System Design - Cloud file storage

https://www.youtube.com/@15minutesystemdesign

Functional requirements:

Functional requirements:

- Upload/Download: User should be able to upload and download files and folders

Functional requirements:

- Upload/Download: User should be able to upload and download files and folders
- Share file access: User should be able to share files with other users

Functional requirements:

- Upload/Download: User should be able to upload and download files and folders
- Share file access: User should be able to share files with other users
- Synchronization: After updating a file on one device it should get synchronized on all other devices

Functional requirements:

- Upload/Download: User should be able to upload and download files and folders
- Share file access: User should be able to share files with other users
- Synchronization: After updating a file on one device it should get synchronized on all other devices

Non Functional requirements:

Functional requirements:

- Upload/Download: User should be able to upload and download files and folders
- Share file access: User should be able to share files with other users
- Synchronization: After updating a file on one device it should get synchronized on all other devices

Non Functional requirements:

- Low Latency

Functional requirements:

- Upload/Download: User should be able to upload and download files and folders
- Share file access: User should be able to share files with other users
- Synchronization: After updating a file on one device it should get synchronized on all other devices

Non Functional requirements:

- Low Latency
- High Availability

https://www.youtube.com/@15minutesystemdesign

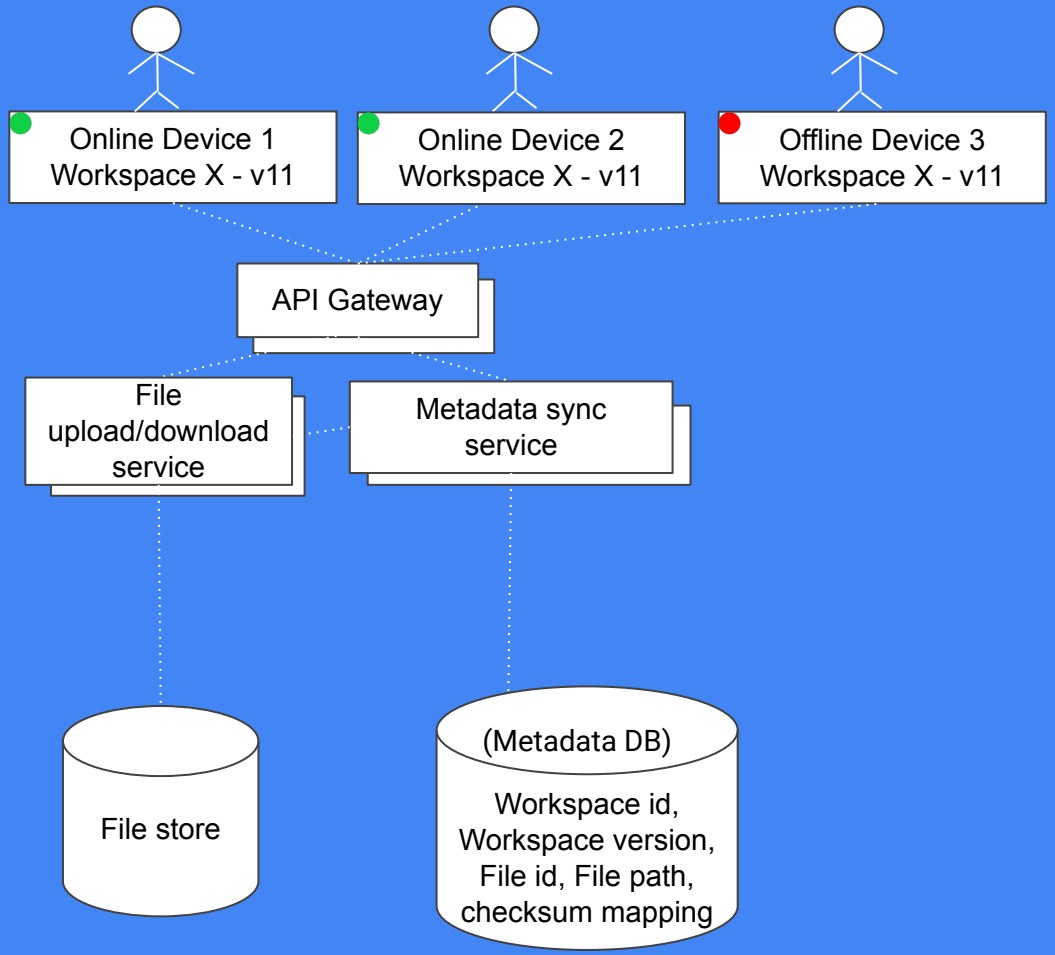
Functional requirements:

