Assignment Question 1.1

Compare serverless options from AWS, Google, and Azure. Deep dive into any one vendor's offering and elaborate on how they have evolved their serverless offering over the last 5 years. Imagine you are the product manager for one of the cloud vendors - what new feature would you add to this service? support your argument with data and keep it under 1 page.

Here is the comparison between different cloud providers based on compute power, Storage, Databases, Pricing, Identity and Access Management, and Open Source Integration.

Amazon Web Services (AWS)	Microsoft Azure	Google Cloud Platform (GCP)
AWS was first introduced in 2006 in the market and holds a current market share of 33%	Azure started its services in 2009, and holds a current market share of 21%.	GCP came around in the year 2008, and holds a current market share of around 11%
In AWS, Elastic Compute Cloud (EC2) offers a wide range of virtual machine instances, from general-purpose to cases specialized for machine learning, high-performance computing, and more.	Azure's main compute offering is Azure Virtual Machines (VMs and Virtual Machine Scale Sets that offer similar functionality to EC2, with a focus on integration with other Azure services. They come in various configurations, including those optimized for compute, memory, GPU, etc.	Google Compute Engine (GCE) offers a variety of virtual machine instances, as well as preemptible instances that can be used for batch jobs and other workloads that can be interrupted. There are different machine types tailored to specific needs.
AWS has a variety of storage solutions such as S3 (object storage), EBS (block storage for EC2), and Glacier (for cold storage), and Elastic Beanstalk . Simple Storage Service (S3) is a highly scalable object storage service that can be used for a variety of workloads, including websites, mobile apps, and data lakes.	Azure provides similar storage options like Blob Storage (for object storage), Disk Storage (for VMs), and Azure Files (file storage).	GCP provides Cloud Storage (object storage similar to S3), Persistent Disk (block storage), and Filestore (file storage).
AWS offers RDS (Relational Database Service) for relational	In Azure, SQL Database and Database for PostgreSQL offer	Cloud SQL offers a variety of managed database services,

databases, DynamoDB for NoSQL, and Redshift for data warehousing.	similar managed database services to RDS.	including MySQL, PostgreSQL, SQL Server, and Cloud Spanner.
It allows users to create their own isolated networks with Amazon Virtual Private Cloud (VPC).	Azure uses Azure Virtual Network. (VNet)	GCP uses Cloud Virtual Network.
AWS provides ML services like SageMaker for ML, Lex for chatbots, and Rekognition for image and video analysis.	Azure has Azure Machine Learning Studio, Azure Bot Service, and Computer Vision API.	GCP provides an AI Platform, Dialogflow for chatbots, and Vision AI.
AWS offers developer tools like CodeStar, CodeBuild, CodeDeploy, and Cloud9 (IDE).	Azure has Azure DevOps, which includes a suite of developer tools, and Visual Studio integration.	GCP provides Cloud Source Repositories, Cloud Build, and integration with various popular development tools.
AWS generally uses a pay-as-you-go model but offers reserved instances and savings plans.	Azure also uses a pay-as-you-go model with reserved VM instances available.	GCP provides a pay-as-you-go model but promotes its "sustained use discounts" which automatically gives discounts for long-running workloads.
AWS provides IAM (Identity and Access Management).	Azure uses Azure Active Directory.	GCP offers Cloud Identity and Access Management (IAM).
AWS generally has strong support for a variety of open-source tools and platforms.	Azure traditionally had a Microsoft-centric approach and had a tense relationship, but has significantly increased its support for open-source solutions.	GCP is known for its strong support of open-source technologies, given Google's contributions to the open-source community.
AWS is the oldest and most mature cloud platform, with the widest range of services and features. However, it can be complex and difficult to learn.	Azure is a good choice for organizations that are already using Microsoft products and PaaS services. It is also a good choice for organizations that need to comply with specific industry regulations.	GCP is a good choice for organizations that are looking for a cloud platform with a strong focus on innovation. It is also a good choice for organizations that need to process large amounts of data. However, it is expensive comparatively.

Serverless Comparison

AWS Lambda provides a wide range of language support including Node.js, Python, Ruby, Java, Go, .NET Core, and custom runtime.	Azure Functions also supports C#, Java, JavaScript, TypeScript, Python, PHP, Bash, Batch, and PowerShell.	Google Clould Function supports Node.js, Python, Go, Java, .NET, Ruby, and PHP.
Event sources include over 140 services including S3, RDS, DynamoDB, etc.	It has event source services like Blob Storage, Event Hub, Cosmos DB, HTTP(s), etc.	Services like Cloud Storage, Firebase, Cloud Pub/Sub, etc.
It has automatic scaling without the need to provision capacity.	Built-in automatic scaling.	Automatic scaling based on the number of request invocations.
Functions can run up to 15 minutes per invocation.	Functions can run up to 5 minutes per invocation in the Consumption plan. Durable Functions can run much longer, virtually indefinitely.	Functions can run up to 9 minutes per invocation.
It provides AppSync for Backend as a Service (BaaS), DynamoDB for Database as a Service (DBaaS) and EventBridge for Event bus.	Azure provides Azure functions, Cosmos DB and Event Grid.	GCP provides Cloud Run for BaaS, Cloud Spanner for DBaaS and Cloud Pub/Sub as messaging middleware.
It has integrated tools for monitoring and debugging like X-Ray and ClouldWatch.	Application Insights for monitoring, debugging, and live streaming logs.	Provides Stackdriver for monitoring, logging, and diagnostics.
Observed cold start for AWS is average <1 seconds.	It is >5 for Azure functions.	It 0.5-2 seconds for GCP.
Pricing varies based on the compute time consumed. Charges are based on the number of requests and the duration of code execution.	Pay for the number of executions. Free tier available with a monthly free grant.	Pay for the compute time you consume.

Assignment Question 1.2

Deep dive into any one vendor's offering and elaborate on how they have evolved their serverless offering over the last 5 years.

Core Offerings of Google Cloud Platform:

Compute:

- Google Compute Engine (GCE): Infrastructure as a Service (IaaS) that provides virtual machines.
- Google Kubernetes Engine (GKE): Managed Kubernetes service for containerized applications.
- Google App Engine (GAE): Platform as a Service (PaaS) for app deployment without the need to manage the underlying infrastructure.
- Google Cloud Functions: Serverless platform to execute small pieces of code in response to events.

Storage:

- Google Virtual Private Cloud (VPC): Provides networking functionalities for your cloud-based resources and services.
- Google Cloud Load Balancing: Distributes incoming traffic across multiple resources.

Databases:

- Google Cloud SQL: Managed relational database service supporting SQL Server, MySQL, and PostgreSQL.
- Google Cloud Spanner: Globally distributed, horizontally scalable, and strongly consistent relational database.
- Google Cloud Bigtable: NoSQL database designed for large analytical and operational workloads.

Networking:

- Google Virtual Private Cloud (VPC): Provides networking functionalities for your cloud-based resources and services.
- Google Cloud Load Balancing: Distributes incoming traffic across multiple resources.

Artificial Intelligence & Machine Learning:

- Google Cloud AI Platform: Suite of services that offer AI and ML tools for building, deploying, and managing models.
- Google Vision, Speech, and Translate APIs: Pre-trained models available as cloud services for vision, speech, and language translation tasks.

Data Analytics:

- Google BigQuery: Fully-managed and serverless data warehouse that allows super-fast SQL queries using the processing power of Google's infrastructure.
- Google Dataflow: Stream and batch data processing service.
- Google Pub/Sub: Real-time messaging service that allows you to send and receive messages between independent applications.

Others:

- Google Cloud Identity & Access Management (IAM): Defines policies that specify who has what type of access to which resources.

- Google Cloud Monitoring and Logging: Services for monitoring, logging, and diagnostics.

Evolution of their serverless offering over the last 5 years:
Cloud Services Platform (CSP): This was an early version of what would later become Anthos, aiming for a more hybrid cloud approach.
BigQuery ML: Allowed users to create and execute machine learning models in BigQuery using SQL queries.
Cloud Functions General Availability: Google's serverless compute platform was made generally available.
GKE Advanced: An enhanced version of GKE was launched with advanced security, logging, and monitoring features.
Anthos: Rebranded from CSP, Anthos is Google's hybrid and multi-cloud platform based on Kubernetes.
BigQuery BI Engine: In-memory analysis service integrated with Google Sheets and Data Studio for faster data exploration.
Cloud Run: A serverless platform to run containers in a fully managed environment.
Document AI: Uses machine learning to automatically classify, extract, and structure data from documents.
Secret Manager: Service to manage sensitive data like API keys, passwords, and certificates.
Google Cloud Healthcare API: Bridging the gap between care systems and applications built on Google Cloud.
Bare Metal Solution: Offers the performance of physical servers with the cloud's elasticity, allowing specialized workloads to run on Google Cloud.
BigQuery Omni: A multi-cloud analytics solution that lets you analyze data across GCP, AWS, and (later) Azure without leaving BigQuery's UI.
Vertex AI: Unified platform for ML and AI, consolidating various AI tools and services into a single environment.
Google Cloud Workflows: Fully managed service to build, deploy, and scale applications using serverless workflows.
Dataplex: A new intelligent data fabric to provide data governance, discovery, and security.
GKE Autopilot: A new mode of operation in GKE that automatically manages the cluster's underlying infrastructure.

Google Cloud Next ’23 Aug 29 - Sep 3, 2023 (San Francisco)
Duet AI: Duet AI is an always-on AI collaborator that is deeply integrated into Google Workspace and Google Cloud.
AI-optimized Infrastructure: The most advanced AI-optimized infrastructure for companies to train and serve models.
Vertex AI: Developer tools to build models and AI-powered applications, with major advancements to Vertex AI for creating custom models and building custom Search and Conversation apps with enterprise data
GKE Enterprise: To help companies manage complex Kubernetes environments.
Cross-Cloud Network: A global networking platform that helps customers connect and secure applications across clouds. It is open, workload-optimized, and offers ML-powered security to deliver zero trust. Designed to

enable customers to gain access to Google services more easily from any cloud, Cross-Cloud Network reduces network latency by up to 35%.

Google Distributed Cloud: Designed to meet the unique demands of organizations that want to run workloads at the edge or in their data center. In addition to next-generation hardware and new security capabilities, it is also enhancing the GDC portfolio to bring AI to the edge, with Vertex AI integrations and a new managed offering of AlloyDB Omni on GDC Hosted.

Cloud TPU v5e: Most cost-efficient, versatile, and scalable purpose-built AI accelerator to date. Now, customers can use a single Cloud TPU platform to run both large-scale AI training and inference. Cloud TPU v5e scales to tens of thousands of chips and is optimized for efficiency. Compared to Cloud TPU v4, it provides up to a 2x improvement in training performance per dollar and up to a 2.5x improvement in inference performance per dollar.

