ASSIGNMENT 1

Part 1

Comparison of AWS, Google, Azure Serverless Options:

1. **AWS Lambda**

1) AWS Lambda is serverless computing that provides the ability to run code without managing servers. With AWS Lambda, applications can be scaled by running the code automatically in response to the trigger events such as HTTP requests, data changes, or state transitions.
2) Event-driven: It supports over 200 AWS services as event sources that can trigger Lambda, which includes S3, DynamoDB, API Gateway, and more.
3) Multilanguage Support: Node.js, Python, Java, Go,.NET, Ruby and custom runtimes.
4) Fine-grained controls: Pay only for the consumed compute time, and Lambda functions can scale themselves to handle high loads.
5) Integrations: Tightly integrated with other AWS services, such as S3, RDS, and DynamoDB, making it an extremely natural choice for those already on AWS.
6) Pros: Tight integration with AWS, strong event-driven functions, highly scalable.
7) Cons: This might be too complicated for new users with all options and integrations.

2. **Google Cloud** – Google Cloud Functions

1) Google Cloud Functions is a Google serverless execution environment that runs the code in response to events from Google Cloud or other systems.
2) Automatic Scaling: Automatically scales with demand, and no provisioning is required.
3) Event-driven: Integrated with Google services (Firestore, Pub/Sub, Cloud Storage) and third-party services like Twilio.
4) Language Support: Node.js, Python, Go, Java, Ruby, and .NET.
5) Firestore and Firebase Integration: Especially great for developers into both mobile and web application development using Firebase, it can provide very tightly integrated serverless backends.

6) Pros: The system works perfectly with most of the Google services, including Firestore and Firebase. It is also very easy to set up.
7) CONS: It doesn't have that much usage outside of Google's ecosystem; it has relatively fewer triggers compared to AWS Lambda.

3. **Microsoft Azure** – Functions

1) Azure Functions is a serverless compute service from Microsoft that allows you to run event-driven code without explicitly provisioning or managing infrastructure.
2) Multitype Triggers: Can use an HTTP trigger, a queue, event, timer, and integration with Azure services.
3) Extensive: it supports languages like C#, JavaScript, Python, PowerShell, TypeScript, and Java.
4) Durable Functions: These allow for the orchestration of stateful functions, thus chaining together several serverless functions.
5) Integration of Azure Services: Tightly integrated with the Azure ecosystem, comprising Azure Event Hub, Blob Storage, and Cosmos DB.
6) Pros: Tight integration with Microsoft's Azure services and development tools such as Visual Studio; immensely powerful Durable Functions feature.
7) Cons: It is less mature than AWS Lambda in many ways, and there are fewer sources of events available than with AWS.

Deep Dive into AWS Lambda

AWS Lambda has developed a lot over the last five years:

Year	Feature	Description
2018	Custom Runtimes Support	Enabled the use of any runtime or programming language not natively supported by AWS, broadening the application range.
	Lambda Layers	Introduced Layers for code reusability and better management of common dependencies across multiple functions.
	Provisioned Concurrency	Allowed functions to stay "warm" to reduce cold start time and improve response times.

2019	Amazon EventBridge	Provided a hub for event buses and advanced filtering and routing for serverless functions.
2020	Lambda Container Support	Added support for packaging functions into container images up to 10 GB, overcoming ZIP archive size limits and supporting larger application packages.
	EFS Integration	Enabled Lambda to mount Amazon EFS for applications requiring persistent and shared storage, extending Lambda to stateful applications.
2021	Graviton2 Support	Introduced support for Graviton2 processors, offering up to 34% better price/performance compared to x86-based instances, improving cost and performance.
	Extension Support	Allowed the addition of monitoring, security, and observability capabilities to functions without altering business logic, with integrations for tools like Datadog and CloudWatch.
2022	SnapStart	Reduced cold start latency for Java functions through early environment initialization and snapshot caching.
	AWS SAM and CDK Improvements	Updated SAM and CDK for better runtime priorities, streamlining infrastructure as code processes for rapid deployment and management of Lambda functions.
2023	Edge Lambda Enhancements	Extended Lambda@Edge capabilities for running serverless functions closer to users, reducing latency in globally distributed applications.
	Better Monitoring and Observability	Enhanced integration of third-party and AWS-native monitoring tools through Lambda Extensions, improving observability and performance tracking.
	Advanced Integration of Storage	Expanded EFS support to cover a broader range of use cases requiring persistent storage, including large-scale machine learning and stateful applications.

