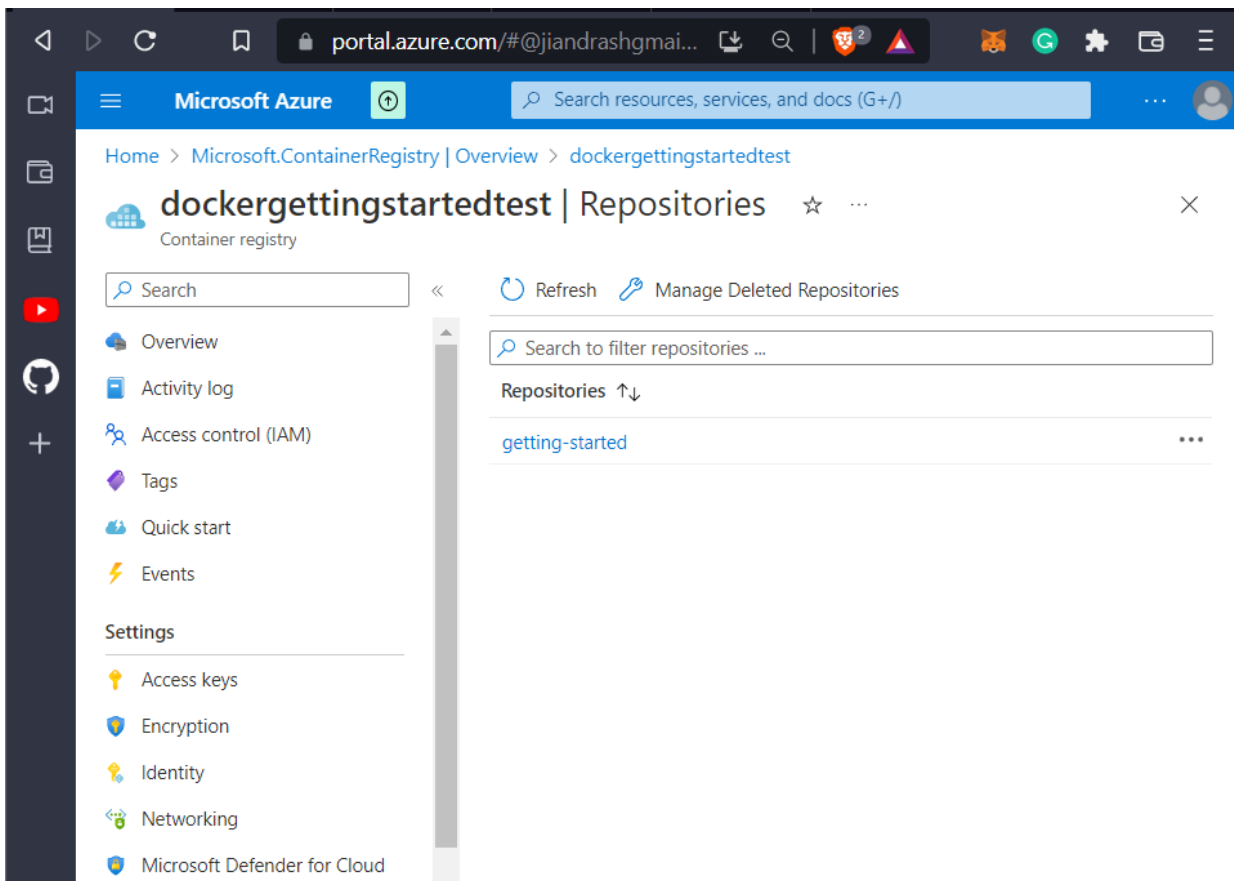
```
PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app> docker push dockergettingstartedtest.azurecr.io/getting-started:v1
The push refers to repository [dockergettingstartedtest.azurecr.io/getting-started]
a2f7a4682ddc: Pushing [=====>] 10.67MB/86MB
36a3d3814490: Pushed
ff2d7fe00be3: Pushed
11694d90c925: Pushing [=>] 7.133MB/222.6MB
7f30ce3f699: Pushed
fe810f5902cc: Pushing [=====>] 5.762MB/7.838MB
dfd8c046c602: Pushing [=====>] 7.75MB/77.6MB
4fc242d58285: Pushing [=====>] 1.04MB/5.575MB
[]
```

It might take a few minutes to push the image to the Azure Container.

