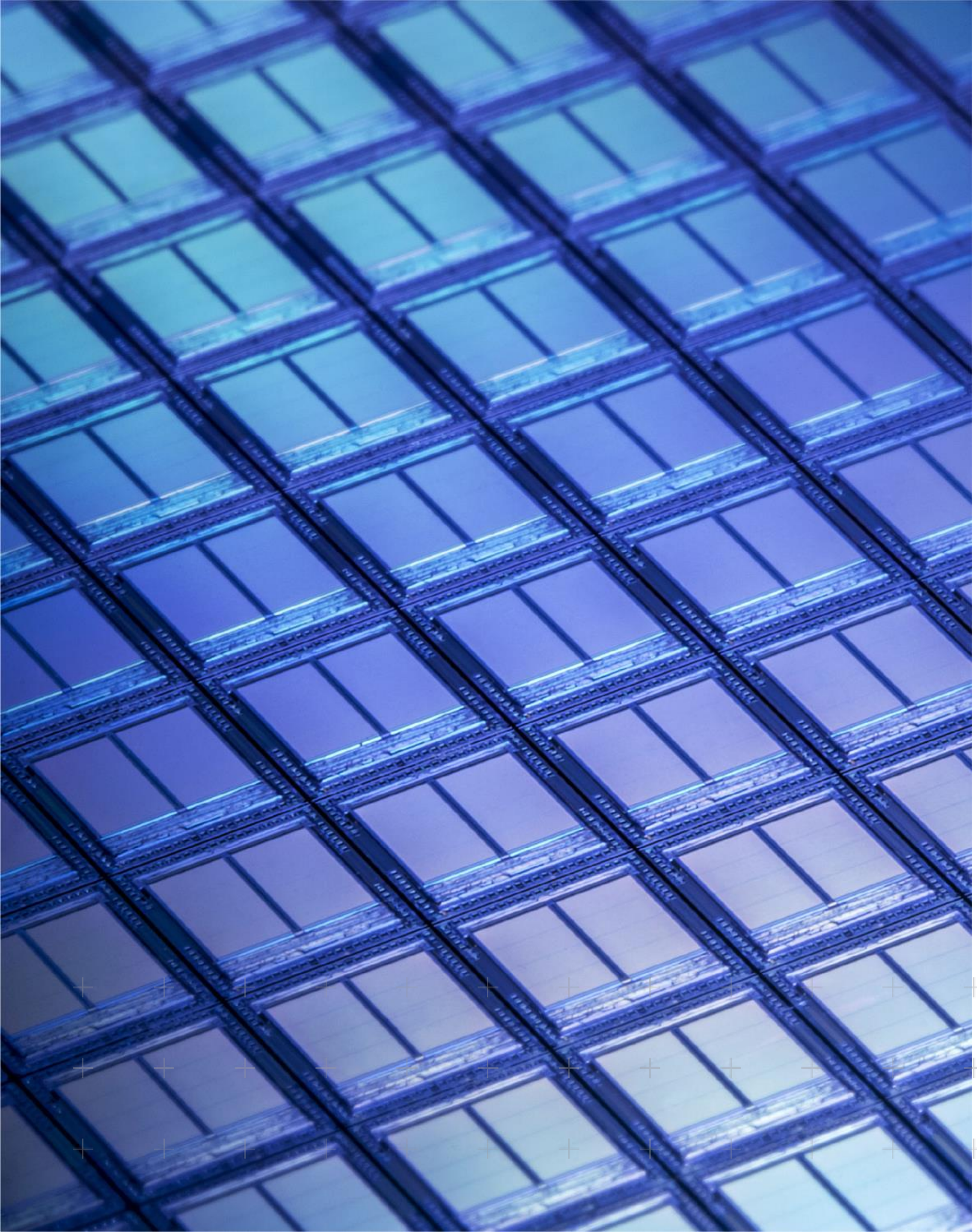


Peer Based Cloud Object storage

Feb 2019

Problem Statement:

Design and Implement a distributed cloud storage.

