Prepare Application and create application image

Step 1.

Create a application.

Open spring tool suit-4 (location Desktop)

Go to file and choose new -> new spring starter project

A screenshot of a computer

Description automatically generated

Define project Name

Choose project type  
  
Go for next and choose required dependencies for the project

Suh as sql. Spring web, jpa etc..

A screenshot of a computer

Description automatically generated

This will download project folder

A screenshot of a computer

Description automatically generated

Note the project folder will be downloaded under the directory which spring is using as project directory.

Now we will be able to see the project

A computer screen shot of a program

Description automatically generated

Run the application  
  
A screenshot of a computer

Description automatically generated

If you encounter any error on database. That is because of we selected sql driver during creation. So the SQl driver dependencies are installed but, we have not configured DB connection.

To resolve this problem. Go to src/main/resources -> application.properties

Configure DB connection query.

server.port=8089

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/testdatabase?userSSL=false

spring.datasource.username=root

spring.datasource.password=partha123

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

Configure rest controller on application

Add “@RestController” annotation at the class

* package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@RestController

@SpringBootApplication

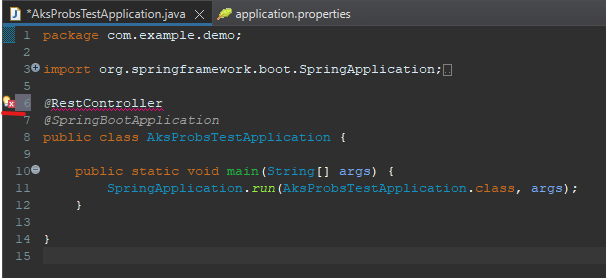
public class AksProbsTestApplication {

public static void main(String[] args) {

SpringApplication.run(AksProbsTestApplication.class, args);

}

}

It will show as error  
  


Click on error option and import (press enter)  
  
A screenshot of a computer

Description automatically generated

Add controller to map request to different functions

@GetMapping(“/map-name”)

Here we can do get mapping directly on “public static void main(String[] args) {“

But it will create a problem. That is this class should void type only as it’s main class.

So, create a new class inside main class and define controller mapping.

Ex-

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@SpringBootApplication

public class AksProbsTestApplication {

public static void main(String[] args) {

SpringApplication.run(AksProbsTestApplication.class, args);

}

@GetMapping("/index")

public static String Index() {

return "index page. Ststus 'OK'. addition application startup prob";

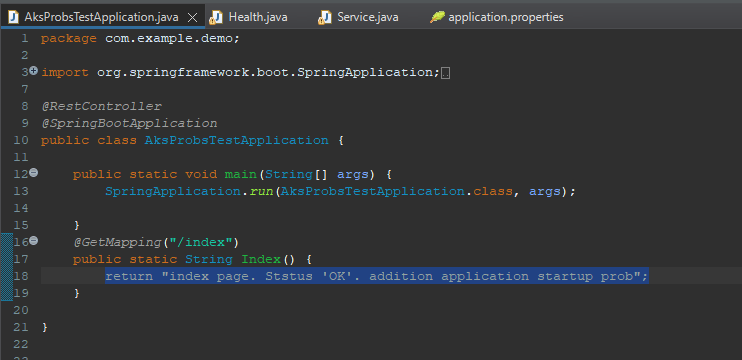
}

}

Add some return value to check on page.

return "index page. Ststus 'OK'. addition application startup prob";

Additionally, as this is our starting page of application. so, we will define this page map as application starting. (startup prob).



Create other 2 class

How to create class?

* Go to controller -> new -> class

Provide inputs to create class

A screenshot of a computer

Description automatically generated

Health.java (class)

Additionally we will configure this page map (/health-check). As readiness prob.

* package com.example.demo;
* import org.springframework.boot.SpringApplication;
* import org.springframework.web.bind.annotation.GetMapping;
* import org.springframework.web.bind.annotation.RestController;
* *@RestController*
* public class Health {
* *@GetMapping*("/health-check")
* public static String Health() {
* return "Health Check page. health-check Ststus 'OK'. additional readiness prob.";
* }
* }

Service.java

This will be considered as liveness prob

package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

*@RestController*

public class Service {

*@GetMapping*("/Service-Health")

public static String Service() {

return "Service Health page. Service-Health Ststus 'OK' \n additional liveness prob.";

}

}

A screen shot of a computer

Description automatically generated

Run the application and check working functionality.

A screenshot of a computer

Description automatically generated

Check the application. Heck the application from postman and browser.

Note- check the application is running on which port

It will be defined on application.properties page in project.