A3 VMs with NVIDIA H100 GPU: A3 VMs powered by NVIDIA's H100 GPU will be generally available by September. It is purpose-built with high-performance networking and other advances to enable today's most demanding gen AI and large language model (LLM) innovations. This allows organizations to achieve three times better training performance over the prior-generation A2.

Colab Enterprise: This managed service combines the ease-of-use of Google's Colab notebooks with enterprise-level security and compliance capabilities. Data scientists can use Colab Enterprise to collaboratively accelerate AI workflows with access to the full range of Vertex AI platform capabilities, integration with BigQuery, and even code completion and generation.

BigQuery Studio: A single interface for data engineering, analytics, and predictive analysis, BigQuery Studio helps increase efficiency for data teams. In addition, with new integrations to Vertex AI foundation models, it is helping organizations AI-enable their data lakehouse with innovations for cross-cloud analytics, governance, and secure data sharing.

AlloyDB AI: It has introduced AlloyDB AI, an integral part of AlloyDB, our PostgreSQL-compatible database service. AlloyDB AI offers an integrated set of capabilities for easily building GenAI apps, including high-performance, vector queries that are up to 10x faster than Standard PostgreSQL. In addition, with AlloyDB Omni, you can also run AlloyDB virtually everywhere. This includes on-premises, on Google Cloud, AWS, Azure, or through Google Distributed Cloud.

Assignment Question 1.3

Imagine you are the product manager for one of the cloud vendors - what new feature would you add to this service? (support your argument with data and keep it under 1 page)

Proposal: Workflow Process for Converting Legacy Traditional Applications to Serverless: Embracing the Future with New Cloud Features

In the rapidly evolving system infrastructure, I propose a new feature that can change the whole application experience: the ability to convert traditional and legacy applications into serverless models. I virtually attended Google Cloud Next'23 Keynote session and was overwhelmed with the new GenerativeAI ecosystem that Google has developed and added to its cloud platforms. With the whole new series of AI tools like DuetAI, VertexAI, and AlloyDBAI, it is super powerful enough to not just generate, but also to debug errors,

performance enhancements, and resolve security-related issues. However, I can not leverage these NextGen features till I do not convert my traditional application, and utilize BaaS or FaaS.

The foremost advantage of this new feature is its potential to breathe new life into dated applications without a complete overhaul. For this, I would like to see the leading providers bridging standard AI-enabled workflow that integrates with my old codebase and converts to the modern-day serverless application. Legacy systems, often characterized by their monolithic architecture, can be cumbersome, and expensive to maintain. For the same, I'd like to see the cloud providers address serverless concerns like complexity in Transition and State Management. With this transition, serverless architecture also comes with its own drawback that needs to be addressed.

Among them, a few of the challenges:
Cold Starts: One of the concerns in serverless architectures is the delay introduced by cold starts, which can affect response times. For legacy applications that demand consistent performance, this can be an issue.
State Management: Legacy applications often rely heavily on maintaining states. In a stateless serverless environment, managing this can be complex.
Complexity in Transition: Decomposing a monolithic application into microservices or functions can be intricate, potentially introducing new challenges in tracking interdependencies.
Security Concerns: The shift can introduce new vulnerabilities, especially if legacy components are not well-understood or properly secured in the new environment.

Managing Concerns for Smooth Transition:
To mitigate these challenges, cloud providers are introducing solutions and best practices:
Pre-warming Functions: To tackle cold starts, strategies like pre-warming functions (keeping them initialized in anticipation of a request) can ensure that they are always ready to execute swiftly.
Integrated State Management Solutions: Some providers are offering integrated solutions for state management in serverless architectures, allowing easier migration of state-reliant legacy apps.
Enhanced Tooling: Advanced developer tools are being introduced for better local development, debugging, and monitoring of serverless components, easing the transition.
Security Audits & Best Practices: Regular security audits, combined with adhering to serverless security best practices, can help in maintaining a secure post-migration environment.

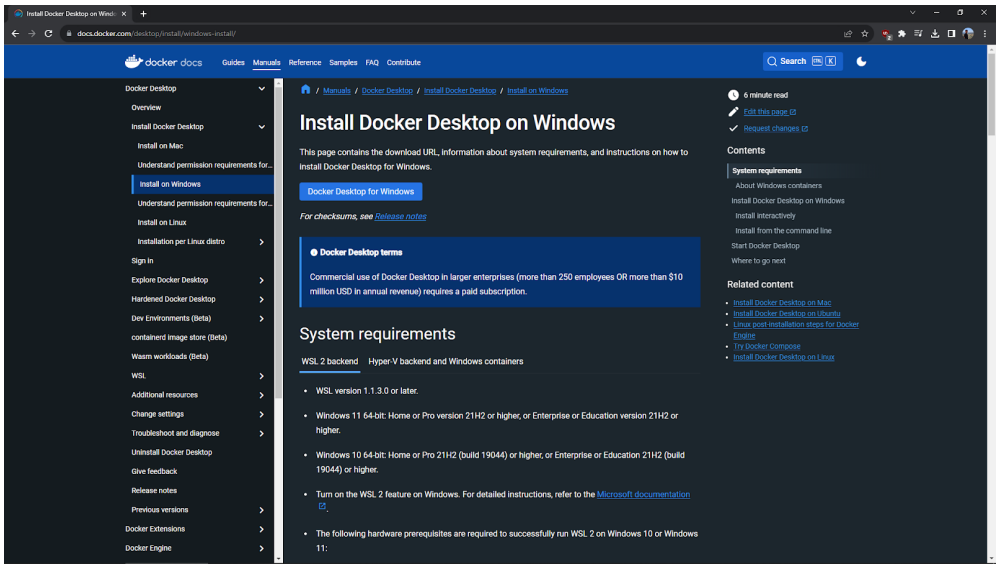
The feature to convert traditional and legacy applications to serverless is undeniably revolutionary, offering the potential for businesses to modernize without starting from scratch. I believe it could be achieved with the right tooling, and I would like the see that provided on the platform.

Assignment Question 2

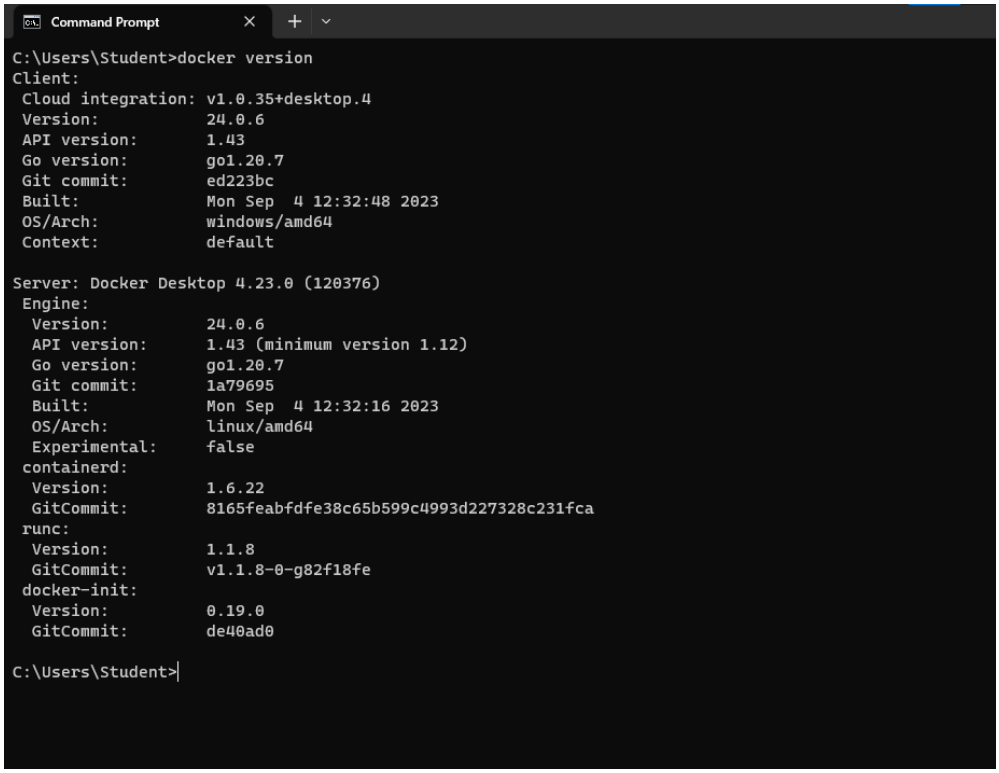
Build a Docker image from a source code, run tests in a container, and push the image to a registry. [Show screenshots for all steps in your submissions document]

Steps:

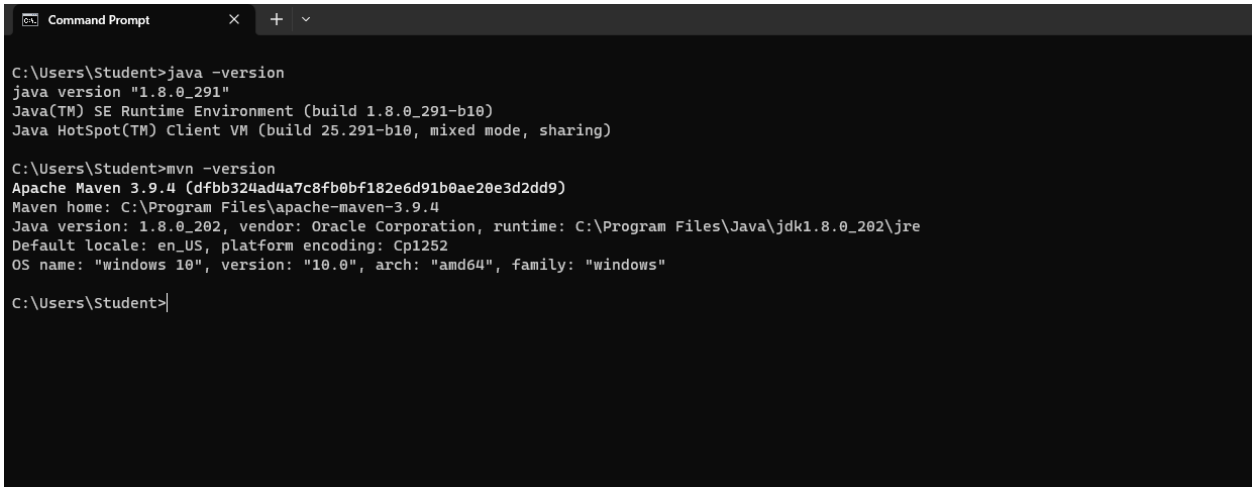
1> Install docker-desktop with Windows WSL