16. Once completed it should show the following screen

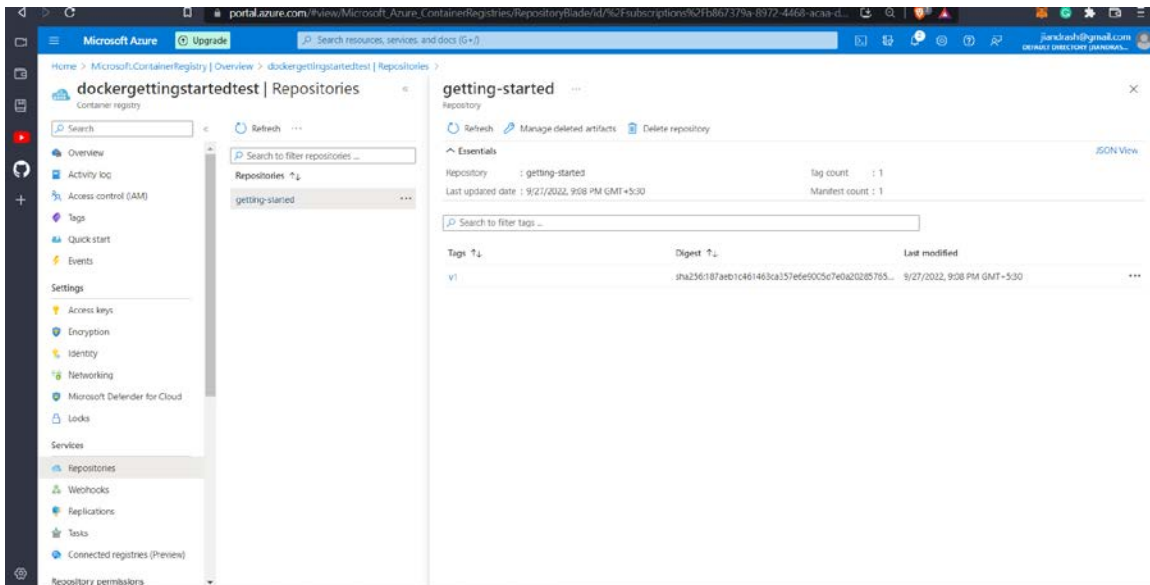
```
PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app> docker push dockergettingstartedtest.azurecr.io/getting-started:v1
The push refers to repository [dockergettingstartedtest.azurecr.io/getting-started]
a2f7a4682ddc: Pushed
36a3d3814490: Pushed
ff2d7fe00be3: Pushed
11694d90c925: Pushed
7f30ce3f699: Pushed
fe810f5902cc: Pushed
dfd8c046c602: Pushed
4fc242d58285: Pushed
v1: digest: sha256:187aeb1c461463ca357e6e9005d7e0a202857651cbb8cd5d84e262239f016821 size: 2000
PS C:\Users\Darsh Jain\Desktop\azureBlockchainSoution\getting-started-master\app>
```