A screenshot of a computer program

Description automatically generated

Now hit the loalhost request to each page and check

Note- you must define correct map to get response back.

Ex-

Index page ([localhost:8089/index](http://localhost:8089/index))

A screenshot of a computer

Description automatically generated

Service health page ([localhost:8089/Service-Health](http://localhost:8089/Service-Health))

A screenshot of a computer

Description automatically generated

Health check page ([localhost:8089/health-check](http://localhost:8089/health-check))

A screenshot of a computer

Description automatically generated

Check by Postman

A screenshot of a computer

Description automatically generated

Step 2.

Build the docker image for the application

Open project with vs ode for more flexibility to create docker file

Create dockerfile in project directory

Doker file code-

FROM openjdk:8-jdk-alpine

ARG JAR\_FILE=build/libs/\*.jar

COPY ${JAR\_FILE} app.jar

RUN mkdir destination-dir-for-add

ADD sample.tar.gz /destination-dir-for-add

If the above code did not work try below docker file code. Which is specifically for gradel.

# Use the official OpenJDK 17 image

FROM openjdk:17-jdk-alpine

# Set the working directory in the container

WORKDIR /app

# Copy the Gradle wrapper and build files

COPY gradlew .

COPY gradle gradle

COPY build.gradle .

COPY settings.gradle .

# Copy the source code

COPY src src

# Grant execution rights to the Gradle wrapper

RUN chmod +x gradlew

# Build the application

RUN ./gradlew build

# Copy the built jar file to the container

ARG JAR\_FILE=build/libs/\*.jar

COPY ${JAR\_FILE} app.jar

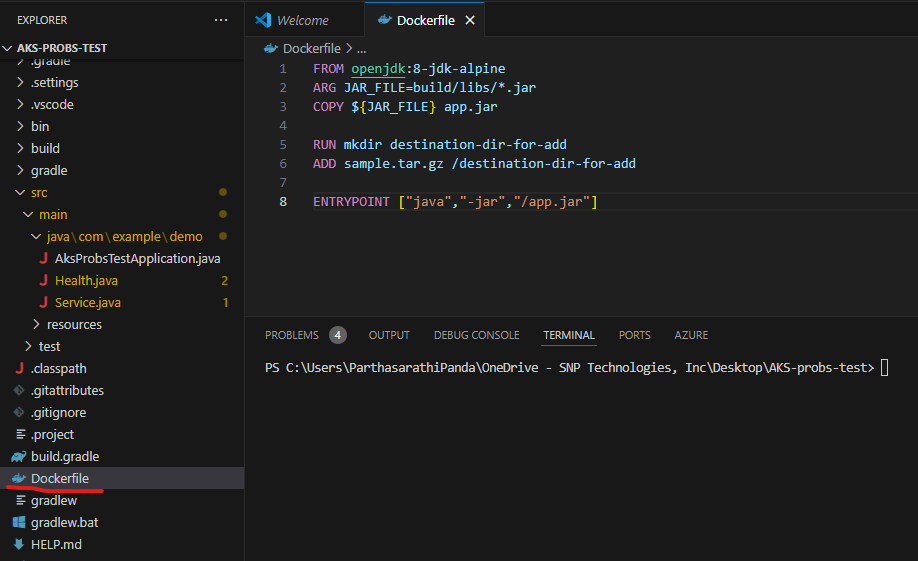
# Expose the port the application runs on

EXPOSE 8080

# Run the application

ENTRYPOINT ["java", "-jar", "/app.jar"]

ENTRYPOINT ["java","-jar","/app.jar"]

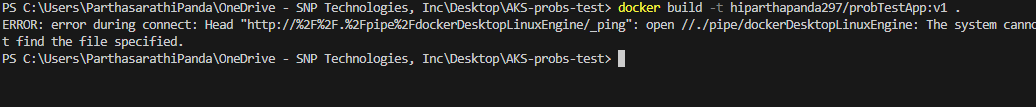


Build the docker image-

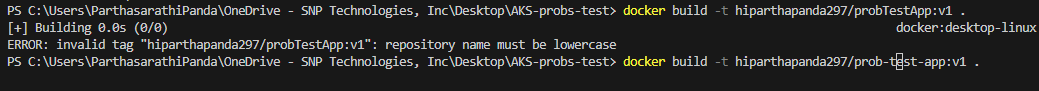
Go to project directory terminal

>> docker build -t <image\_name>:<tag> <path\_to\_dockerfile>

Ex- docker build -t hiparthapanda297/probTestApp:v1 .



Note- make sure docker desktop is running.



Change the repository name as per policy and run the command again

ERROR---------

The error I was getting because the project was in running state in backend (spring boot)

Stop running the project and re run build cmd.

