Deploying ASP.NET Core MVC Application to Kubernetes

1. Framework Selection

I selected ASP.NET Core MVC for this task because:

- It represents a fundamentally different architecture from NodeJS (C# vs JavaScript, integrated middleware pipeline vs Express middleware)
- Microsoft's official documentation provides comprehensive deployment guides
- The framework includes built-in support for containerization
- It enforces strict MVC separation (Controllers/Views/Models folders structure)

Unlike NodeJS which uses package.json for dependencies, ASP.NET Core uses NuGet packages and csproj files. The routing system is also more structured, with explicit controller-action mapping rather than NodeJS's callback-based routes.

2. Learning Process

Learning steps:

1. Followed Microsoft's "Get started with ASP.NET Core MVC" tutorial
2. Created basic BookController with Index action
3. Added simple Book model class
4. Scaffolded basic CRUD views

Key challenges:

- Understanding Razor view syntax (transitioning from Handlebars/EJS)
- Configuring Dockerfile for .NET Core (different from NodeJS multi-stage builds)
- Runtime differences between development and production modes

Solutions:

- Used dotnet new mvc template as foundation
- Referenced Microsoft's containerization guidelines
- Enabled detailed error pages during development

3. Deployment Process

Deployment steps:

1. Created optimized Dockerfile using SDK/Runtime stages
2. Built and tested container locally
3. Set up local registry on Kubernetes cluster
4. Created Kubernetes manifests with:
    - Deployment (2 replicas)
    - NodePort service
5. Verified operation through:
    - kubectl get endpoints
    - Load testing with kubectl port-forward
    - Log inspection

Critical fixes:

- Adjusted containerPort to match ASP.NET's default 8080
- Added proper liveness probes
- Configured proper environment variables for production

4. Reflection

Key learnings:

- Containerizing .NET apps requires understanding of runtime vs SDK images
- Kubernetes deployments need proper readiness checks for MVC apps
- ASP.NET's configuration system differs significantly from NodeJS

Challenges:

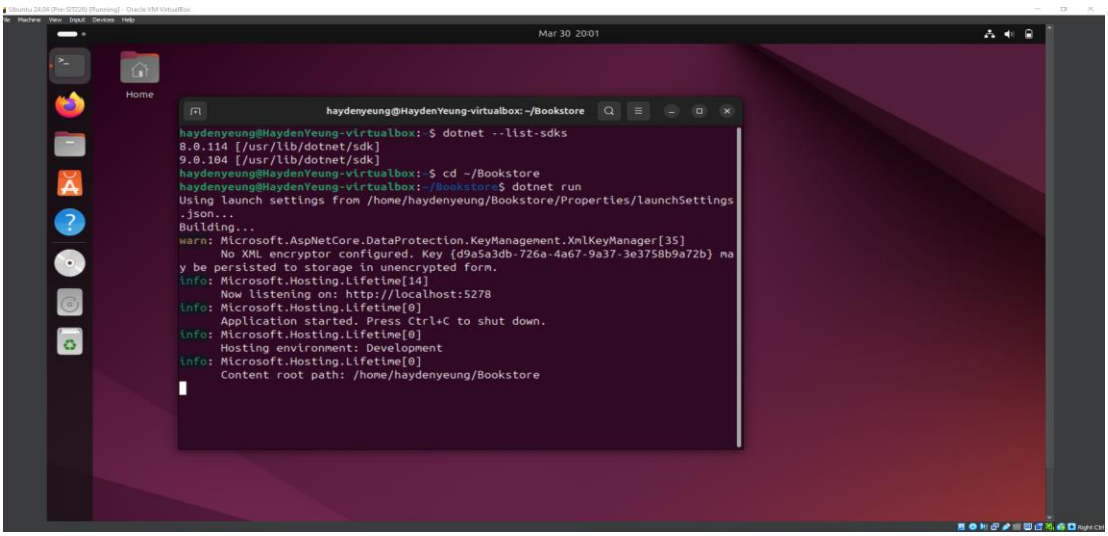
- Debugging containerized applications requires different approaches
- Kubernetes networking for local development needs careful planning