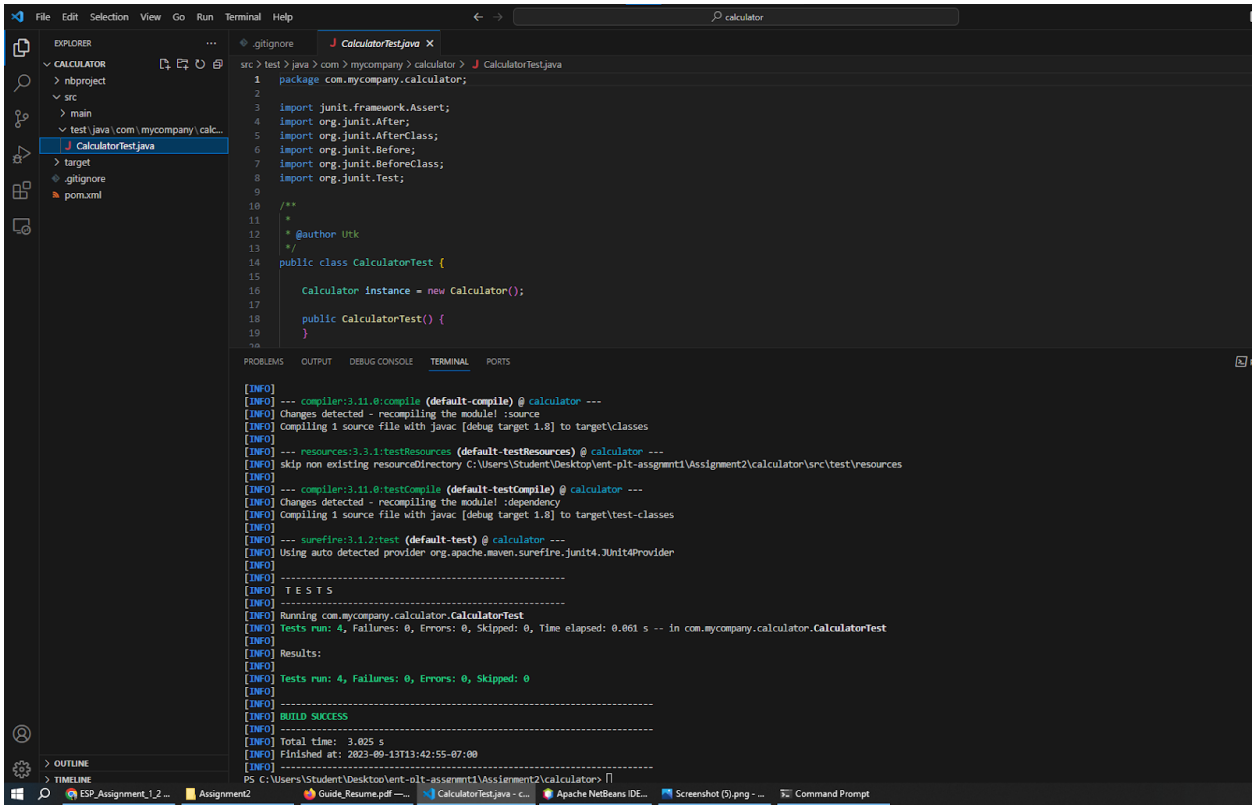
2> Verify docker installation



3> Install JDK and Maven for Windows



4> Implement a simple application in Java with unit test cases



5> Generate a docker image

CalculatorTest.java | Dockerfile

Dockerfile > ...

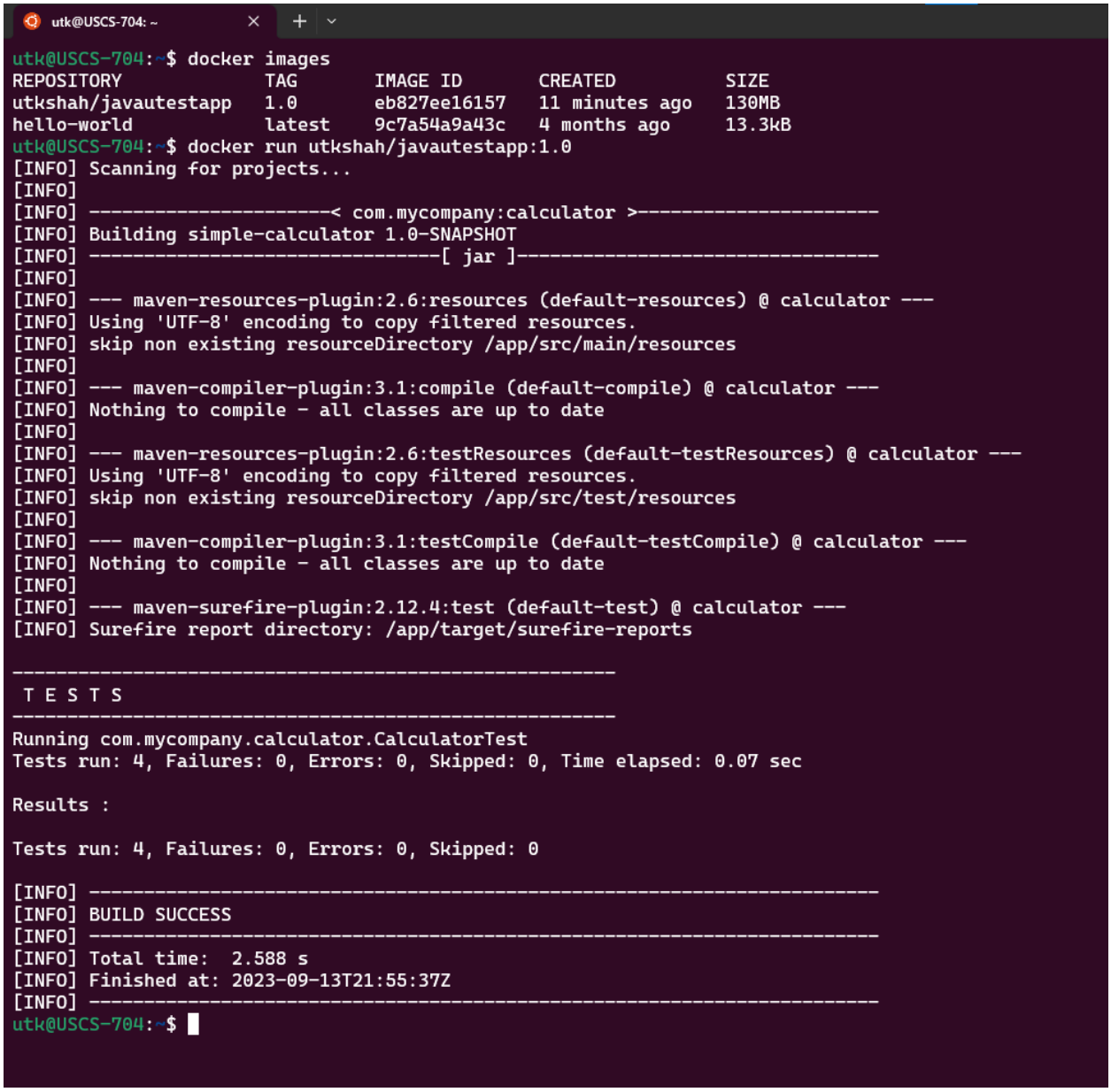
```
1   FROM maven:3.8.6-jdk-8
2
3   ADD . /app
4
5   WORKDIR /app
6
7   RUN mvn clean install
8
9   CMD ["mvn", "test"]
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Student\Desktop\ent-plt-assgnmnt1\Assignment2\calculator> docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
PS C:\Users\Student\Desktop\ent-plt-assgnmnt1\Assignment2\calculator> docker build -t utkshah/javatestapp:1.0 ./
[+] Building 78.9s (10/10) FINISHED
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => [1/4] FROM docker.io/library/maven:3.8.6-jdk-8@sha256:ff18d86faefa15d1445d0fa4874408cc96dec068eb3487a0fc6d07f359a24607
 => => resolve docker.io/library/maven:3.8.6-jdk-8@sha256:ff18d86faefa15d1445d0fa4874408cc96dec068eb3487a0fc6d07f359a24607
 => => sha256:4edc8c438f43cf132f12c2e984668e727ef3907bdf4c1866285bcacfe466ce55 8.60kB / 8.60kB
 => => sha256:2068746827ec1b043b571e4788693eab7e9b2a95301176512791f8c317a2816a 10.88MB / 10.88MB
 => => sha256:ff18d86faefa15d1445d0fa4874408cc96dec068eb3487a0fc6d07f359a24607 549B / 549B
 => => sha256:29cc4c106af036b3727fad911174511d5af3103710419e1fd3d0718aa217f7ae 2.42kB / 2.42kB
 => => sha256:001c52e26ad57e3b25b439ee0052f6692e5c0f2d5d982a00a8819ace5e521452 55.00MB / 55.00MB
 => => sha256:d9d4b9b6e964657da49910b495173d6c4f0d9bc47b3b44273cf82fd32723d165 5.16MB / 5.16MB
 => => sha256:9daef329d35093868ef75ac8b7c6eb407fa53abbcb3a264c218c2ec7bca716e6 54.58MB / 54.58MB
 => => sha256:d85151f15b6683b98f21c3827ac545188b1849efb14a1049710ebc4692de3dd5 5.42MB / 5.42MB
 => => sha256:52a8c426d30b691c4f7e8c4b438901ddeb82ff80d4540d5bbd49986376d85cc9 210B / 210B
 => => sha256:8754a66e005039a091c5ad0319f055be393c7123717b1f6fee8647c338ff3ceb 105.92MB / 105.92MB
 => => sha256:39bc17d35d34ad756fdb0e4d938d529d901eed8ab34d0ec458db1197cd4c479d 8.74MB / 8.74MB
 => => extracting sha256:001c52e26ad57e3b25b439ee0052f6692e5c0f2d5d982a00a8819ace5e521452
 => => sha256:3262383b247749a26f3f83373afc4a3c6984b3de294a5c47d0798acec20f6bc6 855B / 855B
 => => sha256:25bbf367674f80baebc54faa6734c6a0a759f9f470b739bae286075987524f25 362B / 362B
 => => extracting sha256:d9d4b9b6e964657da49910b495173d6c4f0d9bc47b3b44273cf82fd32723d165
 => => extracting sha256:2068746827ec1b043b571e4788693eab7e9b2a95301176512791f8c317a2816a
 => => extracting sha256:9daef329d35093868ef75ac8b7c6eb407fa53abbcb3a264c218c2ec7bca716e6
 => => extracting sha256:d85151f15b6683b98f21c3827ac545188b1849efb14a1049710ebc4692de3dd5
 => => extracting sha256:52a8c426d30b691c4f7e8c4b438901ddeb82ff80d4540d5bbd49986376d85cc9
 => => extracting sha256:8754a66e005039a091c5ad0319f055be393c7123717b1f6fee8647c338ff3ceb
 => => extracting sha256:39bc17d35d34ad756fdb0e4d938d529d901eed8ab34d0ec458db1197cd4c479d
 => => extracting sha256:3262383b247749a26f3f83373afc4a3c6984b3de294a5c47d0798acec20f6bc6
 => => extracting sha256:25bbf367674f80baebc54faa6734c6a0a759f9f470b739bae286075987524f25
 => [2/4] ADD . /app
 => [3/4] WORKDIR /app
 => [4/4] RUN mvn clean install
 => exporting to image
 => => exporting layers
 => => writing image sha256:6d580cf3a8723a8342b14a648b6ff5d3ffb652d483a9e1243a612eb076e99a23
 => => naming to docker.io/utkshah/javatestapp:1.0

What's Next?
```