Note- if you get failed at the time of building the project. Run “./gradlew clean” in project directory to clean any old build foder. And then try to build manually to test “./gradlew build”

A screen shot of a computer

Description automatically generated

After successfully build try to build docker image.

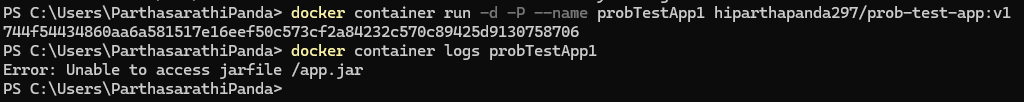
>> docker build -t hiparthapanda297/prob-test-app:v1 .

A screenshot of a computer program

Description automatically generated

Now run the application locally and check everything working fine

>> docker container run -d -P --name probTestApp1 hiparthapanda297/prob-test-app:v1



Let’s Upload the image to docker hub as version 1 and we will troubleshoot with next version.

>> docker push imagename

Ex- docker push hiparthapanda297/prob-test-app:v1

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now run the cleanup and remove created container

>> docker container ls -a

>> docker container rm container-name or can provide container id

A black screen with white text

Description automatically generated

Now we had created image by using alternative dockerfile content

A screenshot of a computer program

Description automatically generated

Cleanup previous build folder

>> ./gradlew clean

Replace dockerfile content with

FROM openjdk:8-jdk-alpine

ARG JAR\_FILE=build/libs/\*.jar

COPY ${JAR\_FILE} app.jar

RUN mkdir destination-dir-for-add

ADD sample.tar.gz /destination-dir-for-add

A screen shot of a computer

Description automatically generated

Save the file and run docker build

>> docker build -t hiparthapanda297/prob-test-app:v2 .

This did not work as the issue with sample.tar.gz we guess that was the old formate.

New formate is app.jar

Replace the dockerfile and rerun docker build

A screen shot of a computer program

Description automatically generated

Error- => ERROR [10/10] COPY build/libs/\*.jar app.jar 0.1s

------

> [10/10] COPY build/libs/\*.jar app.jar:

A black screen with white text

Description automatically generated

But manually build the project and then create docker image is worked.

By troubleshooting the issue that It looks like the COPY build/libs/\*.jar app.jar step is failing because the build/libs directory does not exist or the jar file is not being found.

As the manual build worked so I did manual build and checked files under build -> libs

1. AKS-probs-test-0.0.1-SNAPSHOT-plain.jar
2. AKS-probs-test-0.0.1-SNAPSHOT.jar

A screenshot of a computer

Description automatically generated

Now modify the docker file and run docker file again.

Don’t forget to cleanup build folder before moving to docker build.

Doker file-

# Use the official OpenJDK 17 image

FROM openjdk:17-jdk-alpine

# Set the working directory in the container

WORKDIR /app

# Copy the Gradle wrapper and build files

COPY gradlew .

COPY gradle gradle

COPY build.gradle .

COPY settings.gradle .

# Copy the source code

COPY src src

# Grant execution rights to the Gradle wrapper

RUN chmod +x gradlew

# Build the application

RUN ./gradlew build

# Copy the built jar file to the container

COPY build/libs/AKS-probs-test-0.0.1-SNAPSHOT.jar app.jar

# Expose the port the application runs on

EXPOSE 8080

# Run the application

ENTRYPOINT ["java", "-jar", "/app.jar"]

This too did not work.

Issue is with docker file.

Use docker file

# Use a specific base image with OpenJDK 17 (adoptopenjdk is now Eclipse Temurin)

FROM eclipse-temurin:17-jdk-alpine AS build

# Set the working directory in the container

WORKDIR /app

# Copy the Gradle wrapper and Gradle files

COPY gradlew .

COPY gradle gradle

# Copy the project source code

COPY src src

COPY build.gradle .