For future work:

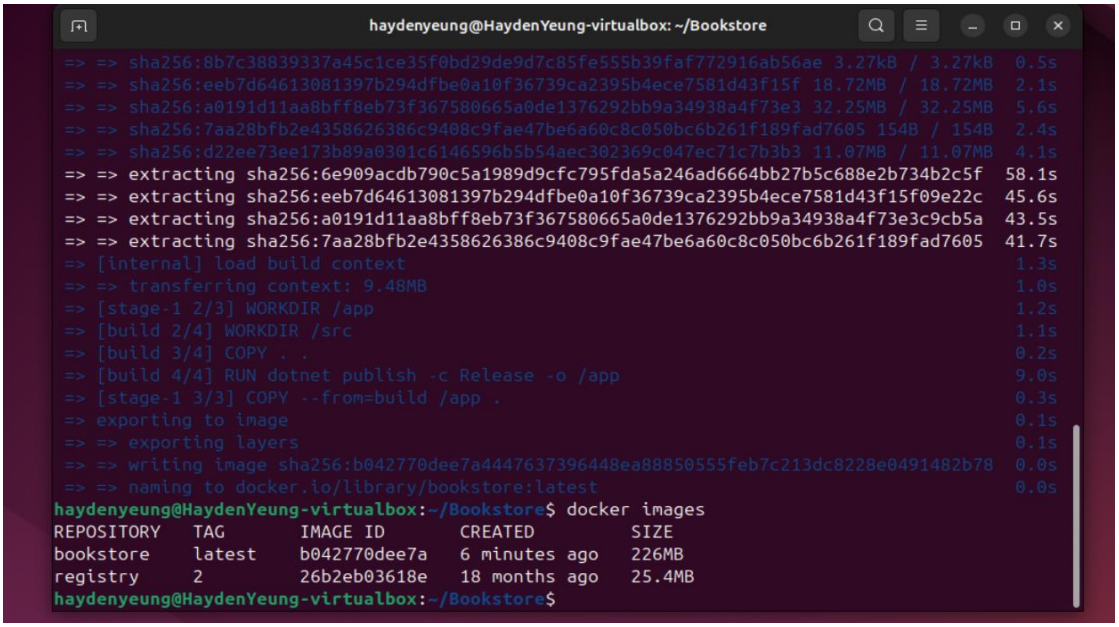
- Implement proper health checks earlier

- Use ConfigMaps for environment variables
- Consider ingress controllers from the start