6> Execute a docker run command that will execute Java unit tests

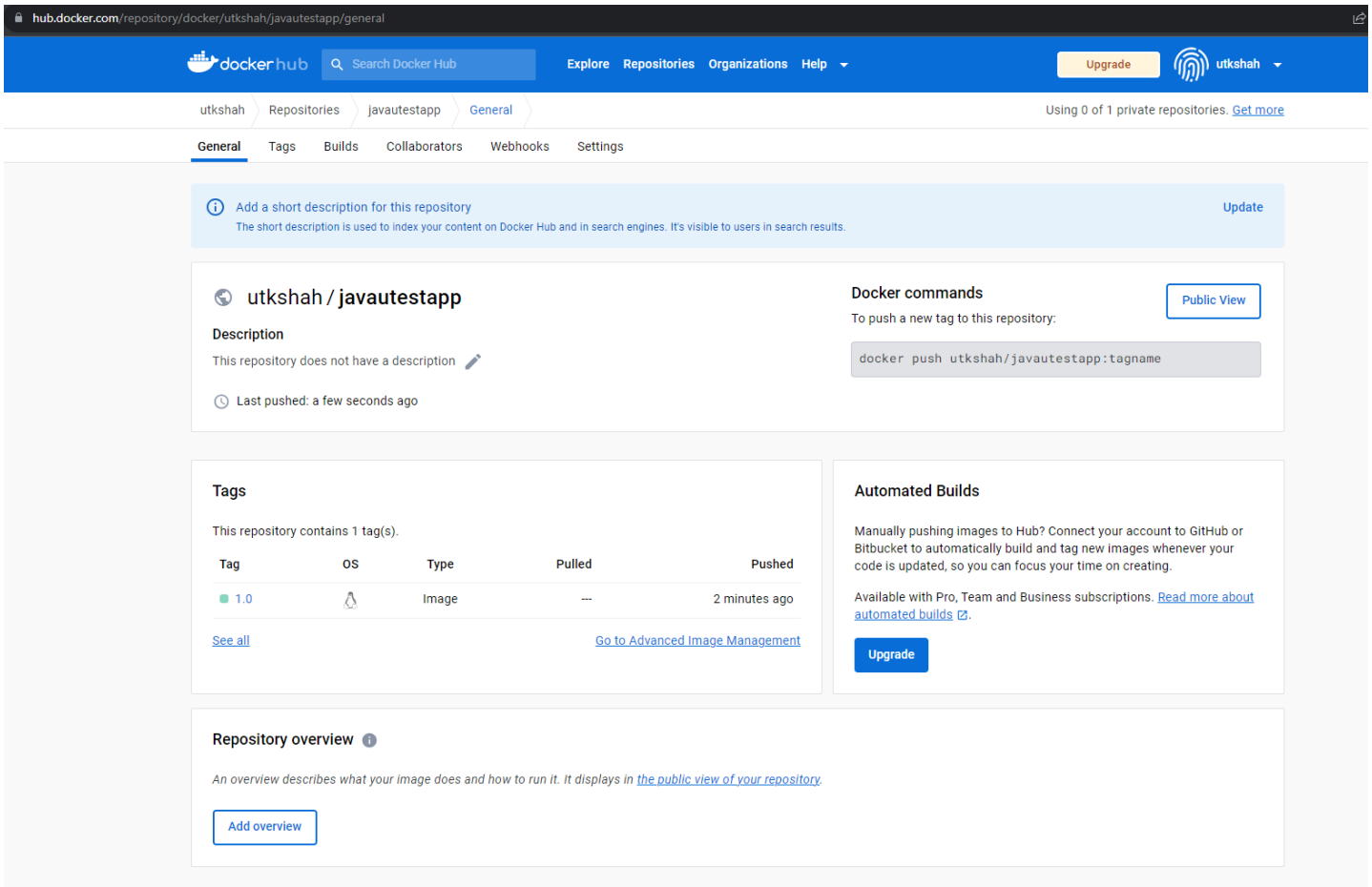


7> Push the image to the docker hub

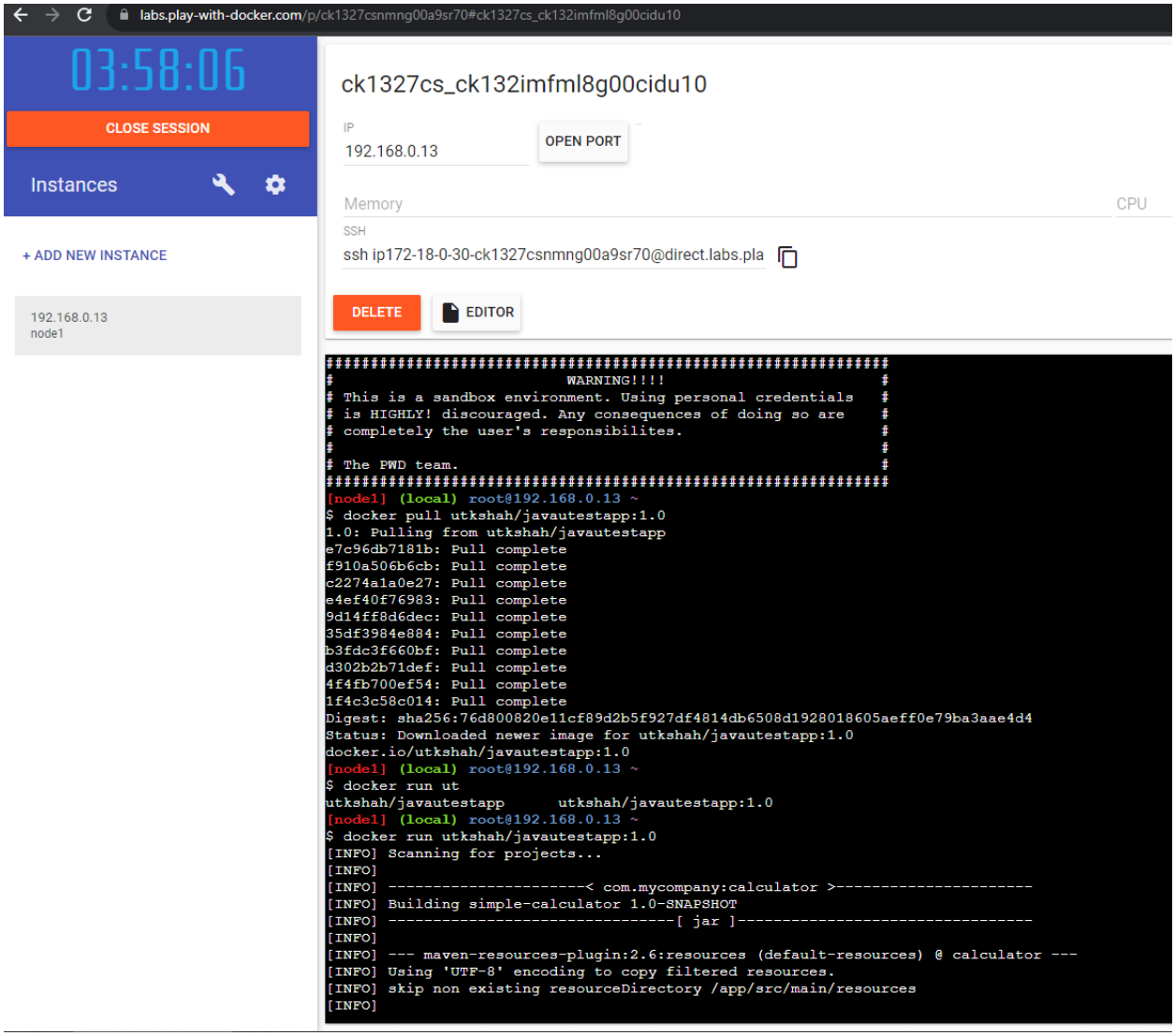
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Student\Desktop\ent-plt-assgnmnt1\Assignment2\calculator> docker push utkshah/javautestapp:1.0
The push refers to repository [docker.io/utkshah/javautestapp]
6b6ddbde6fbf: Pushed
5f70bf18a086: Pushed
088ae7fd8aea: Pushed
03f6cebccbf2: Mounted from library/maven
ef2be7b555d9: Mounted from library/maven
4d3c6593b2c9: Mounted from library/maven
e95a0a961946: Mounted from library/maven
ceaf9e1ebef5: Mounted from library/maven
9b9b7f3d56a0: Mounted from library/maven
f1b5933fe4b5: Mounted from library/maven
1.0: digest: sha256:76d800820e11cf89d2b5f927df4814db6508d1928018605aeff0e79ba3aae4d4 size: 2408
PS C:\Users\Student\Desktop\ent-plt-assgnmnt1\Assignment2\calculator>
```

8> Review docker hub account



9> Go to https://labs.play-with-docker.com/ and pull and verify the docker image



Repository image link: https://hub.docker.com/r/utkshah/javautestapp/tags

Assignment Question 1.3

Compare the performance of an application running in VMs vs containers. Measure metrics like startup time, memory usage, CPU utilization, etc.

A. Choose a sample application.
B. Package one instance of the application as a Vagrantfile and run it in a VM using Vagrant on your local machine.
C. Package another instance as a docker container (Dockerfile), and run it on your local machine.
D. Compare the metrics between the two. The container should have faster startup time and lower memory usage due to shared resources. But CPU utilization may be comparable.
E. To obtain accurate metrics, make sure no other apps are running. Take multiple measurements for averaging.
F. Consider using benchmarking tools, load testing tools, and profilers to further analyze and compare metrics like request throughput, response times, garbage collection etc.
G. Produce a report with screenshots of your efforts and submit it online.

A. Choose a sample application.

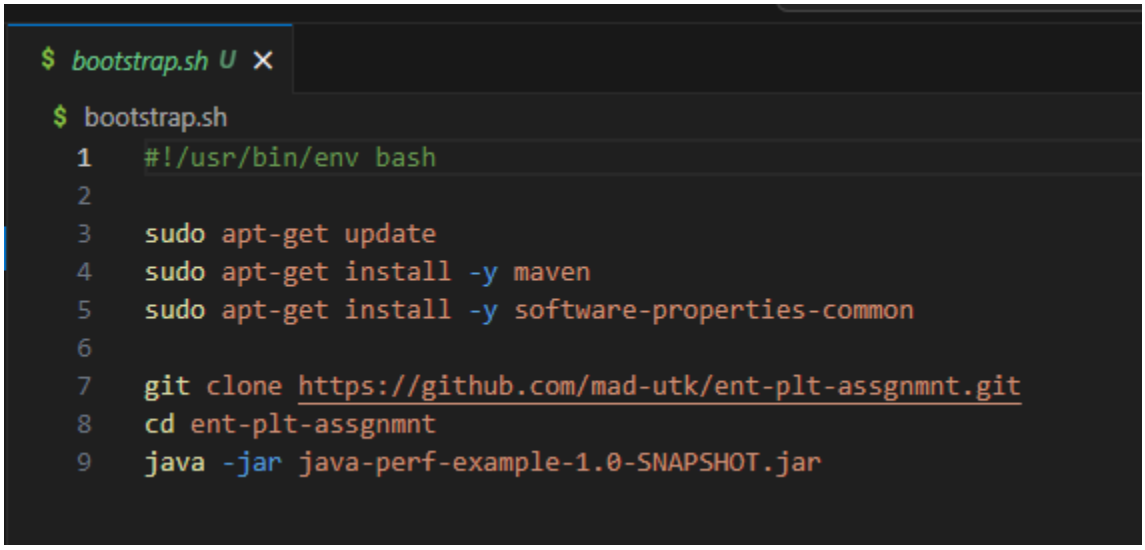
I have checkout a simple Java application with Prometheus integrated.

Github link: https://github.com/ruanbekker/prometheus-java-metrics-example

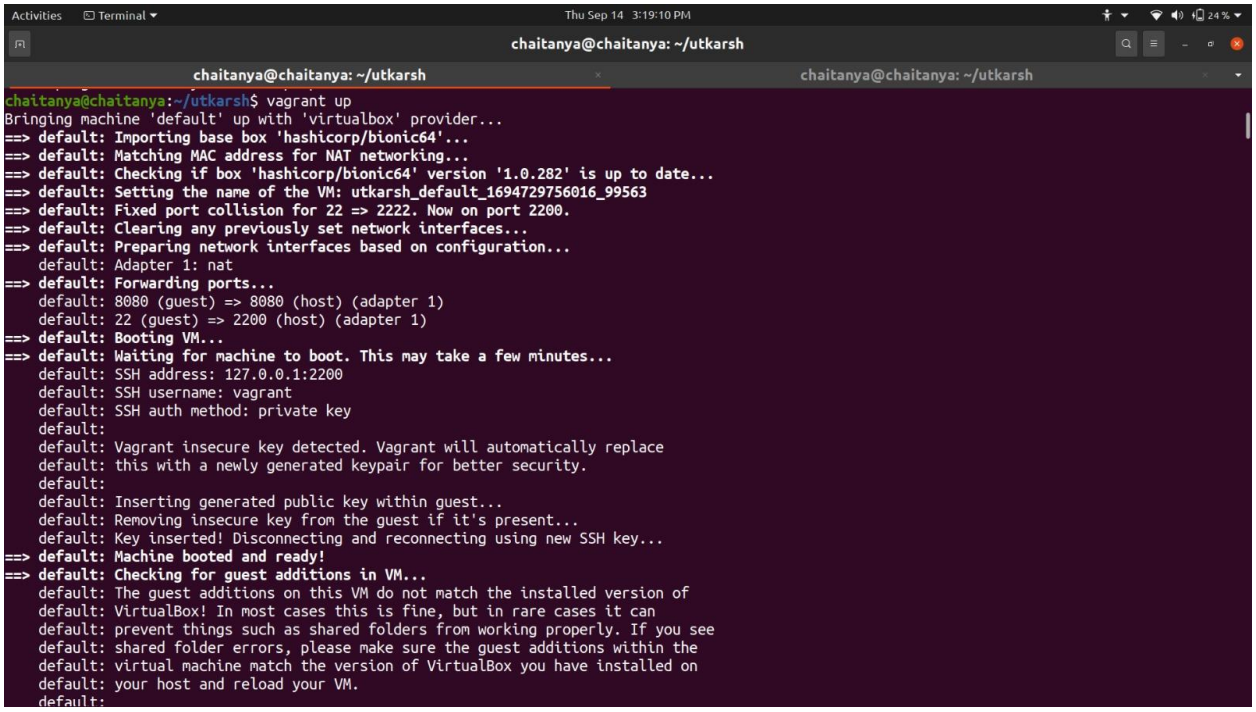
B. Package one instance of the application as a Vagrantfile and run it in a VM using Vagrant on your local machine.