# Make the Gradle wrapper executable

RUN chmod +x gradlew

# Run Gradle build to generate the JAR file (assuming it is configured to build with Gradle)

RUN ./gradlew build --no-daemon

# Create the final image from a smaller base image with just JRE (runtime only)

FROM eclipse-temurin:17-jre-alpine

# Set the working directory in the final container

WORKDIR /app

# Copy the JAR file from the build stage (located under build/libs/ after Gradle build)

COPY --from=build /app/build/libs/\*.jar app.jar

# Expose the port your Spring Boot app will run on (default 8080)

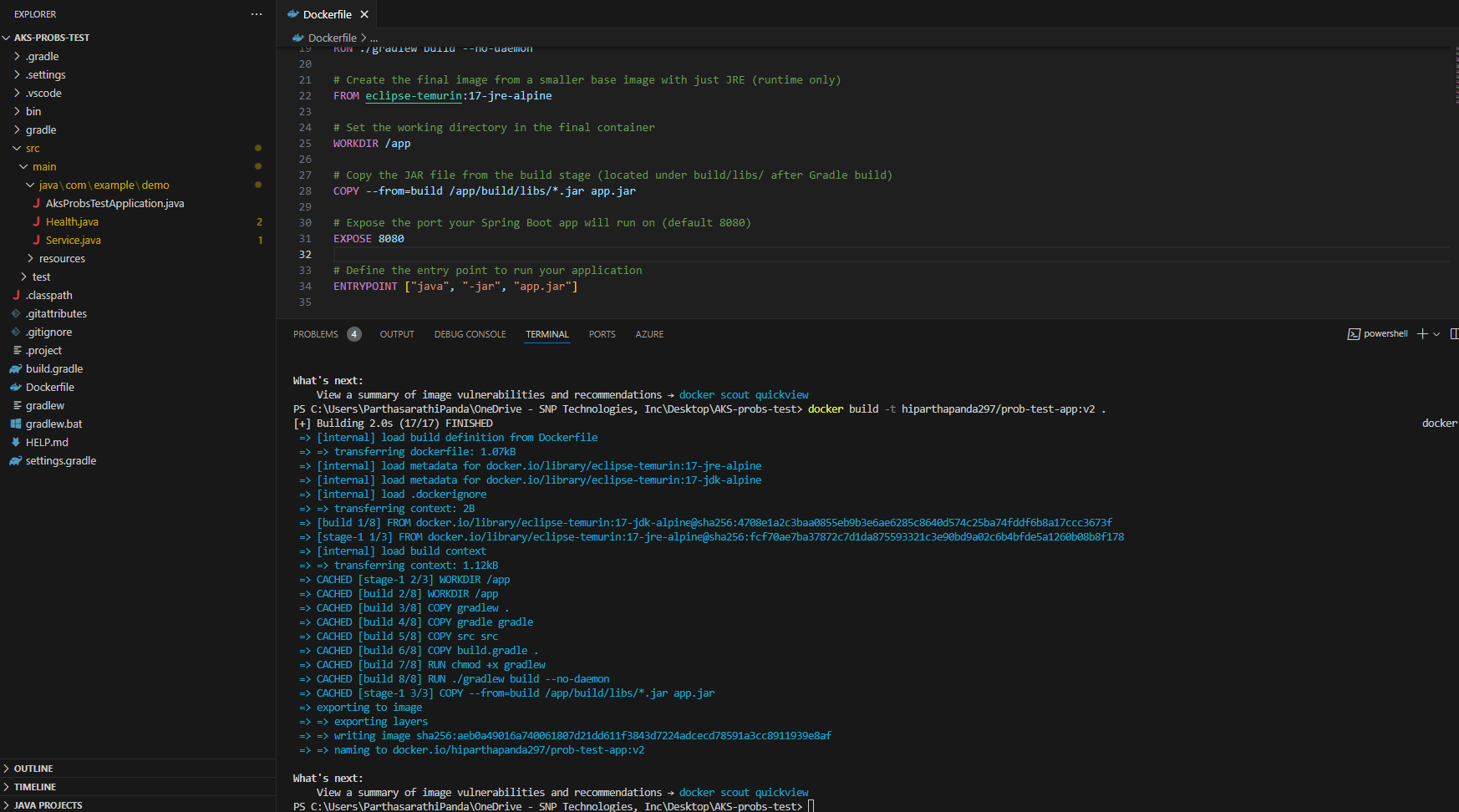
EXPOSE 8080

# Define the entry point to run your application

ENTRYPOINT ["java", "-jar", "app.jar"]

**Explanation:**

1. **Base Image (JDK)**: We are using eclipse-temurin:17-jdk-alpine as the base image for building the project. This is a lightweight OpenJDK 17 image.
2. **Working Directory Setup**: The working directory is set to /app inside the container. The project files (including Gradle wrapper files) are copied into this directory.
3. **Build the Application**: The Gradle wrapper is used (./gradlew build), which compiles the application and generates the JAR file.
4. **Runtime Image (JRE)**: After the build, a much smaller JRE-only image (eclipse-temurin:17-jre-alpine) is used to create the final Docker image for running the Spring Boot application.
5. **Copy the JAR File**: The generated JAR file from the build/libs/ directory is copied into the final image.
6. **Expose the Port**: The default Spring Boot port 8080 is exposed.
7. **Entry Point**: The ENTRYPOINT runs the Spring Boot application using java -jar.