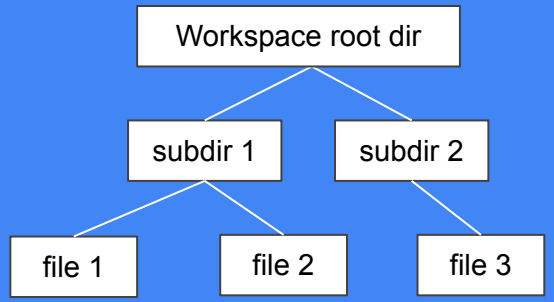
- Upload/Download: User should be able to upload and download files and folders
- Share file access: User should be able to share files with other users
- Synchronization: After updating a file on one device it should get synchronized on all other devices

Non Functional requirements:

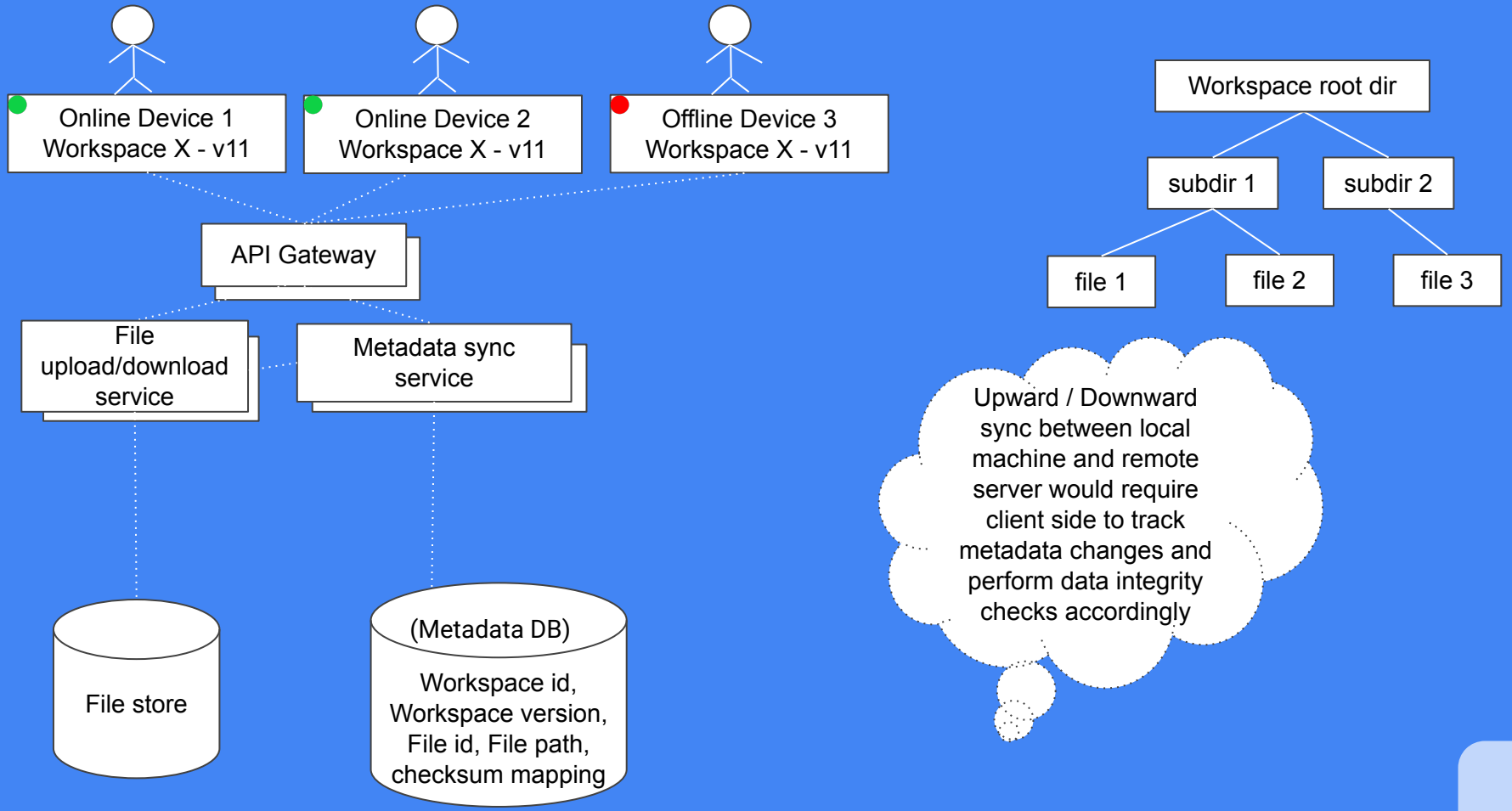
- Low Latency
- High Availability
- High reliability (No data loss, No data corruption)