Suggested Feature: Enhanced Local Development Environment for AWS Lambda

I suggest adding an **Enhanced Local Development Environment** to AWS Lambda. This would let developers test and debug their Lambda functions on their own machines before deploying them to the cloud.

It is useful because of the following reasons:

1. **Faster Development:**
    - **Current Issue**: Developers spend a lot of time debugging and testing. According to Stack Overflow (2023), this takes up 20-30% of their time.
    - **Benefit**: Testing locally means issues can be fixed faster, speeding up development.
2. **Cost Savings:**
    - **Current Issue**: Testing in the cloud can be expensive, with costs for requests and usage adding up (AWS Pricing, 2024).
    - **Benefit**: Local testing can cut down on these costs.
3. **Better Debugging:**
    - **Current Issue**: Debugging tools for Lambda are limited and can be hard to use (AWS Survey, 2023).
    - **Benefit**: Local tools can provide better error tracking and integration with popular development tools.
4. **Complex Testing Made Easier:**
    - **Current Issue**: Testing apps that use multiple AWS services and APIs can be complicated and slow.
    - **Benefit**: A local environment can simulate these services more accurately, making testing easier.
5. **Supports Modern Practices:**
    - **Current Issue**: Efficient CI/CD pipelines are needed for smooth development (DevOps Survey, 2023).
    - **Benefit**: Local development tools fit well with CI/CD practices, helping with faster and smoother deployments.

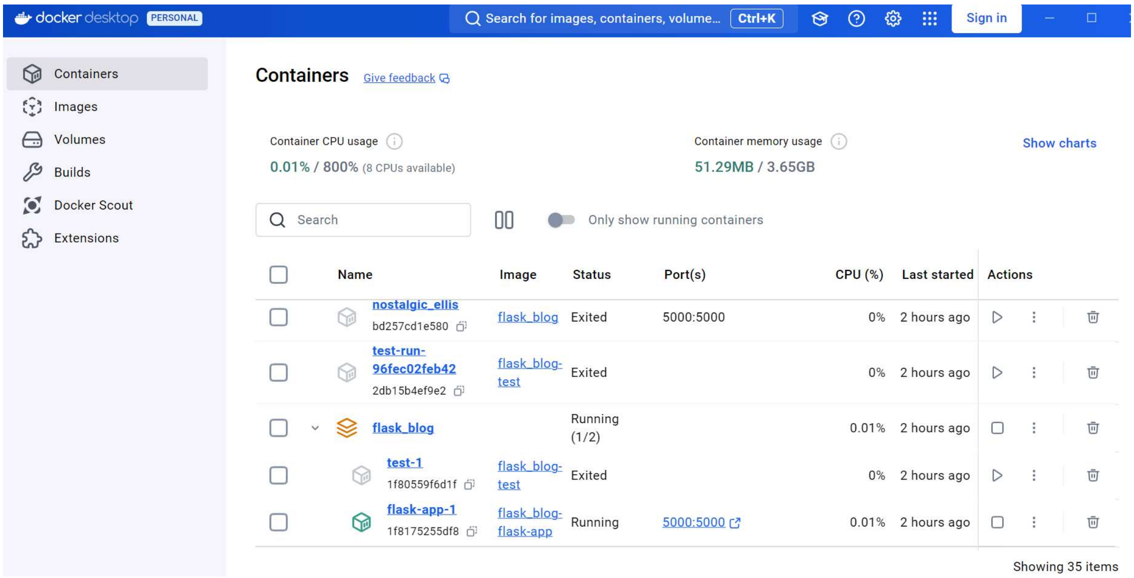
Part 2

Created a flask app for posting, editing and deleting blogs.

