Java Performance Comparison Dashboard Demo

The goal of this demo is to compare performance of Java applications running in a virtual environment or deployed in containers.

Prerequisites

The following demonstration requires OpenJDK (17), GraalVM (17) and the native image module. You'll also need to install Maven, git, docker and **docker-compose**.

Recommended compute instance would include 8 cores and 64GB of memory.

To access the system (example):

```
$ ssh -i ~username/.ssh/<your-private-key> opc@129.146.21.243
```

> **NOTE:** You'll need to obtain/download the necessary key files to access your system.

Starting the Demo Environment

If the repository hasn't already been cloned, you can access it here:

```
$ git clone https://github.com/swseighman/Java-Perf-Gafana.git
```

Change to the demo directory:

```
$ cd /home/opc/repos/Java-Perf-Gafana/demo
```

Build all of the packages and containers:

```
$ ./build.sh
```

Run `docker-compose` to start all of the services:

```
$ docker-compose up
```

Demo Apps

A simple `primes` demo has been provided but other applications can be added.

> **NOTE:** The `primes` demo has been compiled using Java 17, make certain you're using Java 17.

The `primes` demo produces data via `spring-actuator` (see source code) and is consumed by Prometheus. The app runs on port **8080**. Once started, you should begin to see data in the Grafana dashboard.

Running Benchmarks

To help generate real-time data, `hey` has been installed so that you can run benchmark tests. A script has been provided to start the benchmark tests:

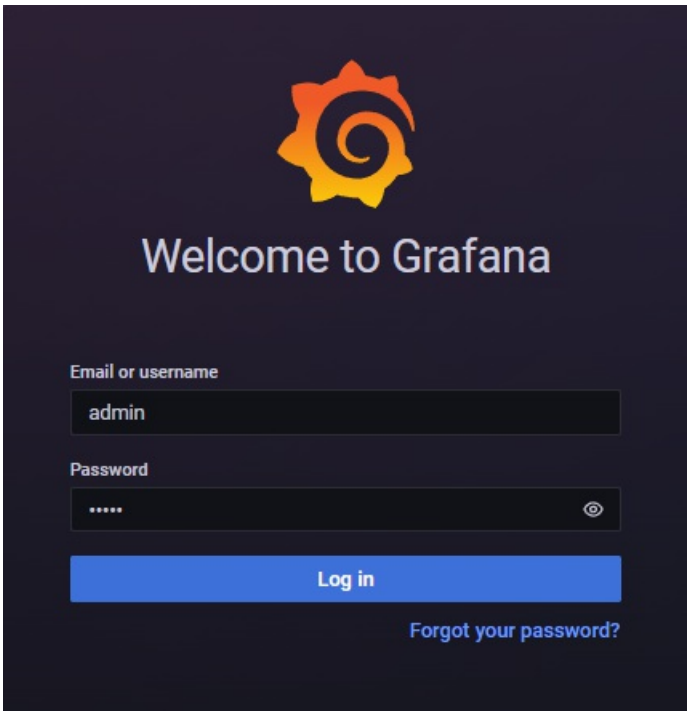
```
$ ./stress.sh
```

To stop all of the services execute:

```
$ docker-compose stop
```

Accessing the Grafana Dashboard

To access the Grafana dashboard (with application data), browse to: **http://129.146.21.243:3000/login**

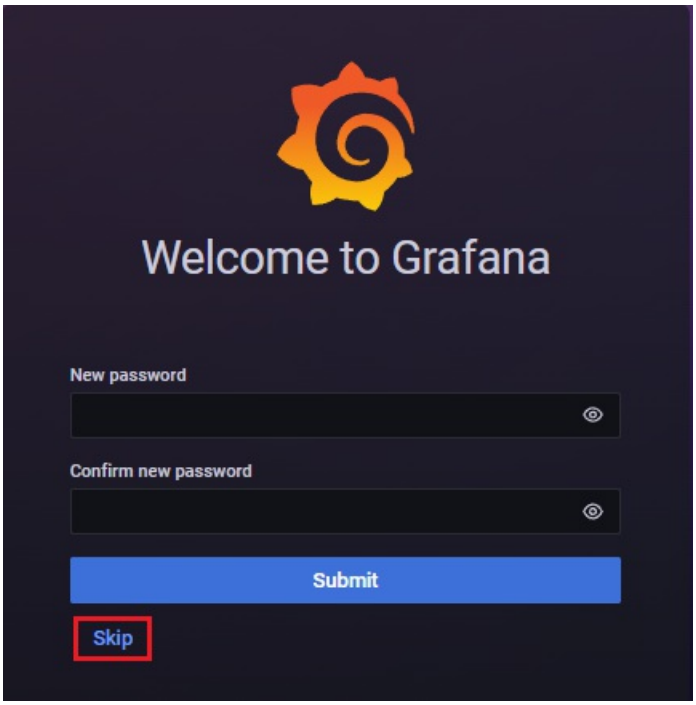


Credentials:

Login: admin

Password: admin

You can **Skip** changing the admin password:



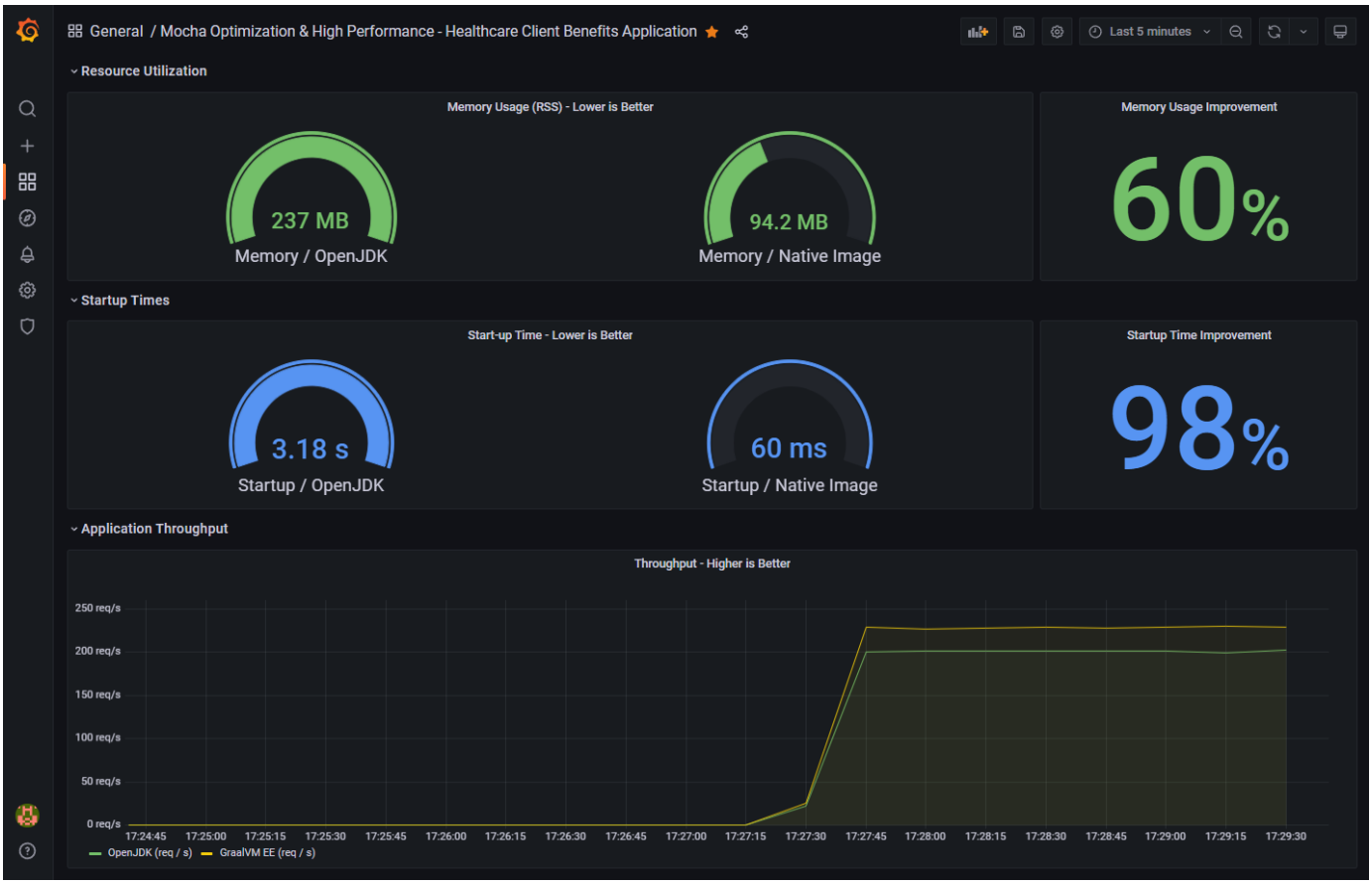
By default, the **Optimization and High Performance** dashboard will be displayed which includes a collection of metrics scraped from:

- Prometheus
- Node (system metrics)
- Cadvisor (containers)
- Spring-actuator (demo app)

This first graph represents memory utilization, the second reflects application startup time. The far-right graphs show the percentage improve of resources (memory) and startup time.

The OpenJDK graph on the left represents an optimized application while native image (on the right) represents the high-performant AOT application.

The last graph displays a comparison of application throughput.


General / Mocha Optimization & High Performance - Healthcare Client Benefits Application
Last 5 minutes
Resource Utilization
Memory Usage (RSS) - Lower is Better
237 MB
Memory / OpenJDK
94.2 MB
Memory / Native Image
Memory Usage Improvement
60%
Startup Times
Start-up Time - Lower is Better
3.18 s
Startup / OpenJDK
60 ms
Startup / Native Image
Startup Time Improvement
98%
Application Throughput
Throughput - Higher is Better
250 req/s
200 req/s
150 req/s
100 req/s
50 req/s
0 req/s
17:24:45
17:25:00
17:25:15
17:25:30
17:25:45
17:26:00
17:26:15
17:26:30
17:26:45
17:27:00
17:27:15
17:27:30
17:27:45
17:28:00
17:28:15
17:28:30
17:28:45
17:29:00
17:29:15
17:29:30
OpenJDK (req / s)
GraalVM EE (req / s)