Appendices

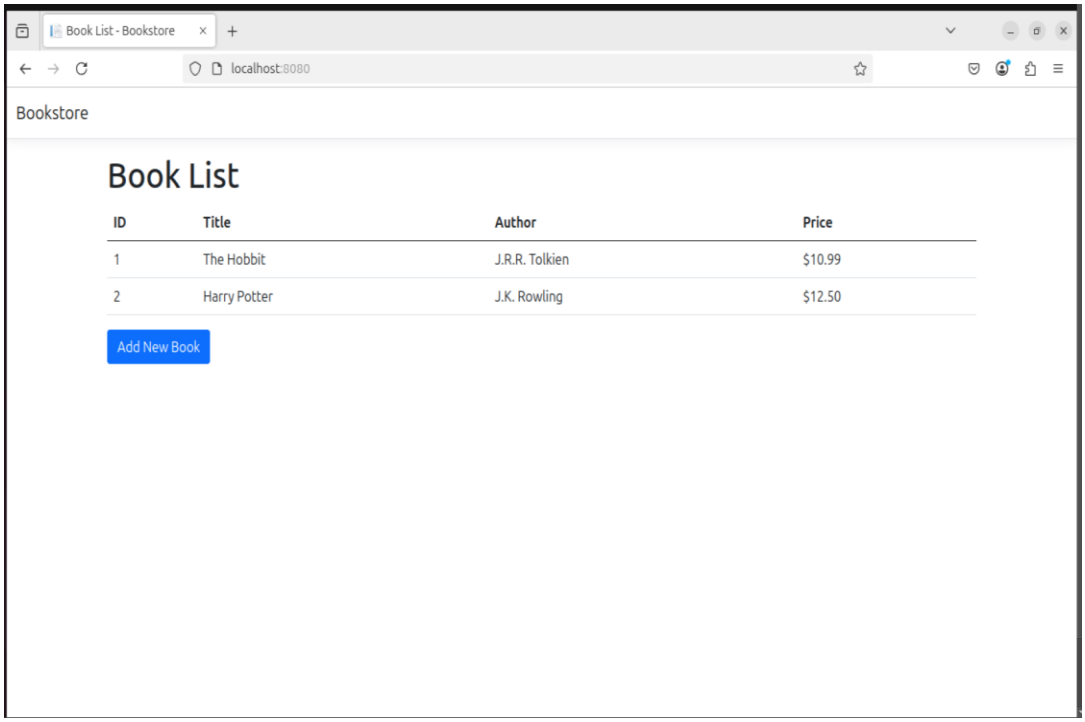


Website is now able to run on Ubuntu via “dotnet run” command.



Successfully create a Docker Image through Dockerfile

```
haydenyeung@HaydenYeung-virtualbox:~/Bookstore$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
bookstore    latest    b042770dee7a   6 minutes ago   226MB
registry     2         26b2eb03618e   18 months ago   25.4MB
haydenyeung@HaydenYeung-virtualbox:~/Bookstore$ docker run -p 8080:80 bookstore:latest
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted
 outside of the container. Protected data will be unavailable when container is destroyed. For
more information go to https://aka.ms/aspnet/dataprotectionwarning
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {69e5be5f-7ab2-4a94-bd83-17cdb345e333} may be persisted
to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:8080
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```



The basic Bookstore website is now accessible through “http://localhost:8080”

```
haydenyeung@HaydenYeung-virtualbox:~/Bookstore/Kubernetes$ cd ~/
haydenyeung@HaydenYeung-virtualbox:~$ cd Bookstore
haydenyeung@HaydenYeung-virtualbox:~/Bookstore$ kubectl apply -f Kubernetes/bookstore-deployment-servic
e.yaml
deployment.apps/bookstore-deployment created
service/bookstore-service unchanged
haydenyeung@HaydenYeung-virtualbox:~/Bookstore$ kubectl get pods
NAME                                  READY   STATUS                   RESTARTS   AGE
bookstore-deployment-5666f9c8d8-dpmmw   1/1     Running                  0          16s
bookstore-deployment-5666f9c8d8-m7nx8   1/1     Running                  0          16s
my-website-5c4d4449b-4lzcn            0/1     Error                    0          121m
my-website-5c4d4449b-59hfv            1/1     Running                  0          114m
my-website-5c4d4449b-64w4l            0/1     ContainerStatusUnknown   1          121m
my-website-5c4d4449b-69z66            1/1     Running                  0          114m
```

```
haydenyeung@HaydenYeung-virtualbox:~/Bookstore$ kubectl get svc
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
bookstore-service   NodePort    10.152.183.112   <none>        80:30771/TCP   5m45s
kubernetes          ClusterIP   10.152.183.1     <none>        443/TCP        30d
my-website          NodePort    10.152.183.218   <none>        80:32671/TCP   4d2h
```

Bookstore Web Application got deployed with Kubernetes

```
haydenyeung@HaydenYeung-virtualbox:~$ kubectl get endpoints bookstore-service
NAME                ENDPOINTS                                AGE
bookstore-service   10.1.186.15:8080,10.1.186.61:8080   29m
```

Checking with command “kubectl get endpoints bookstore-service”

GitHub Link:

https://github.com/HaydenDuong/SIT226_Cloud_Automation_Technologies/tree/main/Coding%20Tasks/2.3D%20-%20Bookstore%20Web%20App