SOCAlgoTestPlatform Hosting Cost Estimate

Part 1 - Resource Requirements

The resources required to host the system comprise 5 main components. A main VM that will run the web server, a cluster of background workers, a database server, a redis server, and block storage. In the following section, we will use some simplifying assumptions to estimate the resource requirements to host the system for a few scenarios. We will always err on the side of overprovisioning the system rather than underprovisioning, to ensure system stability and performance.

Scenarios:

As the system is in a development stage, and we do not have any system load statistics, we will outline a few potential scenarios regarding system load. Each scenario will define a few important assumptions: average requests per second, average number of models submitted per hour. From this, we will determine the number of workers required to run the web server, the number of workers required to run the submissions, and how many completed submissions we will need to store per day.

Scenario 1:

The first scenario is the system is being used by a single engineering class over the course of a semester. For this we can gather roughly 150 active users, all located in the same timezone, possibly using the system in more condensed bursts around deadlines. Due to the potentially condensed nature of the platform usage, we will assume the system will need to handle an average of 50 requests/second for busy periods and that 50 submissions will be made per hour during busy periods. We will calculate the number of completed submissions which we will need to store here, slightly different from the other scenarios, to account for the finite nature of using the system for a single class. If the system is used as part of 4 assignments during a semester, we can assume each user might submit 5 models per assignment, as they fine tune and test. This would result in 4 * 5 * 150 = 3000 total submissions. Each submission requires roughly 2.1 mb of object storage for figures and statistics and 10kb of database storage. From this, we can gather 6300 mb of object storage, and 30000kb of database space will be used for one semester.

Scenario 2:

The second scenario is the system being used by multiple engineering classes, possibly spread over multiple universities. We will assume 10 classes, resulting in 1500 active users. We will remove our previous assumptions about submissions coming in bursts, as with different classes, deadlines are on different days and should result in a more uniform basis. With 1500 active users, we will assume 150 requests/second and 150 submissions per hour. We will again assume these classes are in similar time zones, so the submissions arrive across a 12 hour span. This would result in 1800 submissions per day, or 3780 mb of object storage and 18000kb of database space per day.

Scenario 3:

The third and final scenario will represent (relatively) widespread adoption. We will assume that the application has an active user base of 5000 users, composed of many engineering classes and researchers. We will no longer assume all users are in similar time zones, so requests will now be spread out over a 24-hour period. We will assume 250 requests/second and 250 submissions per hour. This will result in 6000 submissions per day, or 12600 mb of object storage and 60000 kb of database space every day.

System components:

Main VM:

This VM will run the web server. Ideally, the web server would be deployed using Gunicorn. From recommendations in the gunicorn documentation [1] to handle a few hundred requests/second, we would need roughly 4 workers, and to handle a few thousand per second, we would need as many as 12. Scaling a single instance past 12 workers has its own drawbacks, so we will stop our considerations here. From the scenarios defined above, our highest level of traffic is only 150 requests/second. Additionally, the operations of the web server itself are not computationally intense, meaning we will be fine operating with a minimum of 4 workers. Gunicorn recommends operating 2x(number of CPU cores)+1 workers, so if we run the server on a VM with 2 cores, 5 workers would be appropriate. From working experience, 2 GB of RAM will be sufficient for this number of workers, so we can operate the server from a VM with 2 cores and 2 GB of RAM. While we could use a system with a single core for the scenarios with lighter loads, the cost difference will be marginal, so we will use a 2core+2 GB vm to host the web server for all three scenarios.

Background Workers:

The test suite, on average, takes 5-10 minutes to run from start to completion. We will use the higher end of the spectrum for these calculations and assume each submission requires 10

minutes of CPU time to complete. Additionally, each running test requires 1 GB of RAM, so each core requires 1 GB of RAM. We would like to provide some reasonable guarantees about turnaround jobs, a submitted job should not take more than an hour to be processed on average. So, if 15 submissions/hour are made, we need to have enough workers to process all 15 within the hour.

Scenario	Submissions/hour	Computation minutes/hour	Cores+Gb of ram Required
1	50	500	9+9
2	150	1500	27+27
3	250	2500	45+45

Redis Server:

Redis facilitates sending jobs to our celery workers. With 45 workers in our most intense scenario, we have a higher number of connections, but the load on the redis server is relatively low as it is only being used as a message queue. Redis recommends 2 cores and 8 gb of ram in its minimum requirements, so we will use that for all 3 scenarios.

Postgres Server:

Scenario	Data generated/day (kb)	Data generated/year (gb)
1	250	0.09125
2	18000	6.57
3	60000	21.9

Object Storage:

Scenario	Data generated/day (mb)	Data generated/year (gb)
1	52.5	19.16

2	3780	1343
3	12600	4599

(Note that the low value of Data generated/day for Scenario one is due to the unique estimation we performed early due to the usage pattern of a single class.)

Totals:

Scenario	Main vm size (cores + ram)	Background Workers (cores + ram)	Redis Server (cores + ram)	Postgresql storage (1 year) (GB)	Object Storage (1 year) (GB)
1	2+2	9+9	2+8	0.09125	19.16
2	2+2	27+27	2+8	6.57	1343
3	2+2	45+45	2+8	21.9gb	4599

Part 2 - Cost comparisons

AWS

All costs are in USD/month

Scenario	Main VM cost	Background workers cost	Redis cost	Postgres cost	S3 object storage cost	Total cost	Total for 1 year
1	$7.73	$38.65	$32.38	$28.38	$0.46	$107.6	$1291.2
2	$7.73	$107.82	$32.38	$28.38	$30.98	$207.29	$2487.48
3	$7.73	$177.79	$32.38	$28.38	$105.78	$352.06	$4224.72

(Note: the most cost-effective option for Postgres was to deploy it on a separate instance with 50 GB of block storage attached; lowering the amount of block storage did not have a significant impact on the cost)

Hetzner

All costs in USD/month

Scenario	Main VM cost	Background workers cost	Redis cost	Postgres cost	S3 object storage cost	Total cost	Total for 1 year
1	$15.09	$71.18	$15.09	$15.09	$5.99	$122.44	$1469.28
2	$15.09	$222.59	$15.09	$15.09	$16.78	$284.64	$3415.68
3	$15.09	$333.59	$15.09	$15.09	$40.78	$419.64	$5023.68

(Note: For the main vm, redis, and postgres, the smallest available vps is used; this is a 2+8 configuration with 80 GB of SSD storage.)

Part 3 - Other considerations

Comparing the two cost breakdowns, AWS came in under Hetzner for all scenarios, which is somewhat surprising, given the reputation of AWS as a higher-priced provider and Hetzner as a more budget-oriented provider. This is due to the large variety of options provided by AWS, allowing for resources to be precisely provisioned, whereas Hetzner offers fewer size options, resulting in overprovisioning of resources for many of the services needed.

[1] https://docs.gunicorn.org/en/stable/design.htm