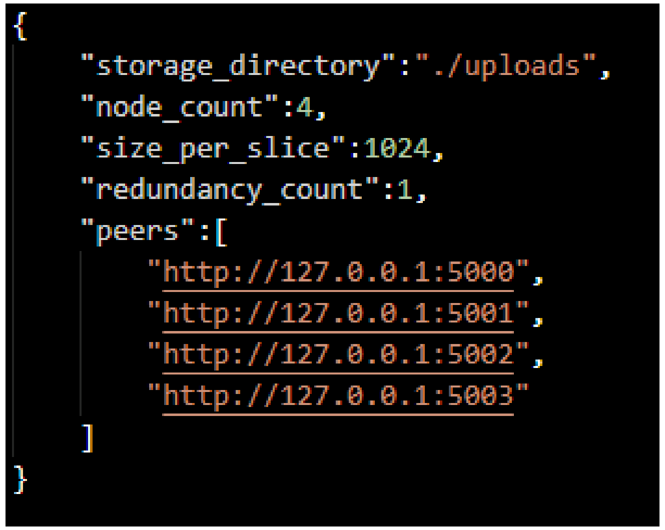
Similar to Google drive, where user can store any Objects.

What is an Object?

- Any BLOB - like txt, jpeg, mp4 etc…
- Which could be stored and retrieved on demand for anywhere

Basic Functionalities

- One web server
- REST API
- Config file:



```
{
    "storage_directory":"./uploads",
    "node_count":4,
    "size_per_slice":1024,
    "redundancy_count":1,
    "peers":[
        "http://127.0.0.1:5000",
        "http://127.0.0.1:5001",
        "http://127.0.0.1:5002",
        "http://127.0.0.1:5003"
    ]
}
```

- PUT
  - Store an incoming file and return the ID of the resource
- GET
  - Download the file for a given <ID>
- LIST
  - List the files stored in the server
- Delete
  - Delete the file for a given <ID>

API documentation

For Detailed documentation: https://praveenkumars.docs.apiary.io/

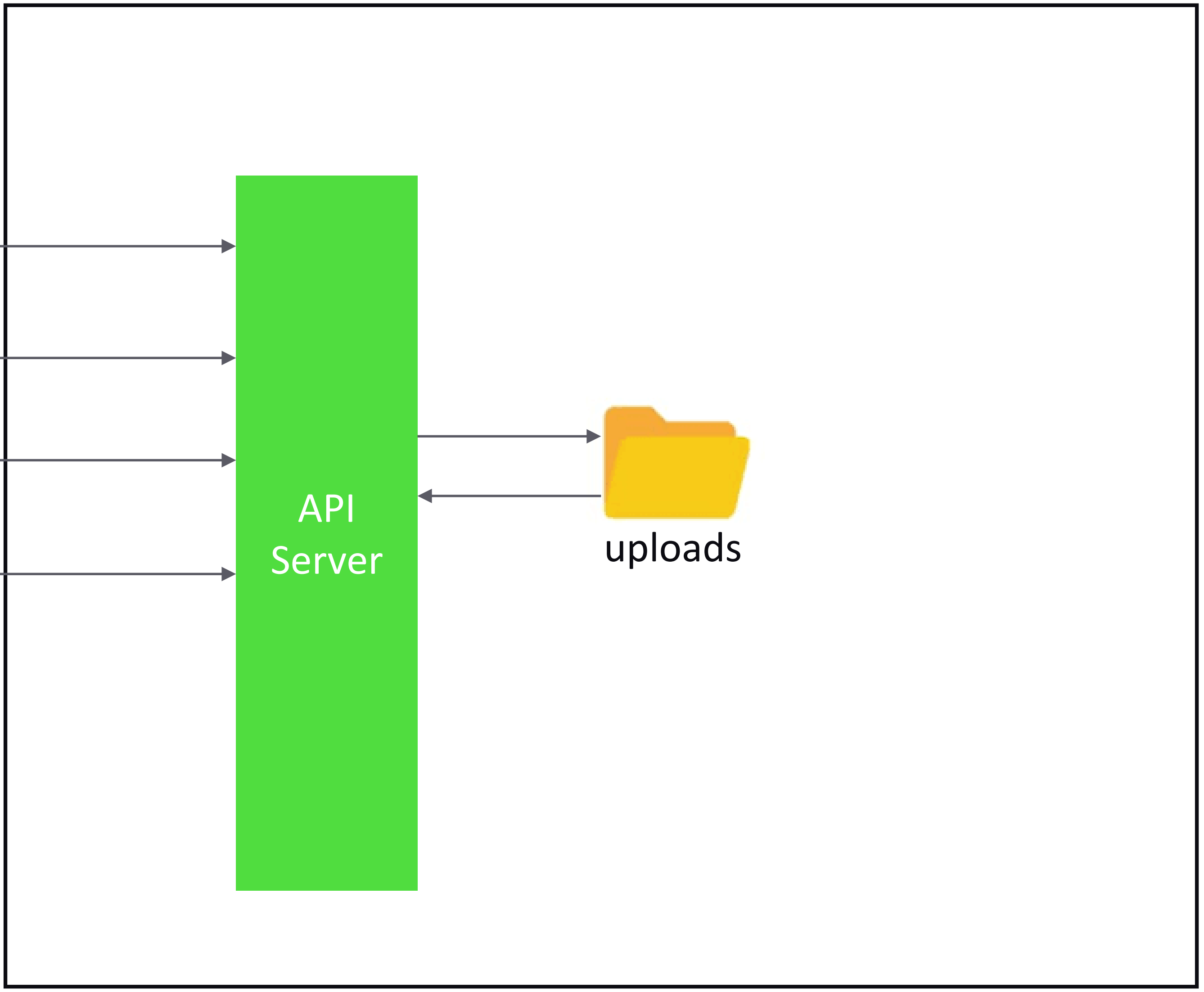
Description	Request	Response
Upload a File	PUT: http://localhost:5000/files ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="file"; filename="/D:/Users/prsubrama/OneDrive - KLA Corporation/Documents/Praveen/PrivateFoundry/milestone-1/Test/TestData/test_file_1.txt" Content-Type: text/plain	Content-Type: text/plain 5e1b3c4c-27f0-11ea-9a34-a4c3f0b4a7fe <id> of the resource is returned
Download a File	GET: http://localhost:5000/files/{id}	The File gets downloaded
Delete a File	DELETE: http://localhost:5000/files/{id}	Content-Type: text/plain object 5e1b3c4c-27f0-11ea-9a34-a4c3f0b4a7fe deleted successfully
List all Files	GET: http://localhost:5000/files/list	Content-Type: application/json [{ "file_name": "test_file_1.txt", "id": "ef5d2258-27cd-11ea-8e88-a4c3f0b4a7fe" }, { "file_name": "test_file_2.txt", "id": "ef6531d4-27cd-11ea-a4b7-a4c3f0b4a7fe" }