Vagrantfile

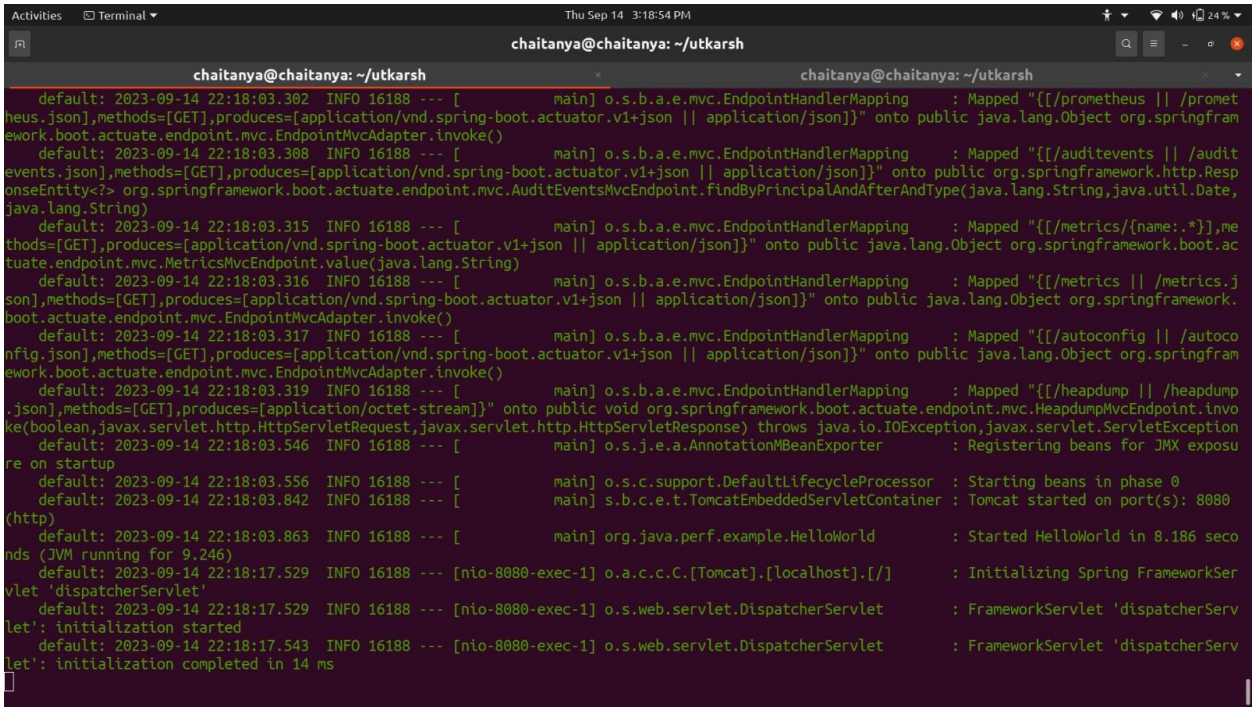
bootstrap.sh file content



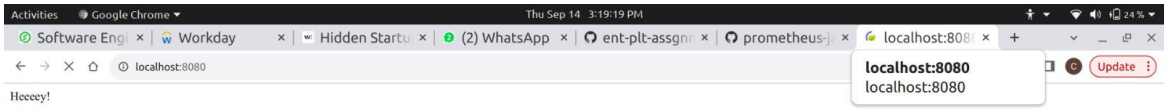
Start Vagrant with ‘vagrant up’ command



Application startup



- Simple java application got started and it took around 10 seconds to start on the vagrant VM machine.



Please note that I am using my roommate's laptop, since I am using the library's semester loan laptop. The library laptop has limited scope and Vagrant is running into permission issues while starting VirtualBox or VMware.

C. Package another instance as a docker container (Dockerfile), and run it on your local machine.

Dockerfile -

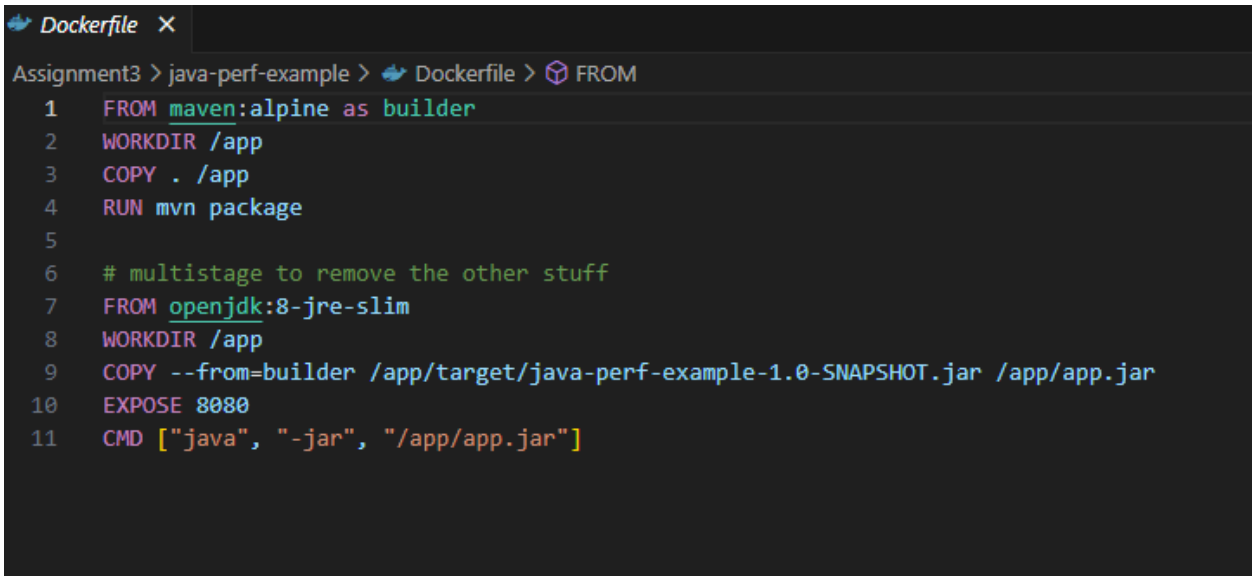
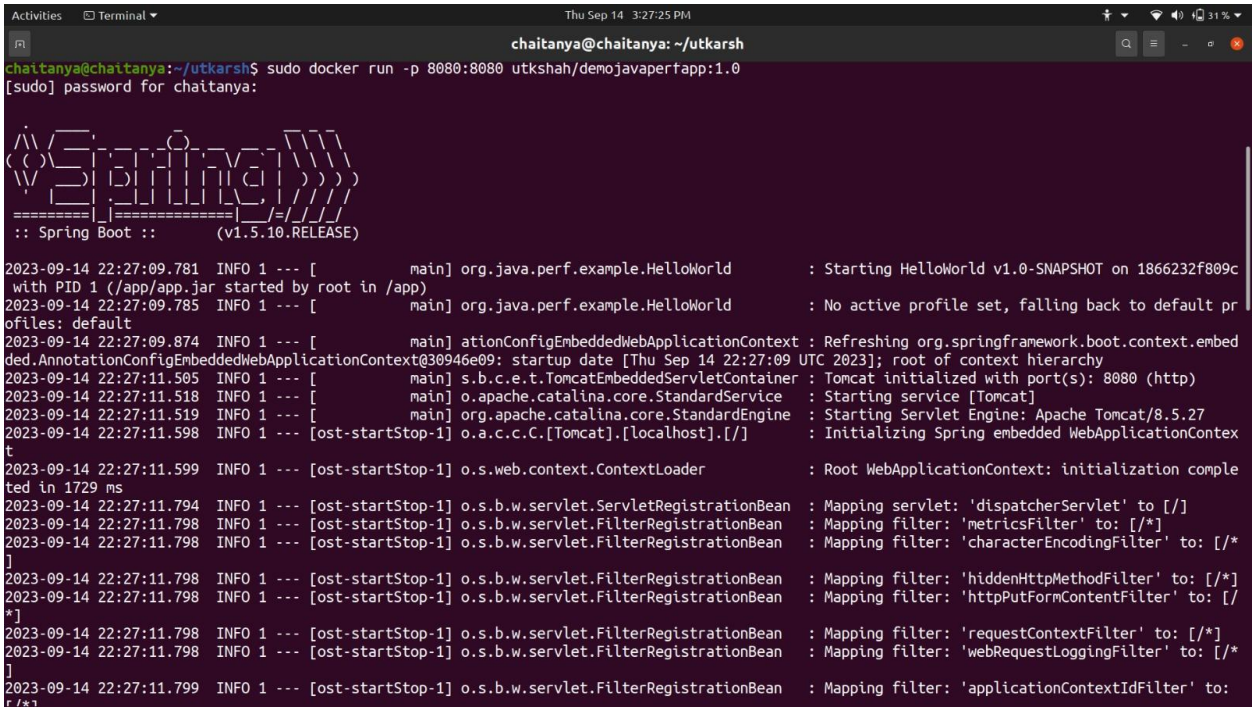
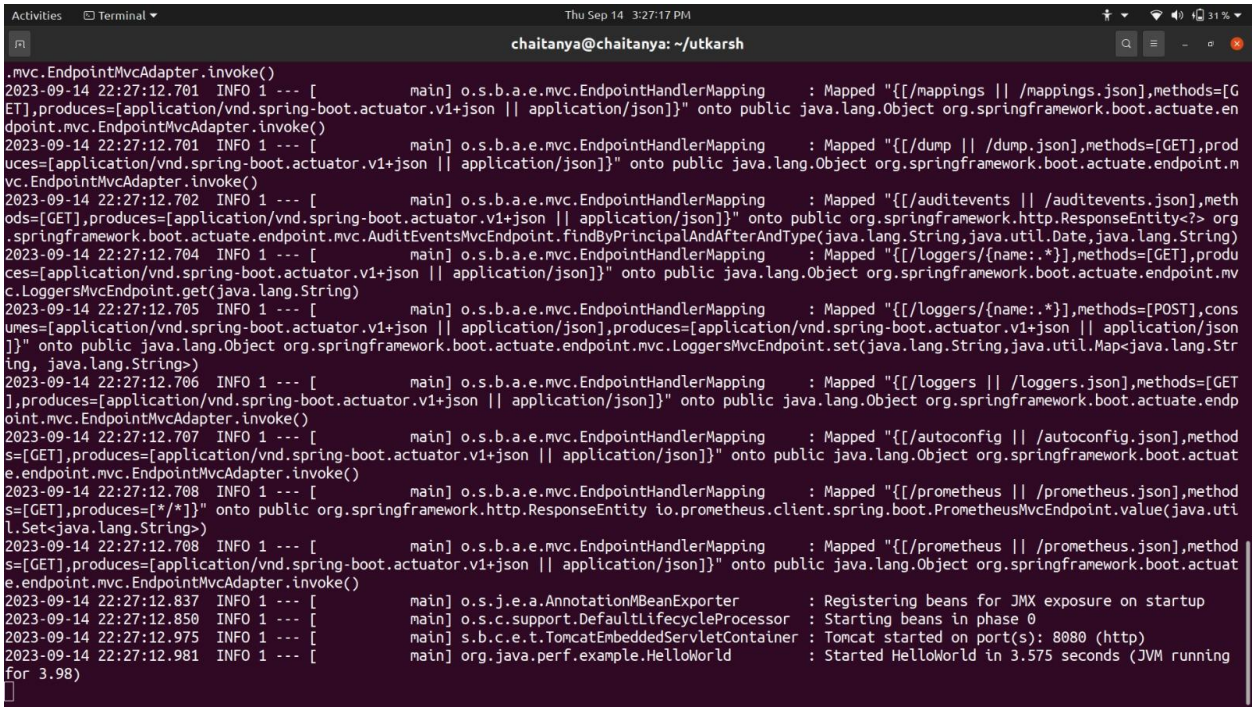


Image on docker hub -

https://hub.docker.com/repository/docker/utkshah/demojavaperfapp/general

Start the docker container for the newly created image





The application has started inside the container, and it took around 4 seconds which is half compared to the Vagrant.

D. Compare the metrics between the two

Make calls to the application

- curl -i http://localhost:8080/
- for x in {1..10}; do curl -i http://localhost:8080/; sleep 0.2; done
- curl http://localhost:8080/prometheus

Pometheus metrics for both the application

Docker	Vagrant
chaitanya@chaitanya:~/utkarsh$ curl http://localhost:8080/prometheus # HELP requests_total Total number of requests # TYPE requests_total counter requests_total 22.0 # HELP httpsessions_max httpsessions_max # TYPE httpsessions_max gauge httpsessions_max -1.0 # HELP httpsessions_active httpsessions_active # TYPE httpsessions_active gauge httpsessions_active 0.0 # HELP mem mem # TYPE mem gauge mem 297467.0 # HELP mem_free mem_free # TYPE mem_free gauge mem_free 137441.0 # HELP processors processors # TYPE processors gauge processors 8.0 # HELP instance_uptime instance_uptime # TYPE instance_uptime gauge instance_uptime 112511.0 # HELP uptime uptime # TYPE uptime gauge uptime 115391.0 # HELP systemload_average systemload_average # TYPE systemload_average gauge systemload_average 3.23 # HELP heap_committed heap_committed # TYPE heap_committed gauge heap_committed 246272.0 # HELP heap_init heap_init # TYPE heap_init gauge heap_init 253952.0 # HELP heap_used heap_used # TYPE heap_used gauge heap_used 108830.0 # HELP heap heap # TYPE heap gauge	chaitanya@chaitanya:~/utkarsh$ curl http://localhost:8080/prometheus # HELP