Build was successful. Image got created and created container on that. But sending request to application is not responding.

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

Push this version as well to docker hub

A screenshot of a computer

Description automatically generated

Check container logs

>> docker container logs container id

docker container logs 160598a12cb1

A black screen with white text

Description automatically generated

I guess the database link given is local DB (sql), so it’s unable to connect to that.

Now here the pod is running but it’s failed to start the application inside. This image will be best to use for start up prob test and also liveness and readiness test.

Yes so the issue is-

**Fix Database Connection**

It looks like your Spring Boot application is trying to connect to a MySQL instance running on localhost, but in a Docker container, localhost refers to the container itself, not your host machine or any other external services.

If your MySQL server is running on your **host machine**, you need to ensure that the container can access the MySQL instance. There are several ways to do this:

**Option A: Use Host Networking (Simpler Approach)**

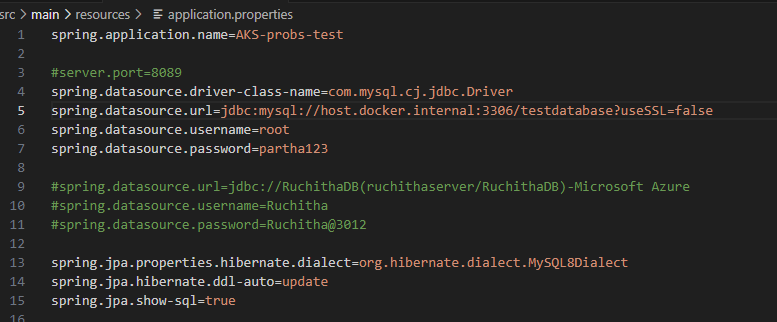
If you're using Docker locally and MySQL is running on your host machine, you can connect to the host machine's network from within the container using host.docker.internal (on Docker Desktop for Windows/macOS):

Update your application.properties:

properties

>> spring.datasource.url=jdbc:mysql://host.docker.internal:3306/testdatabase?useSSL=false

Let’s try it on application.



Also the post on container and application it’s trying to reach to port 8080, so let’s modify that as well.

>> server.port=8080  
  


A computer screen shot of a program

Description automatically generated

Let’s build the v3 image version and test.

>> docker build -t hiparthapanda297/prob-test-app:v3 .

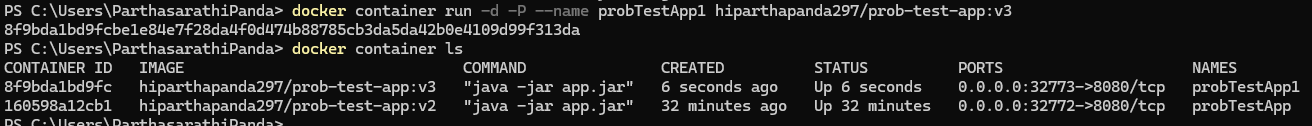
Verify image

>> docker image ls

A screenshot of a computer

Description automatically generated

Create new container and test.

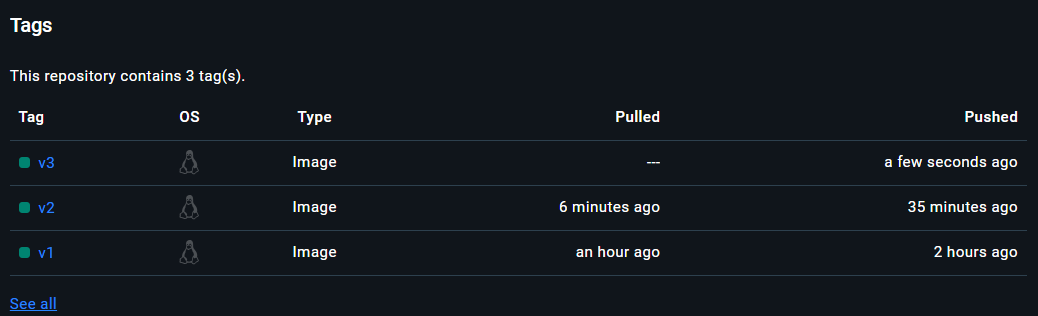


Chek accessibility

A screenshot of a computer

Description automatically generated

So this is the final working version upload this to docker hub.



Step 3.

Deploy the application to AKS.

Create AKS step we will get option to assig labels and taints on nodes to schedule pods on targeted node and node pool.

Labels - Labels are key/value pairs that can be used to categorize or add identifying information to Kubernetes resources such as nodes. Labels for the node pool will be applied to each node in the node pool.