Constraints

- Client server architecture
  - Could be out of proc – stream/queue - REST API
  - Client sends data to be stored on server
- No public database allowed :
  - Expectation is to build their own customized data store
  - Sql , sqlite, redis, caching libraries are not allowed
- Can borrow code snippet / libraries from internet to build your own solution
- Follow schema and URL schemes as per the API documentation
- Follow folder naming conventions as per the documentation
- Postman collection import link: https://www.getpostman.com/collections/76e732538fe48439bfc2

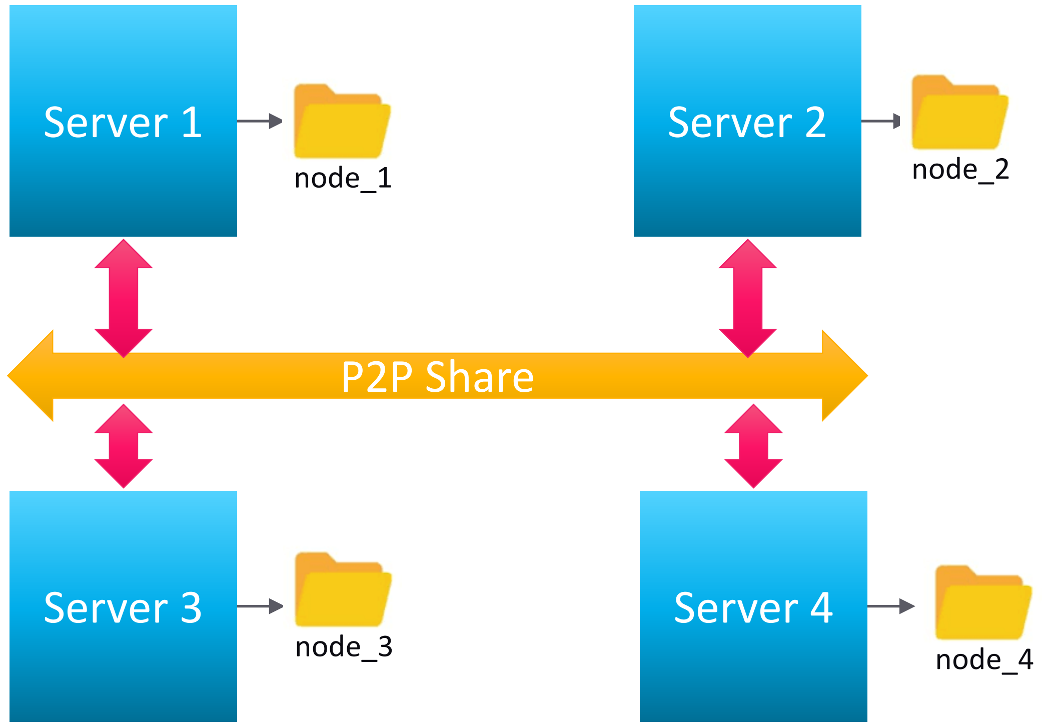
Deliverables - Milestone #1

- Goal: Basic API Implementation
  - Ability to Upload a file to the server
  - Download the file
  - List all the files on the server
  - Delete a file

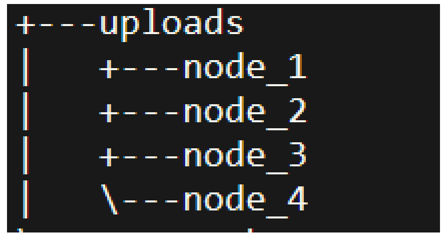


Deliverables - Milestone #1B

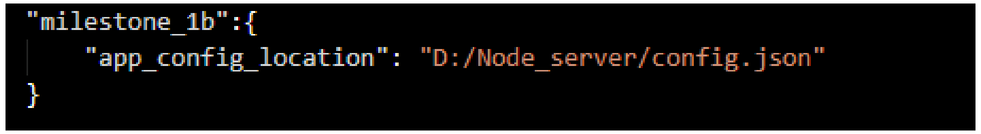