17. Check the Docker Webpage to see the container you created. Navigate to repositories from the tab on the left.

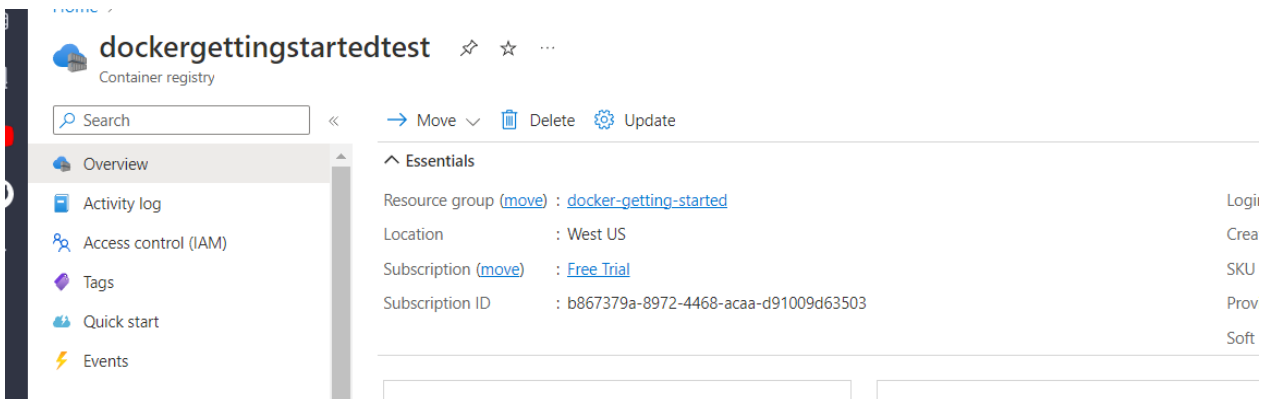


You can see the container we pushed in the list

18. Click the container to see the details of the container like version no, digest etc.

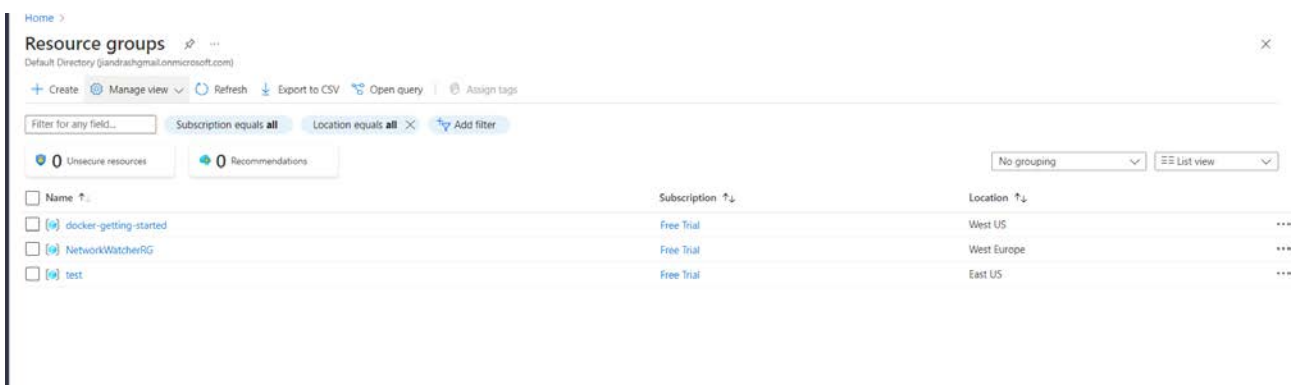


19. Now that we have deployed the container , suppose we want to clean up the resources and services i.e., the container itself. Then In that case we can simply delete the resource group itself.



Simply click on delete to delete the container

Navigate to resource group and delete the resource group that we created to unallocated all the resources and services associated with it.



### Use Cases from Business Point of View-

An organisation or business usually produce several containers especially in IT industry, managing them manually is a difficult task. So Azure and docker can help IT industry by deploying them on cloud so that it's easier to manage their web applications and servers.

An organisation providing cybersecurity services or a company looking to secure its system can use the containers on Azure to test various other software. Using containers on Azure they can conduct tests using different configurations simultaneously such as DOS or DDOS attacks. Making 100s of containers is very easy on docker so such attacks are very easy to perform using Azure and docker. Also, Azure provides different server locations and demographics and hence conducting a more varied set of security tests.

### **Challenges Faced :**

containers are however directed to performance overhead due to overlay networking, interfacing within containers and the host system and so on.

The one major issue is if an application designed to run in a Docker container on Windows IaaS VM, then it can't run on Linux VM or vice versa.

Docker creates new security challenges like the difficulty of monitoring multiple moving pieces within a large-scale, dynamic Docker environment. IaaS VM security is an overhead.

### **Business Benefits:**

Fast application deployment

Transferable across machines – an application and all its dependencies can be grouped into a separate container that is autonomous from the host version of Linux kernel, platform configuration, or deployment type.

we can pursue succeeding versions of a container, inspect irregularities, or go back to previous versions.

we can use a distant Azure container repository to share our container with others, and it is also desirable to configure our own individual repository.