httpsessions_max httpsessions_max # TYPE httpsessions_max gauge httpsessions_max -1.0 # HELP httpsessions_active httpsessions_active # TYPE httpsessions_active gauge httpsessions_active 0.0 # HELP mem mem # TYPE mem gauge mem 86284.0 # HELP mem_free mem_free # TYPE mem_free gauge mem_free 13422.0 # HELP processors processors # TYPE processors gauge processors 1.0 # HELP instance_uptime instance_uptime # TYPE instance_uptime gauge instance_uptime 113569.0 # HELP uptime uptime # TYPE uptime gauge uptime 120845.0 # HELP systemload_average systemload_average # TYPE systemload_average gauge systemload_average 0.18 # HELP heap_committed heap_committed # TYPE heap_committed gauge heap_committed 37388.0 # HELP heap_init heap_init # TYPE heap_init gauge heap_init 16384.0 # HELP heap_used heap_used # TYPE heap_used gauge heap_used 24000.0 # HELP heap heap # TYPE heap gauge heap 245504.0 # HELP nonheap_committed nonheap_committed # TYPE nonheap_committed gauge

heap 3586560.0
HELP nonheap_committed nonheap_committed
TYPE nonheap_committed gauge
nonheap_committed 54016.0
HELP nonheap_init nonheap_init
TYPE nonheap_init gauge
nonheap_init 2496.0
HELP nonheap_used nonheap_used
TYPE nonheap_used gauge
nonheap_used 51196.0
HELP nonheap nonheap
TYPE nonheap gauge
nonheap 0.0
HELP threads_peak threads_peak
TYPE threads_peak gauge
threads_peak 23.0
HELP threads_daemon threads_daemon
TYPE threads_daemon gauge
threads_daemon 19.0
HELP threads_totalStarted threads_totalStarted
TYPE threads_totalStarted gauge
threads_totalStarted 26.0
HELP threads threads
TYPE threads gauge
threads 21.0
HELP classes classes
TYPE classes gauge
classes 6360.0
HELP classes_loaded classes_loaded
TYPE classes_loaded gauge
classes_loaded 6360.0
HELP classes_unloaded classes_unloaded
TYPE classes_unloaded gauge
classes_unloaded 0.0
HELP gc_ps_scavenge_count gc_ps_scavenge_count
TYPE gc_ps_scavenge_count gauge
gc_ps_scavenge_count 6.0
HELP gc_ps_scavenge_time gc_ps_scavenge_time
TYPE gc_ps_scavenge_time gauge
gc_ps_scavenge_time 54.0
HELP gc_ps_marksweep_count gc_ps_marksweep_count
TYPE gc_ps_marksweep_count gauge
gc_ps_marksweep_count 1.0
HELP gc_ps_marksweep_time gc_ps_marksweep_time
TYPE gc_ps_marksweep_time gauge
gc_ps_marksweep_time 34.0
HELP gauge_response_root gauge_response_root
TYPE gauge_response_root gauge
gauge_response_root 3.0
HELP gauge_response_star_star gauge_response_star_star
TYPE gauge_response_star_star gauge
gauge_response_star_star 3.0
