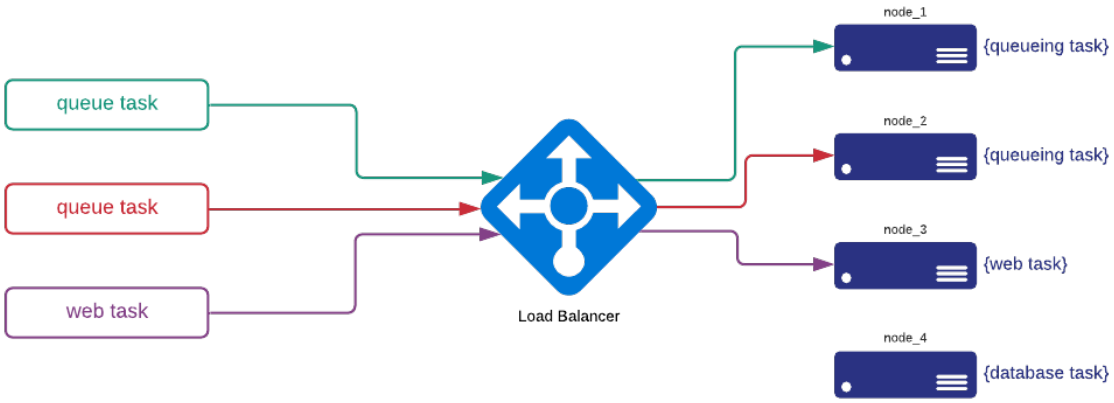
Home Assignment- Load Balancer

General instructions - Please read before you start

- The assignment consists of several checkpoints which rely on each other. Therefore, **it is recommended to read the entire assignment prior to starting**.
- Using a high-level programming language (Java/C#/Python) is recommended.
- Your solution should be submitted via GitHub to a repository given to you. **If you do not have a repository name, please contact us and we will provide you with one**.
- You are encouraged to **use Git best practices** throughout your development, e.g., have each one of the checkpoints committed to their own branch.
- Following to submitting the assignment, you will review it together with the interviewer. During the interview **you might be asked to introduce new features to your solution**.

Objective – Creating a load balancer

Your assignment is to design and implement a load balancer distributing incoming tasks to nodes. Given a list of nodes, each capable of running specific types of tasks, and a list of tasks, the load balancer will decide which node will be assigned to which task.



Example of task dispatching of specific tasks to nodes that can run them.

Input

Your program will receive the following command-line arguments:

1. Path to a JSON file, representing the nodes configuration:

```
{
  "nodes": [
    { "node_name": "node_1", "accepted_task_type": "queueing_task" },
    { "node_name": "node_2", "accepted_task_type": "queueing_task" },
    { "node_name": "node_3", "accepted_task_type": "web_task" },
    { "node_name": "node_4", "accepted_task_type": "database_task" }
  ]
}
```

As we can see, the file consists of a single object with a list field named "nodes". Each element in the list represents a single node and contains two fields:

a. **node_name** – The name of the node
b. **accepted_task_type** – The type of task that the node can execute.

In the example above we have 4 nodes that can execute 3 types of tasks: Both node_1 and node_2 can perform a queueing_task, node_3 can perform a web_task, and node_4 can perform a database_task

You can safely assume that:

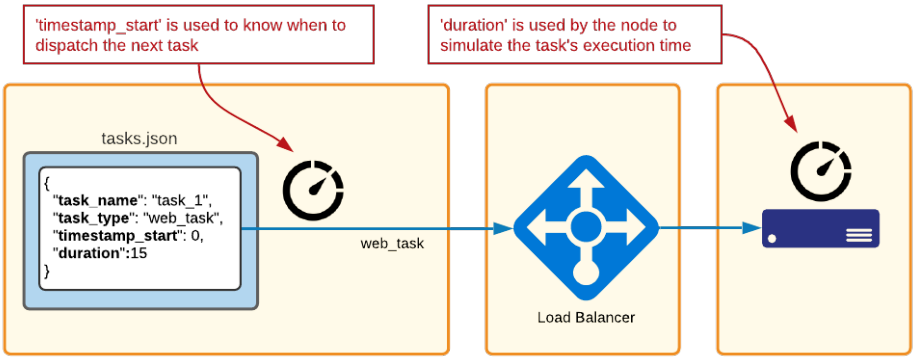
- **Each node supports a single task type.**
- **Each node can receive multiple tasks but can execute only a single task at a given time. If a node receives a new task when it already has one running, the new task will wait and will only be executed after the previous one finished running.**

2. Path to a JSON file, representing the incoming tasks execution order:

```
{ "task_name": "task_1", "task_type": "queueing_task", "timestamp_start": 0, "duration": 15 }
{ "task_name": "task_2", "task_type": "web_task",      "timestamp_start": 0, "duration": 10 }
{ "task_name": "task_3", "task_type": "queueing_task", "timestamp_start": 1, "duration": 3  }
{ "task_name": "task_4", "task_type": "queueing_task", "timestamp_start": 5, "duration": 3  }
```