- Goal: Peer to Peer Sharing
  - Create multiple copies of the created Server (4 copies)
  - All servers(peers) should be up and running
  - Each server must bind to one of the addresses on config->peers
  - Each of them should have their independent node folder
  - **PUT**: Users must be able to upload a file to one server and get it from another
  - **GET**: Ability to retrieve files of any server using any available endpoint
  - **DELETE**: Users must be able to delete file on any server using any endpoint
  - **GET**(List): Ability to cumulatively display files on all servers
  - Important: Change the following value on your testconfig.json before running test cases for this milestone:



Folder Structure



```
+---uploads
|   +---node_1
|   +---node_2
|   +---node_3
|   \---node_4
```

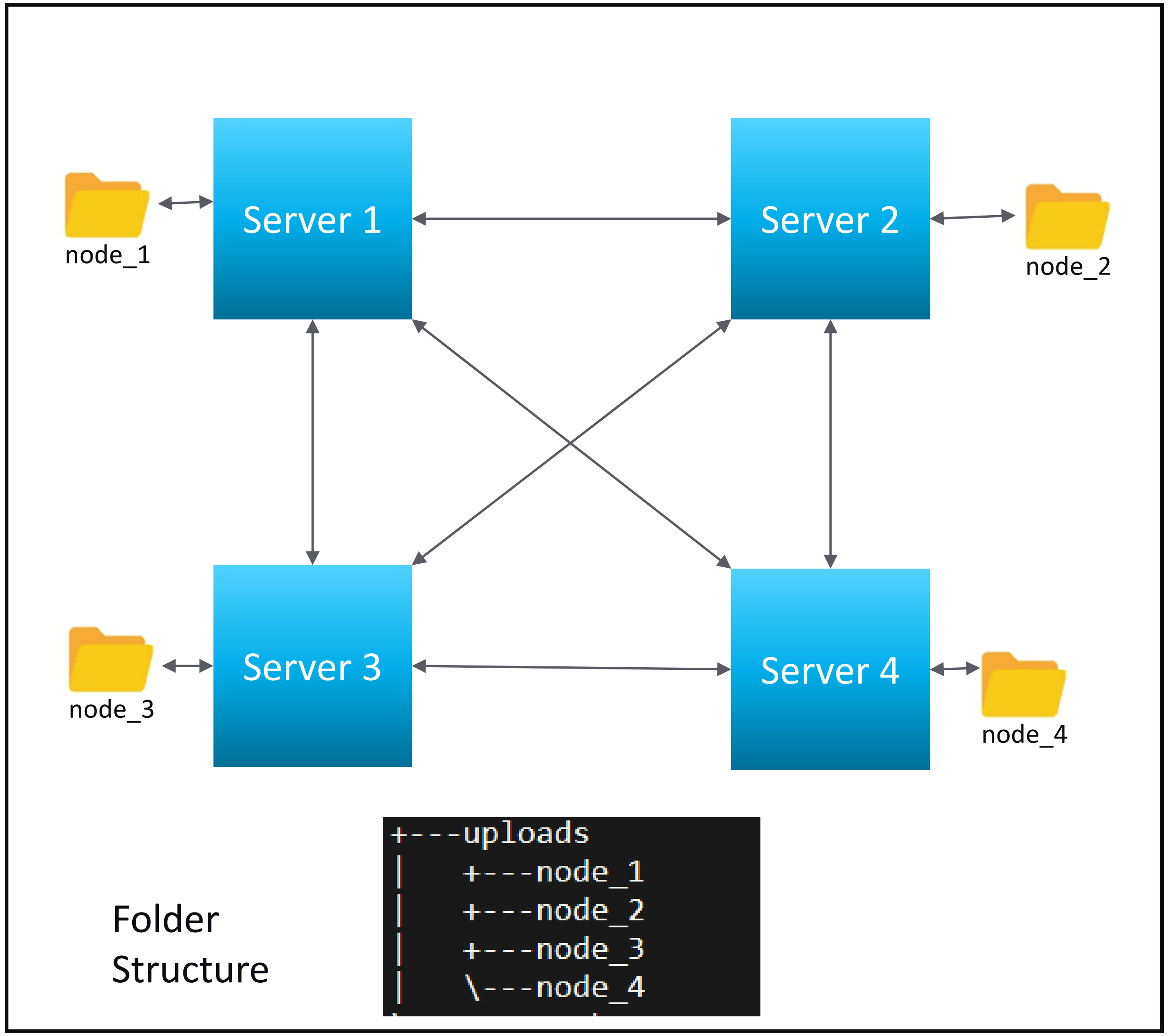


```
"milestone_1b":{
    "app_config_location": "D:/Node_server/config.json"
}
```