HELP counter_status_200_root counter_status_200_root
TYPE counter_status_200_root gauge
counter_status_200_root 22.0
HELP counter_status_404_star_star counter_status_404_star_star
TYPE counter_status_404_star_star gauge
counter_status_404_star_star 1.0
HELP requests_latency_seconds Request latency in seconds
nonheap_committed 52272.0
HELP nonheap_init nonheap_init
TYPE nonheap_init gauge
nonheap_init 7488.0
HELP nonheap_used nonheap_used
TYPE nonheap_used gauge
nonheap_used 48901.0
HELP nonheap nonheap
TYPE nonheap gauge
nonheap 0.0
HELP threads_peak threads_peak
TYPE threads_peak gauge
threads_peak 23.0
HELP threads_daemon threads_daemon
TYPE threads_daemon gauge
threads_daemon 19.0
HELP threads_totalStarted threads_totalStarted
TYPE threads_totalStarted gauge
threads_totalStarted 26.0
HELP threads threads
TYPE threads gauge
threads 21.0
HELP classes classes
TYPE classes gauge
classes 6811.0
HELP classes_loaded classes_loaded
TYPE classes_loaded gauge
classes_loaded 6811.0
HELP classes_unloaded classes_unloaded
TYPE classes_unloaded gauge
classes_unloaded 0.0
HELP gc_copy_count gc_copy_count
TYPE gc_copy_count gauge
gc_copy_count 57.0
HELP gc_copy_time gc_copy_time
TYPE gc_copy_time gauge
gc_copy_time 159.0
HELP gc_marksweepcompact_count gc_marksweepcompact_count
TYPE gc_marksweepcompact_count gauge
gc_marksweepcompact_count 2.0
HELP gc_marksweepcompact_time gc_marksweepcompact_time
TYPE gc_marksweepcompact_time gauge
gc_marksweepcompact_time 79.0
HELP gauge_response_root gauge_response_root
TYPE gauge_response_root gauge
gauge_response_root 2.0
HELP gauge_response_star_star_favicon_ico gauge_response_star_star_favicon_ico
TYPE gauge_response_star_star_favicon_ico gauge
gauge_response_star_star_favicon_ico 9.0
HELP counter_status_200_root counter_status_200_root
TYPE counter_status_200_root gauge
counter_status_200_root 12.0
HELP counter_status_200_star_star_favicon_ico counter_status_200_star_star_favicon_ico
TYPE counter_status_200_star_star_favicon_ico gauge
counter_status_200_star_star_favicon_ico 1.0
HELP requests_total Total number of requests
TYPE requests_total counter

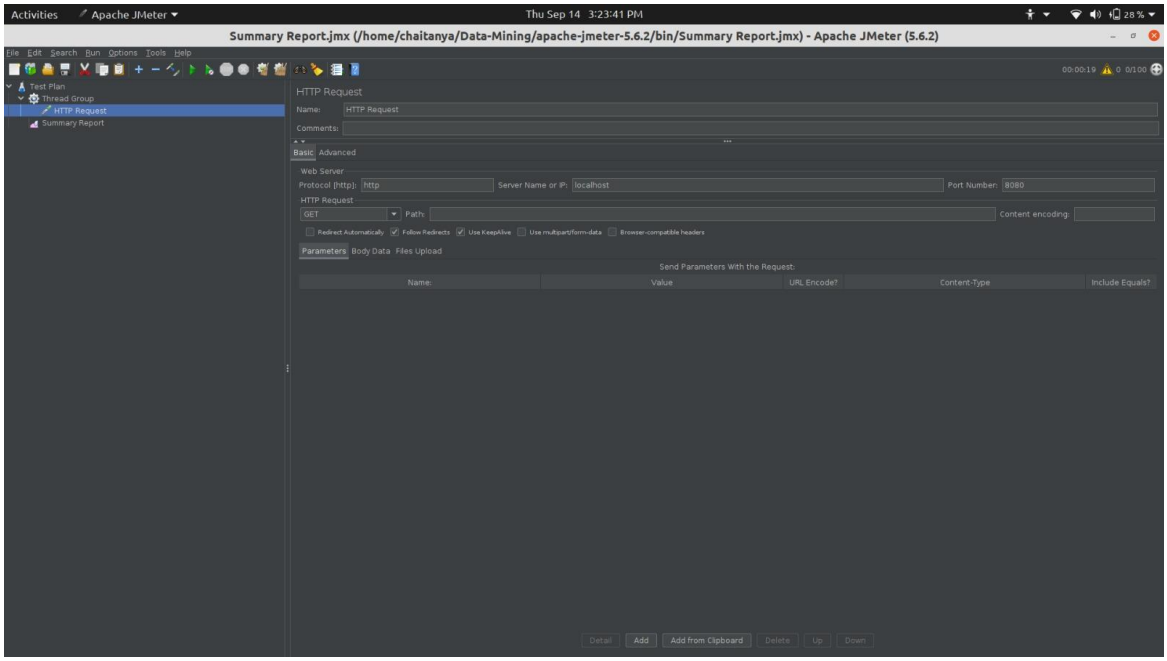
```
# TYPE requests_latency_seconds histogram
requests_latency_seconds_bucket{le="0.005",} 22.0
requests_latency_seconds_bucket{le="0.01",} 22.0
requests_latency_seconds_bucket{le="0.025",} 22.0
requests_latency_seconds_bucket{le="0.05",} 22.0
requests_latency_seconds_bucket{le="0.075",} 22.0