Unlike the nodes configuration file, the tasks file consist of multiple objects, single object per line. Each such object contains the following fields:

a. **task_name** – The name of the task
b. **task_type** – The type of the tasks, to be matched against the nodes '**accepted_task_type**' field.
c. **timestamp_start** – Timestamp representing the task arrival time to the load-balancer**.** Given two tasks, one with *timestamp_start=0* and the other *timestamp_start=5*, the load balancer will receive the first task, and then will need to wait 5 seconds before sending the second task.
d. **duration** – The time that the task will be executed in the node. We have two different fields that used to calculate the passage of time – the '**timestamp_start**' and the '**duration**'. While the '**timestamp_start**' represents the timestamp where the task was dispatched to the load balancer (time passed *before* the task reached the load balancer), the '**duration**' time represents the time that the task took to execute in the node itself (time *after* the task arrived to the load balancer from it to the routed node). Both should be used for delays only.



The tasks input file can be very big, bigger than what can be fit into memory.

You can assume that '`timestamp_start`' is non-decreasing.

OUTPUT

The program's output should be two log lines per executed task:

- A log line when a task started, in the format:
  ```
  Started task='<task_name>' of type='<task_type>' on node='<node_name>' at '<task_start_time>'
  ```
- A log line when a task finished, in the format:
  ```
  Finished task='<task_name>' of type='<task_type>' on node='<node_name>' at '<task_finished_time>'
  ```
  where `task_finished_time` equals to `task_start_time + duration`.

NOTES ABOUT DESIGN AND IMPLEMENTATION

In this assignment we implement three subsystems:

1. The Load-Balancer.
2. A subsystem which feeds the tasks to the load-balancer.
3. A subsystem that simulates the task's execution on the node.

The following guidelines should apply to each one of the subsystems:

- **Well-structured**, **organized**, and **documented** (code and usage). It should be **maintainable**, and **reliable**, addressing **unexpected behaviors and corner cases** (e.g., missing or invalid user input).
- **Testable**, and have a basic testing suite.
- **Extensible**, e.g., providing different input or output sources. Remember that you might be asked to add additional features in the interview itself.
- **Memory efficient** - It should support hundreds of configured nodes, thousands of tasks types, and unbounded number of tasks.
- **Debugged and run easily**.
- It is recommended that you make sure that **each checkpoint is "production-ready"** prior to moving to the next checkpoint. **It is better to not finish all the checkpoints rather to finish them all with lower quality**.
- In case that you extend your solution beyond the stated requirements, please document what features were added.

And finally, you are **encouraged to use advanced language features and standard 1st/3rd party libraries**. There is no reason to reinvent the wheel, even in an interview assignment. If you have a doubt regarding using a specific library, don't hesitate to ask us.

CHECKPOINT I

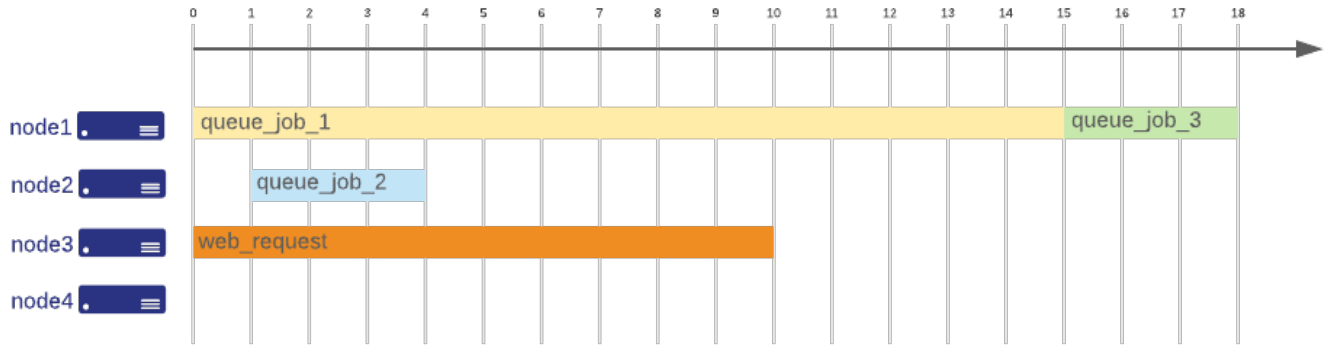
In this checkpoint we need to implement our load-balancer with a simple round-robin logic: Given a task, the load-balancer will round-robin between all the nodes that support it. Given the following nodes and task files:

```
{
  "nodes": [
    { "node_name": "node_1", "accepted_task_type": "queueing_task" },
    { "node_name": "node_2", "accepted_task_type": "queueing_task" },
    { "node_name": "node_3", "accepted_task_type": "web_task" },
    { "node_name": "node_4", "accepted_task_type": "database_task" }
  ]
}
```

```
{ "task_name": "queue_job_1", "task_type": "queueing_task", "timestamp_start": 0, "duration": 15 }
{ "task_name": "web_request", "task_type": "web_task",      "timestamp_start": 0, "duration": 10 }
{ "task_name": "queue_job_2", "task_type": "queueing_task", "timestamp_start": 1, "duration": 3  }
{ "task_name": "queue_job_3", "task_type": "queueing_task", "timestamp_start": 5, "duration": 3  }
```