Value must point to the config file on your server code

Deliverables - Milestone #2

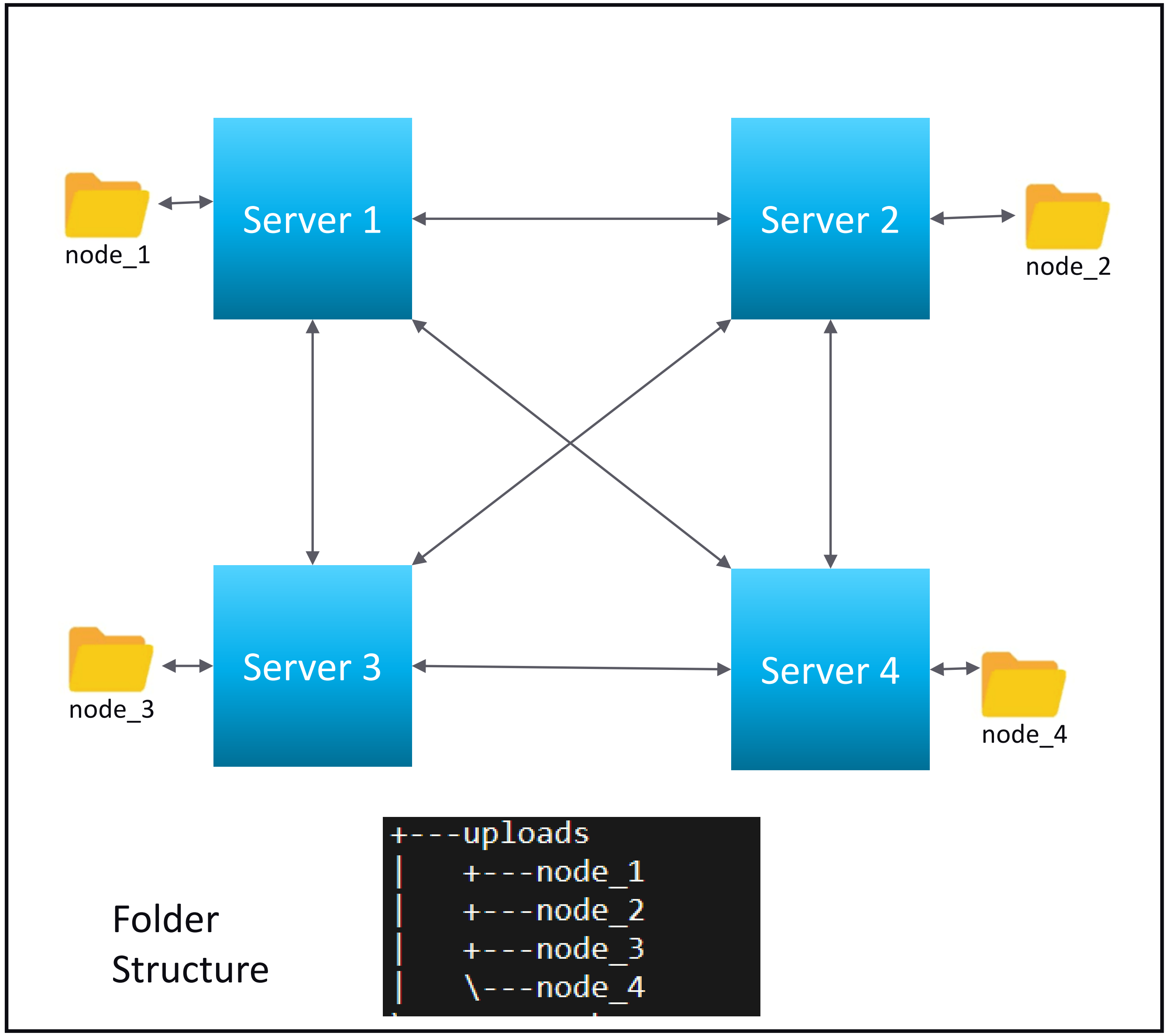
- Goal: Milestone#1B + Load Balancing
  - Nodes (folders) must be created as per the node count on config
  - Nodes must be named as node_<number> eg. node_1
  - Each node must be serviced by an API server
  - When a file is uploaded to any server, the file must be chunked based on the size mentioned in config and distributed to other API servers (peers) and stored on their respective node folders
  - Metadata file(s) must be created to save information on the file chunks and their location



```
+---uploads
|   +---node_1
|   +---node_2
|   +---node_3
|   \---node_4
```