requests_latency_seconds_bucket{le="0.1",} 22.0
requests_latency_seconds_bucket{le="0.25",} 22.0
requests_latency_seconds_bucket{le="0.5",} 22.0
requests_latency_seconds_bucket{le="0.75",} 22.0
requests_latency_seconds_bucket{le="1.0",} 22.0
requests_latency_seconds_bucket{le="2.5",} 22.0
requests_latency_seconds_bucket{le="5.0",} 22.0
requests_latency_seconds_bucket{le="7.5",} 22.0
requests_latency_seconds_bucket{le="10.0",} 22.0
requests_latency_seconds_bucket{le="+Inf",} 22.0
requests_latency_seconds_count 22.0
requests_latency_seconds_sum 6.434200000000002E-5
```

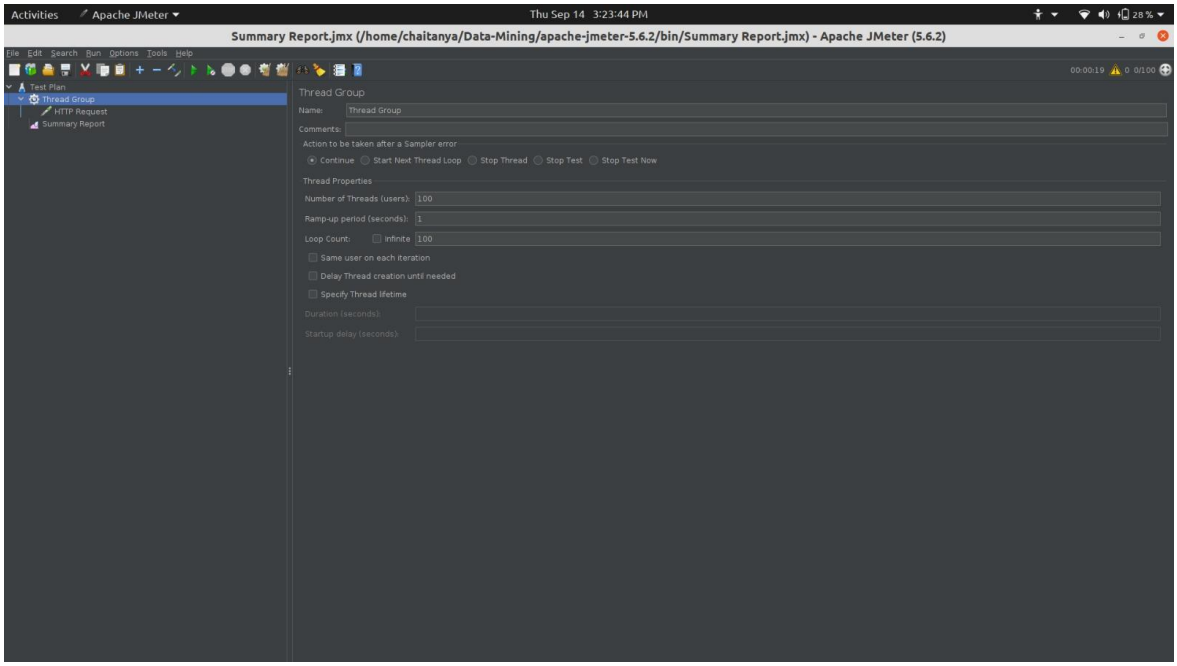
```
requests_total 12.0
# HELP requests_latency_seconds Request latency in seconds
# TYPE requests_latency_seconds histogram
requests_latency_seconds_bucket{le="0.005",} 12.0
requests_latency_seconds_bucket{le="0.01",} 12.0
requests_latency_seconds_bucket{le="0.025",} 12.0
requests_latency_seconds_bucket{le="0.05",} 12.0
requests_latency_seconds_bucket{le="0.075",} 12.0
requests_latency_seconds_bucket{le="0.1",} 12.0
requests_latency_seconds_bucket{le="0.25",} 12.0
requests_latency_seconds_bucket{le="0.5",} 12.0
requests_latency_seconds_bucket{le="0.75",} 12.0
requests_latency_seconds_bucket{le="1.0",} 12.0
requests_latency_seconds_bucket{le="2.5",} 12.0
requests_latency_seconds_bucket{le="5.0",} 12.0
requests_latency_seconds_bucket{le="7.5",} 12.0
requests_latency_seconds_bucket{le="10.0",} 12.0
requests_latency_seconds_bucket{le="+Inf",} 12.0
requests_latency_seconds_count 12.0
requests_latency_seconds_sum 1.8975E-5
```

Here Prometheus has captured different matrics related active Http Sessions, Current memory, free memory, heap memory, Non-heap memory, Garbage Collection run count and time etc. with a histogram showing the latency of requests in different time buckets.

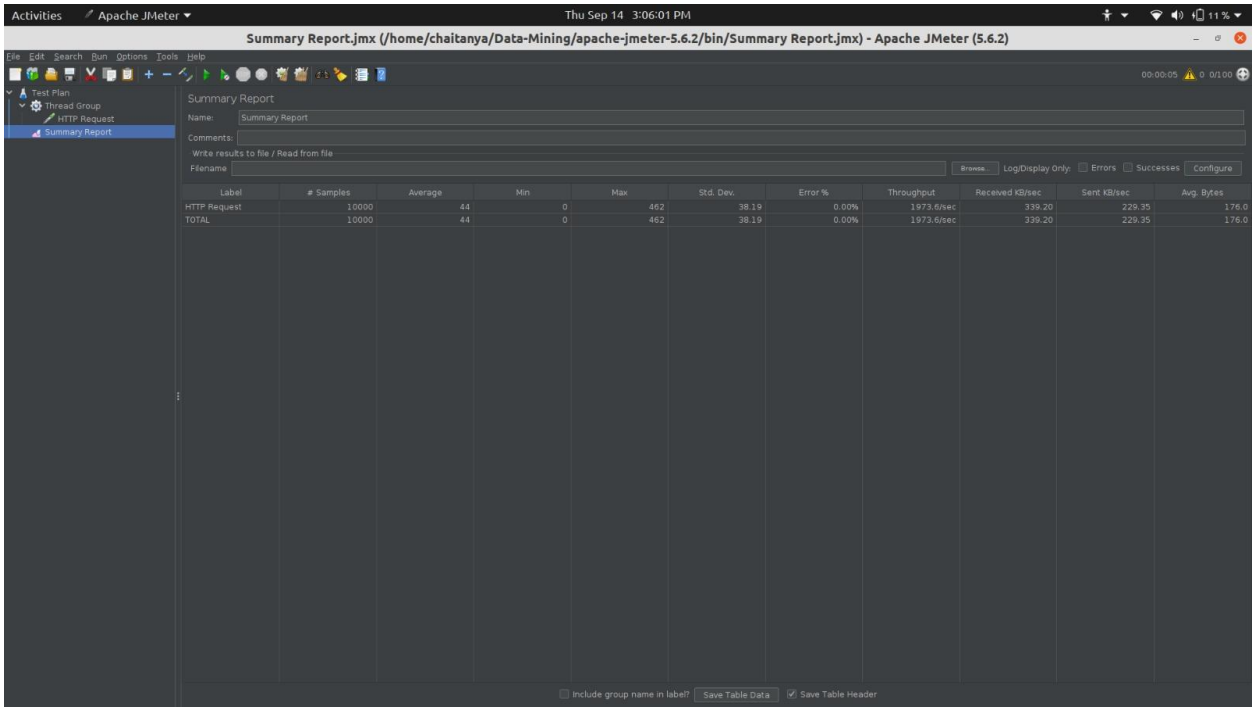
JMeter metrics

Configuration-

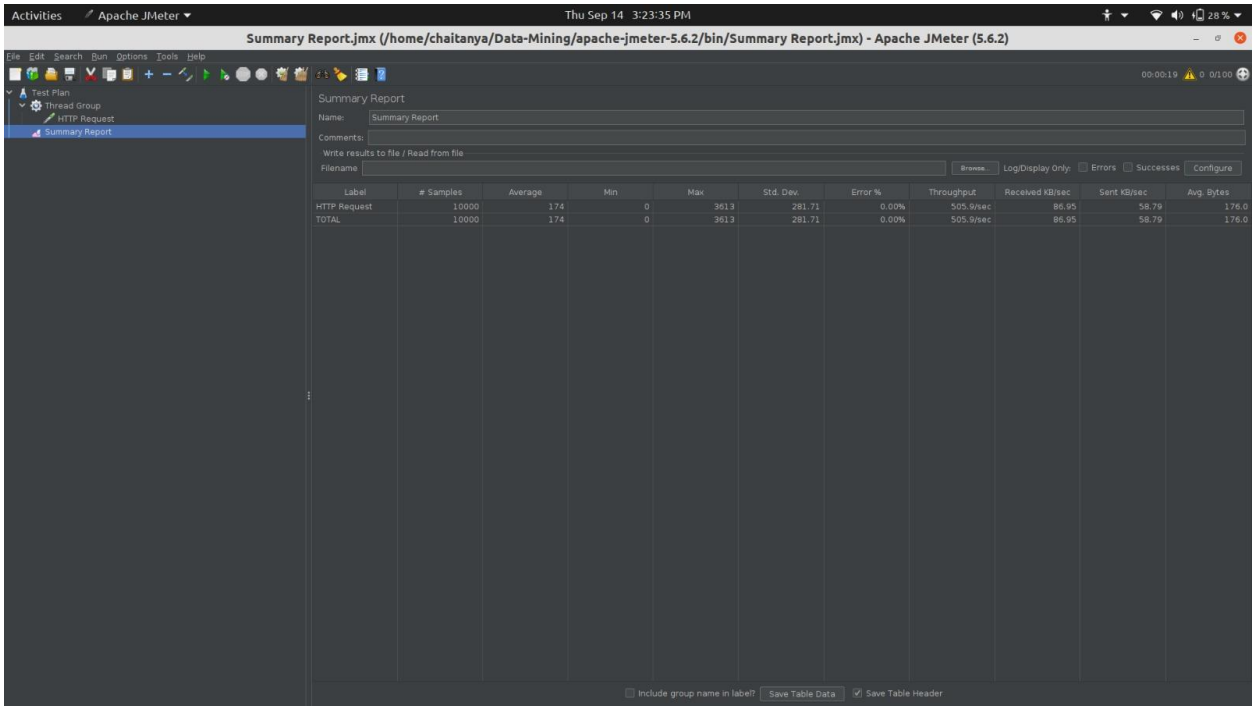




Docker result -

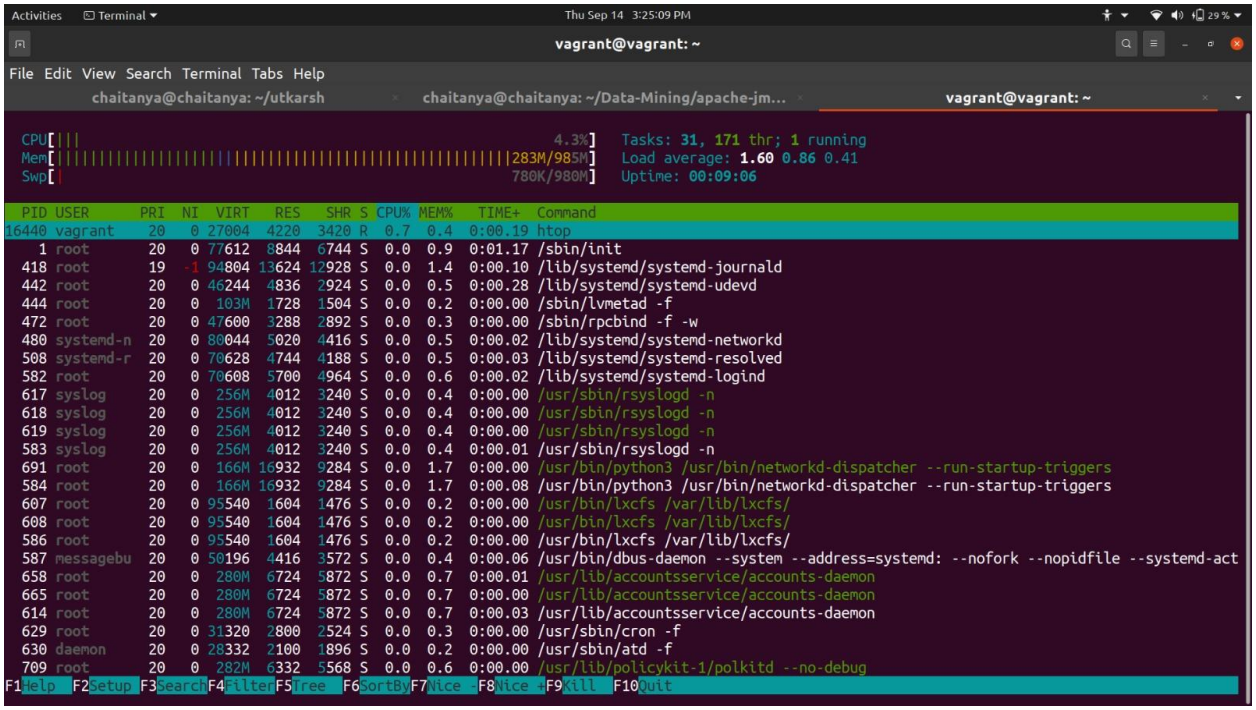


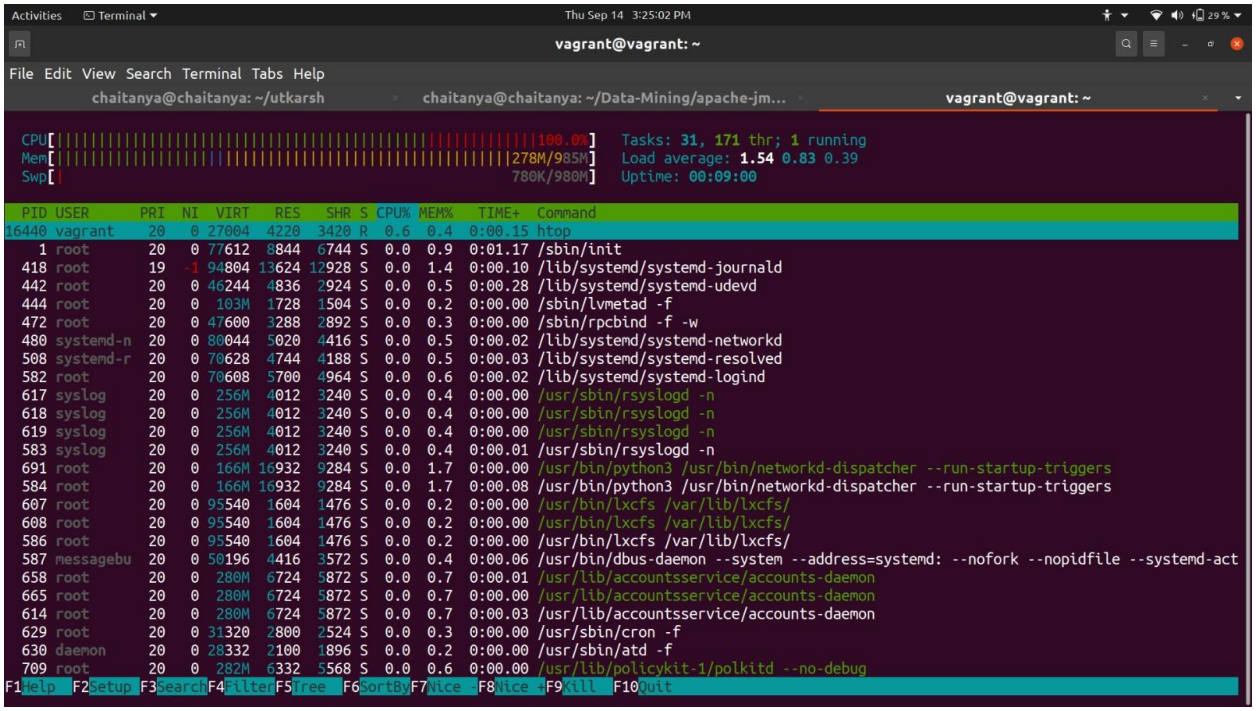
Vagrant result-



Based on the JMeter comparison, Throughput is 1973.6/Sec for the Docker instance whereas it is 505.9/Sec for the Vagrant instance. Apparently, the Throughput is four times higher for the Docker instance compared to the VM instance.

Vagrant resource utilization before and during request -





All the work is available at GitHub repository -

https://github.com/mad-utk/ent-plt-assgnmnt