Taints - Taints are tuples that are used in conjunction with tolerations to determine which pods can be scheduled on which nodes. In order for a pod to be scheduled to a node, it must tolerate all of the taints applied to that node. Taints for the node pool will be applied to each node in the node pool.

A screenshot of a computer

Description automatically generated

Till the time AKS is getting created get ready with deployment.yaml file.

To deploy the resources on aks. We are deploying at deployment level.

Mention liveness readiness and startup probs in deployment.

A startup probe **verifies whether the application within a container is started**. Startup probes run before any other probe, and, unless it finishes successfully, disables other probes. If a container fails its startup probe, then the container is killed and follows the pod’s restartPolicy.

liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs.

The kubelet uses readiness probes to know when a container is ready to start accepting traffic. One use of this signal is to control which Pods are used as backends for Services. A Pod is considered ready when its Ready [condition](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-conditions) is true. When a Pod is not ready, it is removed from Service load balancers. A Pod's Ready condition is false when its Node's Ready condition is not true, when one of the Pod's readinessGates is false, or when at least one of its containers is not ready.

by default, the Application Gateway Ingress Controller (AGIC) provisions an HTTP GET probe for exposed Azure Kubernetes Service (AKS) pods. You can customize the probe properties by adding a [readiness or liveness probe](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/) to your deployment or pod specification.

Deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: prob-test-app

labels:

app: prob-test-app

spec:

replicas: 3

selector:

matchLabels:

app: prob-test-app

template:

metadata:

labels:

app: prob-test-app

spec:

containers:

- name: prob-test-app

image: hiparthapanda297/prob-test-app:v3

ports:

- containerPort: 8080

readinessProbe:

httpGet:

path: /health-check

port: 8080

initialDelaySeconds: 15

periodSeconds: 10

livenessProbe:

httpGet:

path: /Service-Health

port: 8080

initialDelaySeconds: 15

periodSeconds: 10

startupProbe:

httpGet:

path: /index

port: 8080

failureThreshold: 30

periodSeconds: 10

env:

- name: PORT

value: "8080"

Service.yaml

apiVersion: v1

kind: Service

metadata:

  name: prob-test-app-service

spec:

  selector:

    app.kubernetes.io/name: prob-test-app

  ports:

    - protocol: TCP

      port: 80

      targetPort: 8080

Also can do both yaml in same deployment file to deploy at once.  
  
A screen shot of a computer

Description automatically generated

Connect to AKS

We are using workstation (laptop) to deploy to cluster and also will use to monitor application.

A screen shot of a computer

Description automatically generated

Do deployment





Pod deployment

A black background with white text

Description automatically generated

Error (pod logs)

A screen shot of a computer

Description automatically generated

Pod contains error as the aks application is not able to connect to our on prem sql running DB.

Describe the deployment, deployment probs liveness readiness and startup probs are failed as the application is not running, not able to handle requests and also application is failed to start.

A screenshot of a computer

Description automatically generated

This application is good for running on local by docker container. By on aks have connection issue.

This we can use as prob test. Specially for startup prob because due to error application is failed to start.

Make the application ready to start for aks container.

To do this we can do in 3 different ways.

1. Configure to connect your aks application to on prem DB
2. Create SQL Container on aks by using docker image. This is not recommended because the pod may have any issue and the db may restart and loose data.
3. Create SQL flexi server and configure to connect to application.

<https://youtu.be/LGnl38V9pb0?si=gKf2cgMMTRPblGGq>

As of now our application don’t required and database so we are disabling database for the application.

Make changes in application property. “spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration”.

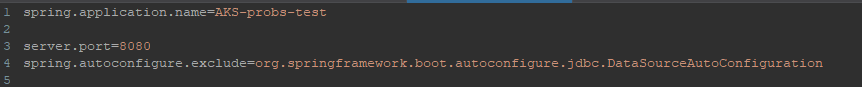
The above data line will avoid default to connect to DB so it will not throw db error.

>>

spring.application.name=AKS-probs-test

server.port=8080

spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration



Now build the application . image version 4.

>> docker build -t hiparthapanda297/prob-test-app:v4 .

Note- make sure application is in stopped stoped state before you run docker build.

Also make sure Docker Desktop is running.

A screen shot of a computer

Description automatically generated

Push the image to Docker hub

A screenshot of a computer

Description automatically generated

Deploy new image to aks.



Check pods status and application log

Note- we missed to change deployment name so the deployment has over ride. If you want to keep both change the deployment name in yaml file.

Application is running and working as expected.