Deliverables - Milestone #3

- Goal: Milestone#2 + Redundancy
  - When one or more Servers(peers) go down, the file must still be retrievable
  - Redundancy_count = 1 value in config means, file must be retrievable when 1 peer goes down.
  - Redundancy level must increase as per the value specified on the config
  - When one of the chunk file is corrupted, the server should still be able to retrieve the file with integrity



```
+---uploads
|   +---node_1
|   +---node_2
|   +---node_3
|   \---node_4
```

Running Test Validator

- Navigate to the Validator folder inside the Student handout folder. Choose Windows or Linux as per your Operating System

- Windows:
  - Open CMD from the Validator_Windows_V3 Folder
  - Sanity:
    - runvalidation.exe -e <your server detail> -t sanity -r <your roll number>
  - Milestone1
    - runvalidation.exe -e <your server detail> -t milestone1 -r <your roll number>
  - Milestone1B
    - runvalidation.exe -e <your server detail> -t milestone1b -r <your roll number>
  - Milestone 3:
    - runvalidation.exe -e <your server detail> -t milestone3 -r <your roll number>

- Linux / Mac
  - Open Bash under the Validator_Linux_NIT_V3
  - Sanity:
    - python runvalidation.py -e <your server detail> -t sanity -r <your roll number>
  - Milestone1
    - python runvalidation.py -e <your server detail> -t milestone1 -r <your roll number>
  - Milestone1B
    - python runvalidation.py -e <your server detail> -t milestone1b -r <your roll number>
  - Milestone 3:
    - python runvalidation.py -e <your server detail> -t milestone3 -r <your roll number>

Instructions

- Open Book format, welcome to use internet resources
- Prize will be awarded based on the following
  - Interactive review of your design approach
  - Result Validation
  - Solution completeness and performance
  - Original code contribution (vs. Library used)
- Participants are encouraged to work individually and refrain from helping other participants
- Participants actively engaged and present through out workshop will be awarded certificates of participation

Instructions II

- Make sure you use the attached Postman collection for testing your API
- Run tests using test validator with your roll number
- Do not run the validator unless you have some change to test
- Create a storage directory as mentioned in the config
- Nodes must have node_<number> as name. Eg. node_1, node_2
- All nodes must be under the storage directory as in config
- Make sure to update the test config with the storage locations before running milestone-1b and milestone-3 tests

Thank you