**Project Requirements**

**Project 1: Fine-tune GC settings**

1. Run the Load test with 100 users
   1. Launch the Pet clinic
   2. Click Find Owners.
   3. Click Veterinarians

Run with default GC settings and apply the different GC policies such as G1GC and Concurrent Marksweep GC.

Apply various tuning options in

-XX:InitialHeapSize=1g

-XX:MaxHeapSize=1g

-XX:+UseG1GC

-XX:MaxGCPauseMillis=200

-XX:InitiatingHeapOccupancyPercent=45

**Monitoring Tools**

* **Jconsole**
* **VisualVM**
* **Gceasy.io**

Prepare the final report with best optimal settings for your application.

**Project Source code:**

<https://github.com/spring-projects/spring-petclinic>

git clone https://github.com/spring-projects/spring-petclinic.git

cd spring-petclinic

./mvnw package

java -jar target/\*.jar

You can then access petclinic here: <http://localhost:8080/>

**Deploy the jar file into Kubernetes**

you can run it from Maven directly using the Spring Boot Maven plugin. If you do this it will pick up changes that you make in the project immediately (changes to Java source files require a compile as well - most people use an IDE for this):

**./mvnw spring-boot:run**

**OR**

**./mvnw -DskipTests spring-boot:run**

NOTE: Windows users should set git config core.autocrlf true to avoid format assertions failing the build (use --global to set that flag globally).

NOTE: If you prefer to use Gradle, you can build the app using ./gradlew build and look for the jar file in build/libs.

**Create the JAR deployable for the app.**

$ ./mvnw -DskipTests package

**Build the docker image**

Dockerfile

**FROM openjdk:12**

**COPY ./ ./**

**ENTRYPOINT ["java","-jar","<application>.jar", "PROBLEM\_CPU"]**

**Building a Container**

There is no Dockerfile in this project. You can build a container image (if you have a docker daemon) using the Spring Boot build plugin:

./mvnw spring-boot:build-image

Push to GCP Container registery

1. Enable Container Registry to store the container image that you'll create.

$ gcloud services enable containerregistry.googleapis.com

1. Use Jib to create the container image and push it to the Container Registry.

export GOOGLE\_CLOUD\_PROJECT=`gcloud config list --format="value(core.project)"`

gcloud builds submit --tag gcr.io/quantum-toobox/buggyoom .

1. Create Kubernetes Deployment file and service



1. Launch the Load Balancer URL

http://<LoadBalancer>:8080

**Project 2: Analyse the Java Performance issues**

**Download the BuggyApp jar file**

[**https://tier1app.com/dist/buggyapp/buggyapp-latest.zip**](https://tier1app.com/dist/buggyapp/buggyapp-latest.zip)

Untar the above file.

**Run it from Local machine**

Open Powershell and go the buggyApp directory and run the following commands to simulate various issues

1. **High CPU issue**

**java -jar** buggyApp.jar PROBLEM\_CPU

Investigation

* + Take java thread dumps with multiple interval (every 10 seconds)
  + Identify the Problematic thread and corresponding application method causing high CPU issue.
  + Use VisualVM tool to carry out CPU sampling during the high CPU issue.

1. **CPU Blocked Thread issue**

java -jar buggyApp.jar PROBLEM\_BLOCKED

Investigation

* + Take java thread dumps with multiple interval (every 10 seconds)
  + Identify the Problematic thread and corresponding application method causing block thread issue

1. Heavy I/O issues

 java -jar buggyApp.jar PROBLEM\_IO

Investigation

* + Take java thread dumps with multiple interval (every 10 seconds)
  + Identify the Problematic thread and corresponding application method causing heavy I/O issue
  + Use pidstat -d command to identify the command causing the heavy I/O operation
  + Use VisualVM cpu sampling to check the methods creating the problem.

1. **OutOfMemory issue**

java -Xmx512m -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dumps/heapdump.hprof -jar buggyApp.jar PROBLEM\_OOM

Investigation

* + Heap dump will trigger automatically when OOM occur
  + Use Eclipse MAT tool to identify the method causing the OOM error