A screen shot of a computer screen

Description automatically generated

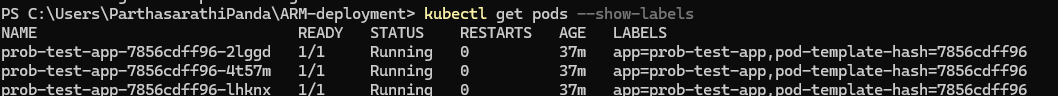
Verify the service and access application

>> kubectl port-forward pod/<pod-name> <local-port>:<remote-port> -n <namespace>

EX- kubectl port-forward --address 0.0.0.0 service/prob-test-app-service 808:80

To check labels applied on pods

>> kubectl get pods --show-labels



Describe service

>> kubectl describe svc prob-test-app-service

A screen shot of a computer

Description automatically generated  
  
in our case port forword was getting fail. This was due to lavels applied on application is “app: prob-test-app”. But, on service it was “selector: app.kubernetes.io/name: prob-test-app”

Changed selector on service

#Service yaml  
  
apiVersion: v1

kind: Service

metadata:

  name: prob-test-app-service

spec:

  selector:

    app: prob-test-app

  ports:

    - protocol: TCP

      port: 80

      targetPort: 8080

A screenshot of a computer

Description automatically generated

Now do port forwording

>> kubectl port-forward svc/prob-test-app-service 8080:80



Now browse the application . we will be able to access it locally not externally.

A screenshot of a computer

Description automatically generated

Note – the service does not have any external ip.

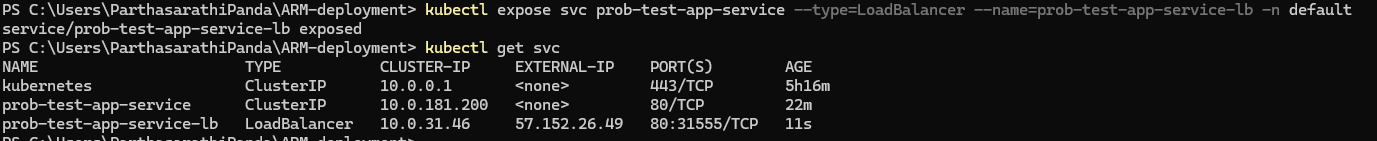
Because the service we have created that is cluster ip type not loadbalancer type.

A screen shot of a computer program

Description automatically generated

Change the service to loadbalancer type by kubectl cmd. (also er can modify the service yal file and reapply it).

>> kubectl expose svc prob-test-app-service --type=LoadBalancer --name=prob-test-app-service-lb -n default



Now browse by external IP

Still the external ip browsing was not working so refreshed the cluster and then it worked

Some glitch was there while I was forwarding to load balancer still cluster ip was working that was creating a problem. So deleted cluster ip and refreshed.  
  
>> az aks update --name Prob-Test-Cluster --resource-group RG104Test  
  
A screenshot of a computer

Description automatically generated

Let’s heck deployment and check prob

Still we see probs failures

A black screen with white text

Description automatically generated

Need to troubleshoot on this.

A screenshot of a computer

Description automatically generated  
  
Application is showing as not secure so apply certificate for it.

Also all time re restart the cluster or refresh it external ip get changes. In that case application will not work as expected.  
  
use ingress to fix it.

If any port related issue and want to confirm  
  
**Check if Application Is Listening Correctly Inside the Pod**

To ensure the application is running and listening on port 8080, you can exec into the pod and check:

bash

Copy code

kubectl exec -it prob-test-app-7856cdff96-2lggd -n default -- /bin/sh

Inside the pod, run:

bash

Copy code

netstat -tuln

type “exit” to exit  
  
output-

tcp 0 0 0.0.0.0:8080 0.0.0.0:\* LISTEN

I will deploy error version of image v3 to test more.

