# Innometrics programming test

Implement a simple backend application that satisfies the below specifications. You do not need to build any frontend parts, only an API and underlying system.

Specifications

* Your app should be implemented as a web service.

Ans : Start service with main class : com.kyleong.innometrics.server.IMServerApp

* It should expose at least three methods:
  + one to increment a named counter by 1

Ans:

Method : POST

Param : name=xxx

REST API URL:

http://localhost:8080/counter /add, (add 1 to counter with name = xxx, if counter xxx not exist, init value as 1)

http://localhost:8080/counter /addasync(add 1 to counter xxx, in asynchronies request processing queue)

* + one to get the current value of a given counter

Ans :

Method:GET

Param : name=xxx

REST API URL:

http://localhost:8080/counter/get, (get single counter with name = xxx and return code, like count not found)

http://localhost:8080/counter/getjson, (get single counter with name=xxx, if not found return -1 as counter value)

* + one to get a list of all counters and their current value

Ans:

Method: GET

PARAM: N/A

REST API URL:

http://localhost:8080/counter /getall, (get all counter with return code)

http://localhost:8080/counter /getalljson(get counter with pure json result)

Notes

* Try to make the API nice to use for a hypothetical frontend developer.
* It’s ok if the counter loses state on app shutdown (you don’t need to implement persistent storage layer).
* For this test we ask you to use some version of Java, but other than that you are free to choose any libraries, frameworks, etc. you want.
* We recommend using [a framework like Jersey](https://jersey.java.net/documentation/latest/getting-started.html), but you are free to choose other options.
* Please put your code into a public github repository
* If you can also host a running version of the app somewhere (for example Heroku) that would be awesome

Additional questions:

*(you do not need to actually implement support for the below items, just have some ideas for how the app would be changed to support each one. We will discuss them during a subsequent code review session.)*

* Persistence. How would you add a persistent storage layer such that the app could be restarted without losing counter states?

Ans : Using redis or other db as the persistence layer, when service get restarted, reload same counter from redis by key(counter name) when services get restarted

Before shutdown make sure request in asynchronies message queue has been handle properly.

* Concurrency. What happens if your service is handling multiple simultaneous requests? How would you update the app to support many concurrent clients?

Ans:

1. if it is a single node and heavy process for handling single request, Services and write in asynchronous way , have a message queue first accept the request, and have other background service(multi-thread) to processing the request object in the queue;

2. or we can using distributed architecture, e.g. have a nginx as the router of the request, all the worker node(distributed services in different machine) or services when get started register to the nginx route list, nginx can using round robin or other route strategy, like according to current work load of each node by (checking)the current pending queue length of the work nodes to distribute request;

* Fault tolerance. How would the app need to change in order to continue working if some part of the underlying system hardware (disk, memory, CPU, network, etc.) were to fail?

Ans:

1. Using nginx HA(master-slave) solution as router, to avoid single router node failed;

2. using multi-work nodes deploy to in different machines to handle the requests distributed by nginx routers, need to using subscribe and notification mechanism to make sure nginx router aware the status of the worker nodes;

3. using redis cluster as the persistence layer HA solution.

4. need to have n seconds heartbeat connection for most of the router nodes, worker nodes, and redis cluster, to notify each other the healthy status of each other

5. need to have an centralized monitor solution to monitor all the healthy of the node, such as cpu, mem, disk, network usage, report alarm when in the threshold or using HA solution such as symantec to monitor and apply switch over strategy;

6. IP groups need to be allocated properly, also need to have system internal network with different sub-net and gateway;

* Scalability. How would the app need to change in order to ensure that it wouldn’t break down if traffic increased by many orders of magnitude?

Ans:

1. all the resources should implement in stateless way(no status), don't depends on single node memory or object, instead using the data store in memcache layer or redis;

2. according to different bottle neck of the entire system by monitor tools, add new nodes or workers in different layers, such as router layer, worker node(services layer), and persistence layer(redis cluster)

3. divide single service to multi-small services(sub system) of the solution, different then scala-out homogeneous services(same resources or rest api functions) with multi-instances, but heterogeneous services(sub-system, different functions api), which also can have multi-instance for every type of them

3. communication between heterogeneous services(sub-system) also can go through services router(requests router, nginx), only have different Authentication strategy with external requests;

* Authentication. How would the app need to change in order to ensure that only authorised users can submit and retrieve data?

Ans: implement authentication in filter or servlet or AOP

1. For external request(user request) we can using normal authentication solution, such as request login process, after user login generate an token(n minutes expired), every user request should be send with an validated token, token expire information also can using redis KV expire mechanism, for every new request update the token expire time

2. have a separated authentication services other than business services, new requests or user login , token validation should be handle by this authentication service;

3. if there is api calls from external, we can using OAuth 2.0 as a developer level authentication, or internal we can using LDAP services

Other optimize work can be done:

1. write more test case with Jersey client

2. add more comments to the code