The expected output will be:

```
Started task='queue_job_1' of type='queueing_task' on node='node_1' at '0'
Started task='web_request' of type='web_task' on node='node_3' at '0'
Started task='queue_job_2' of type='queueing_task' on node='node_2' at '1'
Finished task='queue_job_2' of type='queueing_task' on node='node_2' at '4'
Finished task='web_request' of type='web_task' on node='node_3' at '10'
Finished task='queue_job_1' of type='queueing_task' on node='node_1' at '15'
Started task='queue_job_3' of type='queueing_task' on node='node_1' at '15'
Finished task='queue_job_3' of type='queueing_task' on node='node_1' at '18'
```



Since both 'queue_job_1' and 'web_request' start at the same time, it does not matter which one your program will log first.

CHECKPOINT II

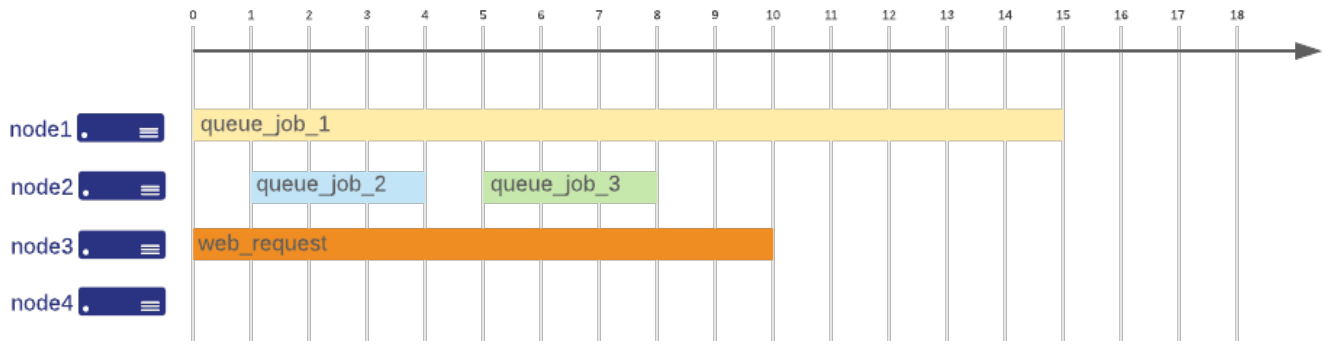
In this checkpoint we will extend our load-balancer with an additional mode, in which nodes with less tasks in their queue are elected first. For the input:

```
{
  "nodes": [
    { "node_name": "node_1", "accepted_task_type": "queueing_task" },
    { "node_name": "node_2", "accepted_task_type": "queueing_task" },
    { "node_name": "node_3", "accepted_task_type": "web_task" },
    { "node_name": "node_4", "accepted_task_type": "database_task" }
  ]
}
```

```
{ "task_name": "queue_job_1", "task_type": "queueing_task", "timestamp_start": 0, "duration": 15 }
{ "task_name": "web_request", "task_type": "web_task",      "timestamp_start": 0, "duration": 10 }
{ "task_name": "queue_job_2", "task_type": "queueing_task", "timestamp_start": 1, "duration": 3  }
{ "task_name": "queue_job_3", "task_type": "queueing_task", "timestamp_start": 5, "duration": 3  }
```

The expected output will be:

```
Started task='queue_job_1' of type='queueing_task' on node='node_1' at '0'
Started task='web_request' of type='web_task' on node='node_3' at '0'
Started task='queue_job_2' of type='queueing_task' on node='node_2' at '1'
Finished task='queue_job_2' of type='queueing_task' on node='node_2' at '4'
Started task='queue_job_3' of type='queueing_task' on node='node_1' at '5'
Finished task='queue_job_3' of type='queueing_task' on node='node_1' at '8'
Finished task='web_request' of type='web_task' on node='node_3' at '10'
Finished task='queue_job_1' of type='queueing_task' on node='node_1' at '15'
```



We will add another command-line argument, which will be the logic that the load-balancer will follow. The possible values are '**`round-robin`**', which will follow the same round robin logic that we've implemented in checkpoint I, and '**`queue-size`**' which will follow the logic that we have implemented in this checkpoint.

CHECKPOINT III

In the previous checkpoints we have implemented two different logic modes: The round-robin and the queue-size. We would like to have the ability to add additional modes **without having to modify or recompile any of our existing code each time a new mode is added**. Consider and implement an extension mechanism that will allow us to do so.

FINAL THOUGHTS BEFORE WE START

- There is no single correct solution to this assignment, as it was designed to give you space to implement your own design.
- In case some things are not *completely* clear, please do not hesitate to contact us for clarifications – We are here to help you succeed.

GOOD LUCK!