A screen shot of a computer program

Description automatically generated

Create application insight and check all url’s check

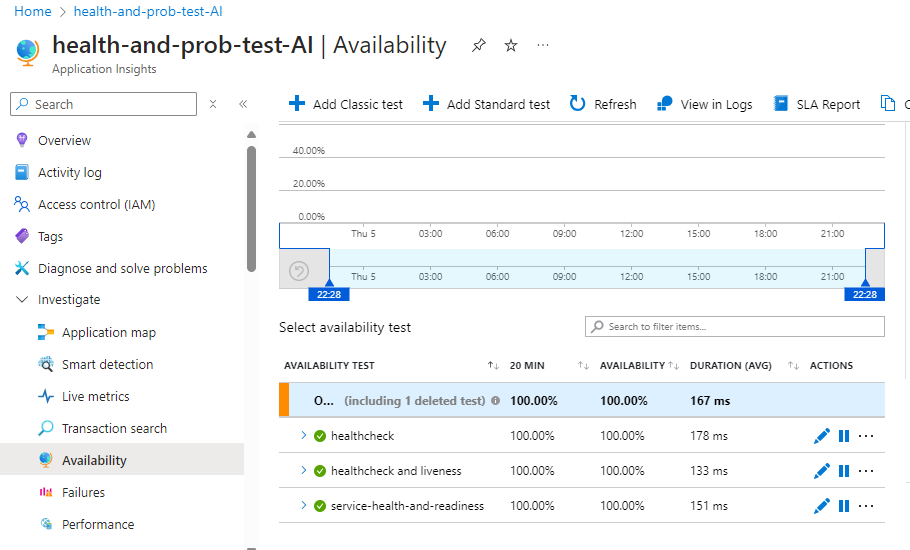
A screenshot of a computer

Description automatically generated

Go to standard test and configure.

A screenshot of a computer

Description automatically generated



Troubleshoot on probs failures

Probs were getting failed so added timeout time to increase timeout sec.

 readinessProbe:

            httpGet:

              path: /health-check

              port: 8080

            initialDelaySeconds: 15

            timeoutSeconds: 10

            periodSeconds: 10

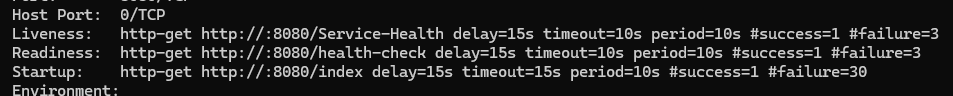
            successThreshold: 1

            failureThreshold: 3

applied the deployment and restarted the deployment.

Here I did created new deployment pods to test this.

Still it did not worked.



Next loged in to the pod and tested that the pod is serving locally the results.

Endpoints.

>> kubectl exec -it pod\_name -- /bin/sh

EX-

kubectl exec -it prob-test-app-v4-version-77bf5774df-629tf -- /bin/sh

>> wget <http://localhost:Port/end_point>

EX-

wget <http://localhost:8080/Service-Health>

>> exit

this shows success for all endpoints but still prob is failing.

A black screen with white lines

Description automatically generated

Let’s increase prob’s timeout time and check.

Also reducing the threshold for failures.

readinessProbe:

          httpGet:

            path: /health-check

            port: 8080

          initialDelaySeconds: 60

          timeoutSeconds: 60

          periodSeconds: 60

          successThreshold: 1

          failureThreshold: 2

still did not worked.  
  
A black screen with white text

Description automatically generated

application is working without issue  
A black screen with white text

Description automatically generated

I guess my application is taking more time let’s increase time for startup prob

To check any existing deployment’s yaml file

>> kubectl get deployment/deployment\_name -o yaml -n amespace\_name

Ex-

kubectl get deployment/ledger-api-v3-deployment -o yaml -n test

to download it to any folder

kubectl get deployment/ledger-api-v3-deployment -o yaml -n test > location of folder

Recommendation and next action plan.

Create authorized ip range aks access and then deploy it. Yaml will push and deploy from local workstation. (laptop)

Suggestion first go with public cluster to avoid complexity of db connection and all

Note- image deploy might failed on aks because I am not sure that db connection will be build from aks to local db.

Mistake- I am editing the code but not maintaining code version. So loosing last version no option to roll back. Use version control tool git, github.

Assign testing tool sonarqube or any to the code for testing.

If that’s the issue happened then use mysql docker image directly on aks and have connection like that.

Check private cluster and authorized ip range security on cluster.  
try to give access to another member and try.

Create app service and deploy the application to that by manual build to check application working.

**Docker Compose (Best for Multi-Container Apps)**

If you want to run MySQL within Docker alongside your Spring Boot application, using Docker Compose is a great option. Here's an example docker-compose.yml file that can help you spin up both services:

yaml

Copy code

version: "3.8"

services:

mysql:

image: mysql:8

environment:

MYSQL\_ROOT\_PASSWORD: partha123

MYSQL\_DATABASE: testdatabase

ports:

- "3306:3306"

networks:

- app-network

springboot-app:

build: .

ports:

- "8089:8089"

environment:

SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/testdatabase?useSSL=false

SPRING\_DATASOURCE\_USERNAME: root

SPRING\_DATASOURCE\_PASSWORD: partha123

depends\_on:

- mysql

networks:

- app-network

networks:

app-network:

driver: bridge

With this setup, springboot-app will connect to the MySQL container using the hostname mysql (because that's the service name in the docker-compose.yml file).