Running the app in docker container:

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> docker run -p 5000:5000 flask_blog
 * Serving Flask app 'app.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

Following is visible in docker desktop application:

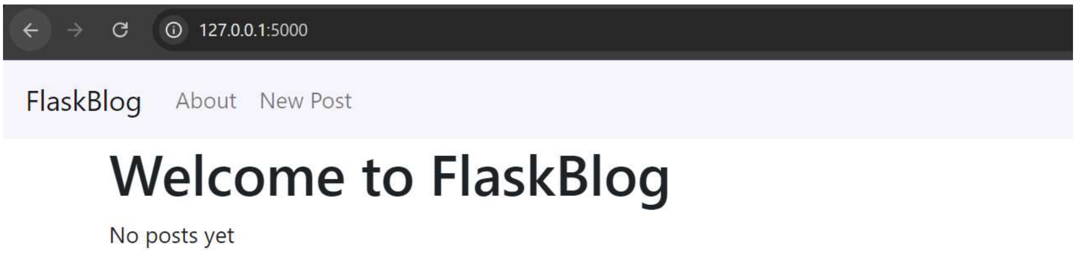


Running unit tests for the flask app:

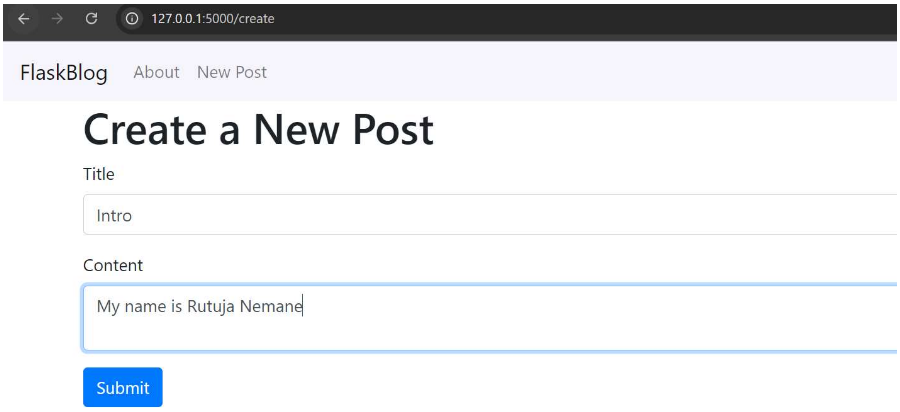
```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> docker-compose run test
time="2024-09-18T20:56:42-07:00" level=warning msg="D:\\Documents\\MS\\SJSU\\272\\Assignment 1\\docker\\flask_blog\\docker-compose.ym
l: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
time="2024-09-18T20:56:42-07:00" level=warning msg="Found orphan containers ([flask_blog-test-run-10d619f6a868 flask_blog-test-run-46
f7ad47ca0a flask_blog-test-run-02a8659840fd flask_blog-test-run-3f3b1077f80f flask_blog-test-run-e066a3256487 flask_blog-test-run-87d
5c8438890 flask_blog-test-run-f3f471fc5227 flask_blog-test-run-e011160af221 flask_blog-test-run-134d406d3575 flask_blog-test-run-af78
b2469fc2 flask_blog-test-run-407e6549b841 flask_blog-test-run-e90bc2040e1d]) for this project. If you removed or renamed this service
 in your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Creating 1/1
 ✔Container flask_blog-flask-app-1  Recreated                                                                                   0.4s
[+] Running 1/1
 ✔Container flask_blog-flask-app-1  Started                                                                                     0.8s
======================================================= test session starts =======================================================
platform linux -- Python 3.9.20, pytest-8.3.3, pluggy-1.5.0
rootdir: /app
collected 4 items

tests/test_routes.py ....                                                                                                      [100%]

======================================================== 4 passed in 2.80s ========================================================
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog>
```

Web page screenshots for each case:

Index page:

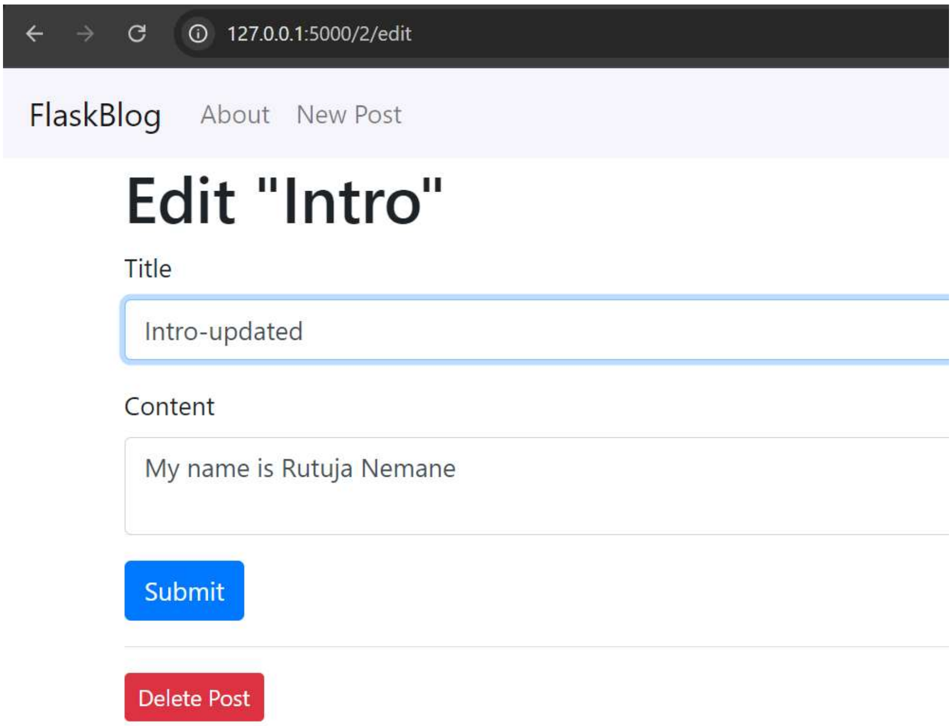


Create post:

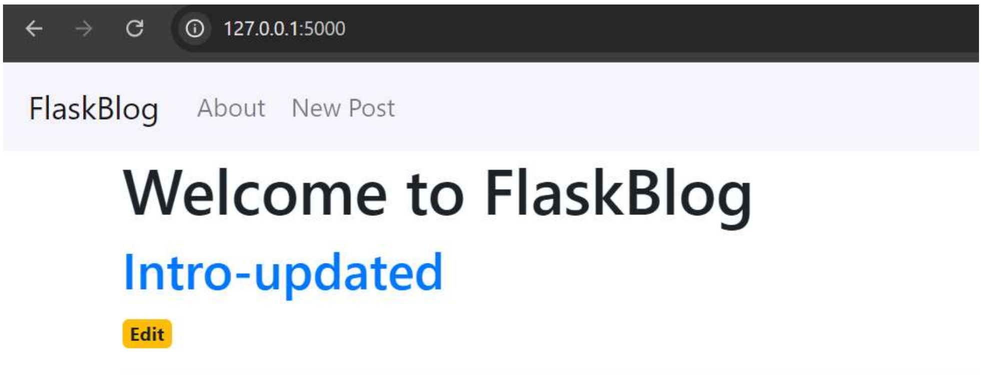


Post created:

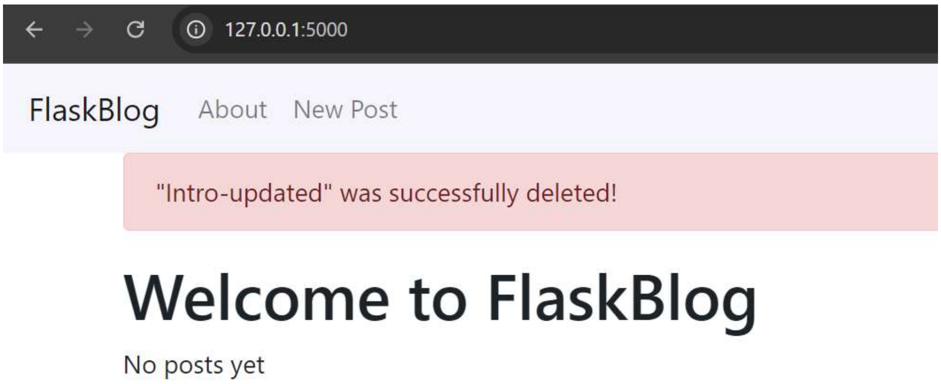
Edit post:



Post updated:



Delete post:



Part 3

Running flask app in Virtual Box using Vagrant:

Running flask app in docker container:

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> docker run -p 5000:5000 flask_blog
 * Serving Flask app 'app.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

Metrics

1. **Startup Time**:

Vagrant:

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> Measure-Command {vagrant up}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 8
Milliseconds      : 225
Ticks             : 82254685
TotalDays         : 9.5202181712963E-05
TotalHours        : 0.00228485236111111
TotalMinutes      : 0.137091141666667
TotalSeconds      : 8.2254685
TotalMilliseconds : 8225.4685
```

Docker container:

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> Measure-Command {docker-compose up}
time="2024-09-18T17:41:54-07:00" level=warning msg="Found orphan containers ([flask_blog-test-run-46f7ad47ca0a flask_blog-test-run-02
a8659840fd flask_blog-test-run-3f3b1077f80f flask_blog-test-run-e066a3256487 flask_blog-test-run-87d5c8438890 flask_blog-test-run-f3f
471fc5227 flask_blog-test-run-e011160af221 flask_blog-test-run-134d406d3575 flask_blog-test-run-af78b2469fc2 flask_blog-test-run-407e
6549b841 flask_blog-test-run-e90bc2040e1d]) for this project. If you removed or renamed this service in your compose file, you can ru
n this command with the --remove-orphans flag to clean it up."
[+] Running 2/2
 ✔Container flask_blog-flask-app-1  Running0.0s
 ✔Container flask_blog-test-1       Created0.2s
```

This shows that docker container has faster startup time than Vagrant.

2. Memory Usage

Vagrant:

```
vagrant@ubuntu-focal:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:            964         136         132           0         696         668
Swap:             0           0           0
```

Docker container:

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> docker stats
CONTAINER ID   NAME                   CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O   PIDS
07dc097b9749   flask_blog-flask-app-1 0.05%     28.92MiB / 3.736GiB   0.76%     1.05kB / 0B     0B / 0B     1
```

3. CPU Utilization

Vagrant: Using htop

```
  1  [||                                              1.3%]   Tasks: 29, 39 thr; 1 running
  2  [|                                               0.7%]   Load average: 0.00 0.00 0.00
  Mem[||||||||||||||||||||||||||||||||||||||||||||138M/965M]   Uptime: 00:44:34
  Swp[                                               0K/0K]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   8776 vagrant    20   0  8240  3824  3064 R  1.3  0.4  0:00.09 htop
```

Docker container: docker stats

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker\flask_blog> docker stats
CONTAINER ID   NAME                   CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O   PIDS
07dc097b9749   flask_blog-flask-app-1 0.05%     28.92MiB / 3.736GiB   0.76%     1.05kB / 0B     0B / 0B     1
```

4. **Request Throughput and Response Times**

Vagrant: Using ApacheBench

```
vagrant@ubuntu-focal:~$ ab -n 1000 -c 100 http://127.0.0.1:5000/
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Werkzeug/3.0.4
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /
Document Length:        2086 bytes

Concurrency Level:      100
Time taken for tests:   5.430 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      2261000 bytes
HTML transferred:       2086000 bytes
Requests per second:    184.15 [#/sec] (mean)
Time per request:       543.037 [ms] (mean)
Time per request:       5.430 [ms] (mean, across all concurrent requests)
Transfer rate:          406.60 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   4.9      0      30
Processing:    42  478  78.8    495     567
Waiting:       25  465  79.2    482     550
Total:         71  480  74.3    495     567

Percentage of the requests served within a certain time (ms)
  50%    495
  66%    507
  75%    515
  80%    520
  90%    533
  95%    540
  98%    547
  99%    552
 100%    567 (longest request)
```

Docker: Using locust script

```
PS D:\Documents\MS\SJSU\272\Assignment 1\docker> locust -f locustfile.py --host=http://localhost:5000
[2024-09-18 18:24:27,524] DESKTOP-8U3992S/INFO/locust.main: Starting web interface at http://localhost:8089 (accepting connections fr
om all network interfaces)
[2024-09-18 18:24:27,583] DESKTOP-8U3992S/INFO/locust.main: Starting Locust 2.31.6
[2024-09-18 18:29:20,169] DESKTOP-8U3992S/INFO/locust.runners: Ramping to 10 users at a rate of 1.00 per second
[2024-09-18 18:29:29,176] DESKTOP-8U3992S/INFO/locust.runners: All users spawned: {"WebsiteUser": 10} (10 total users)
[2024-09-18 18:30:35,310] DESKTOP-8U3992S/INFO/locust.runners: Ramping to 10 users at a rate of 1.00 per second
[2024-09-18 18:31:28,164] DESKTOP-8U3992S/INFO/locust.runners: Ramping to 10 users at a rate of 1.00 per second
[2024-09-18 18:31:37,164] DESKTOP-8U3992S/INFO/locust.runners: All users spawned: {"WebsiteUser": 10} (10 total users)
```

Locust webpage where I ran a test with 10 users for 10 secs.

LOCUST | HOST http://localhost:5000 | STATUS STOPPED | RPS 164.83 | FAILURES 0% | NEW | RESET

STATISTICS | CHARTS | FAILURES | EXCEPTIONS | CURRENT RATIO | DOWNLOAD DATA | LOGS

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	1428	0	22	41	89	24.62	9	144	2086	164.83	0
	Aggregated	1428	0	22	41	89	24.62	9	144	2086	164.83	0

5. Garbage Collection and profiling

Using python "**gc**" module for garbage collection and printing the results on a webpage.

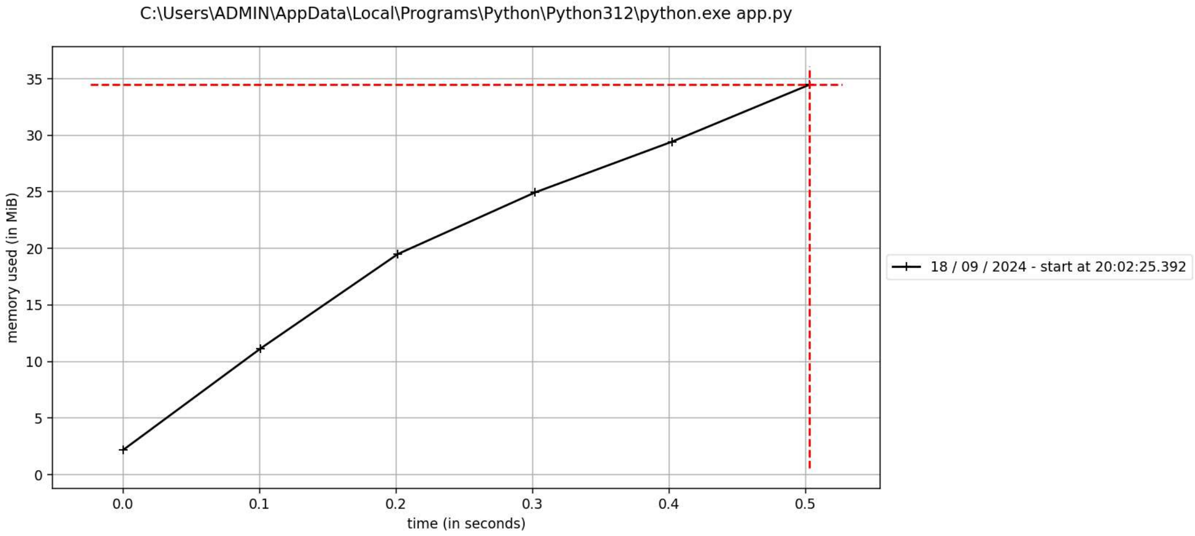
localhost:5000/gc-stats

Garbage collection stats: [{'collections': 75, 'collected': 391, 'uncollectable': 0}, {'collections': 6, 'collected': 9, 'uncollectable': 0}, {'collections': 2, 'collected': 11, 'uncollectable': 0}]

Using **memory profiler** for plotting graph of memory utilization:

Commands used:

mprof run app.py
mprof plot


C:\Users\ADMIN\AppData\Local\Programs\Python\Python312\python.exe app.py
memory used (in MiB)
time (in seconds)
0
5
10
15
20
25
30
35
0.0
0.1
0.2
0.3
0.4
0.5
18 / 09 / 2024 - start at 20:02:25.392