https://www.youtube.com/@15minutesystemdesign

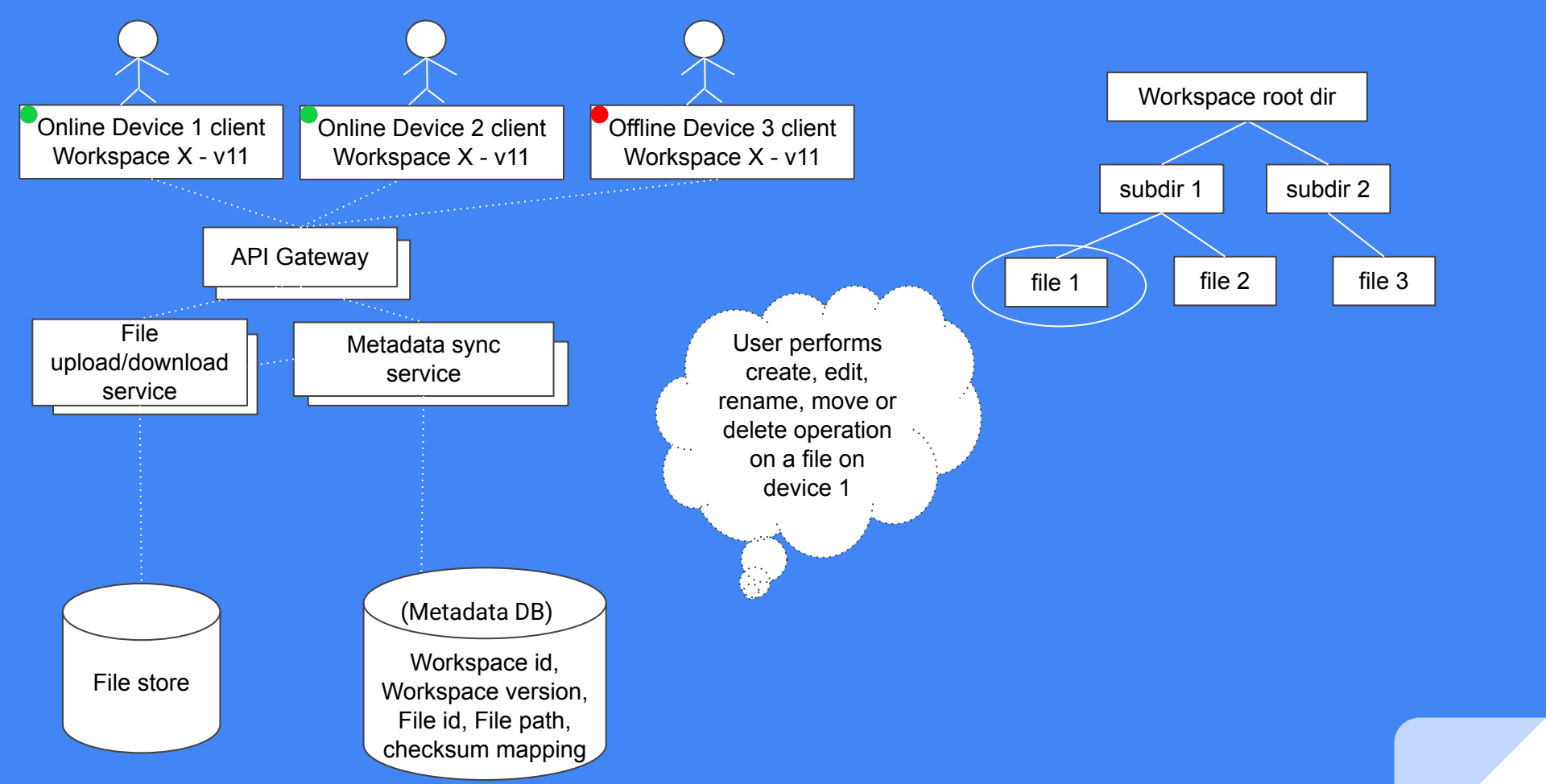




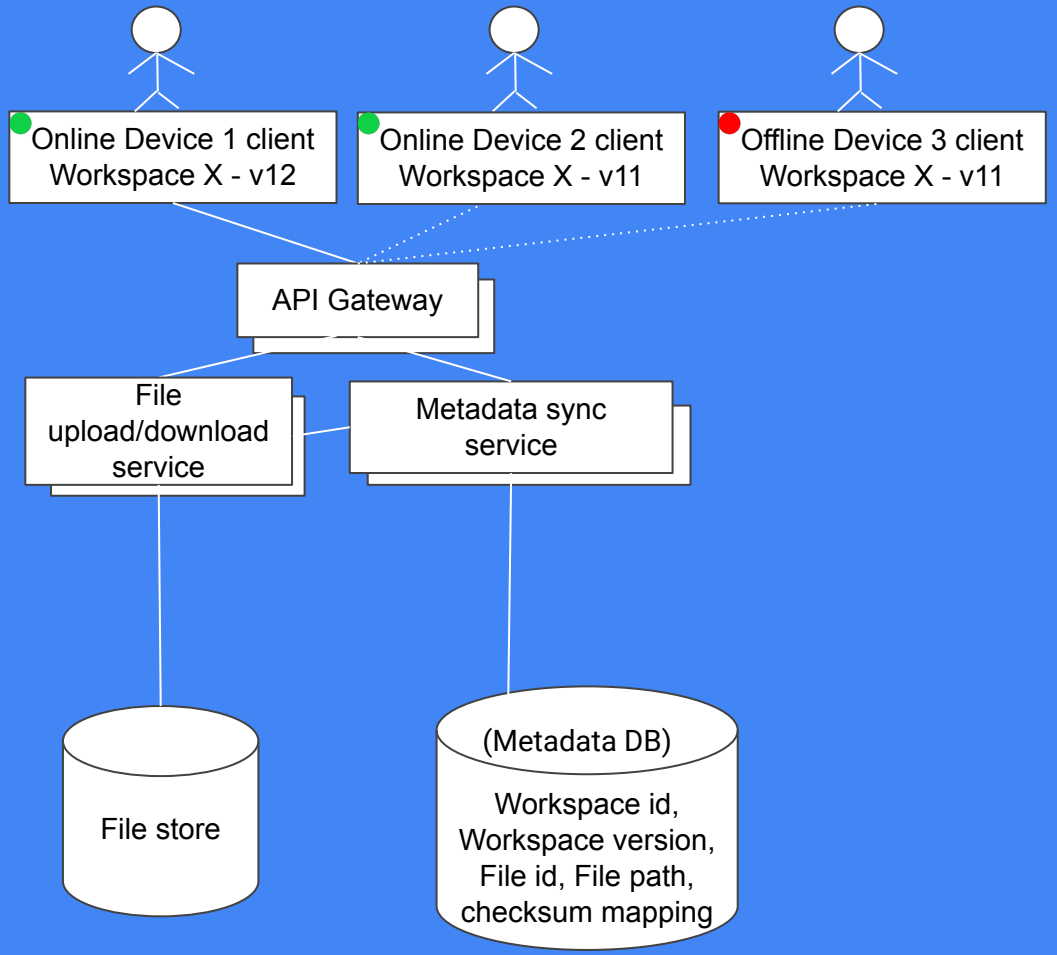
https://www.youtube.com/@15minutesystemdesign



https://www.youtube.com/@15minutesystemdesign

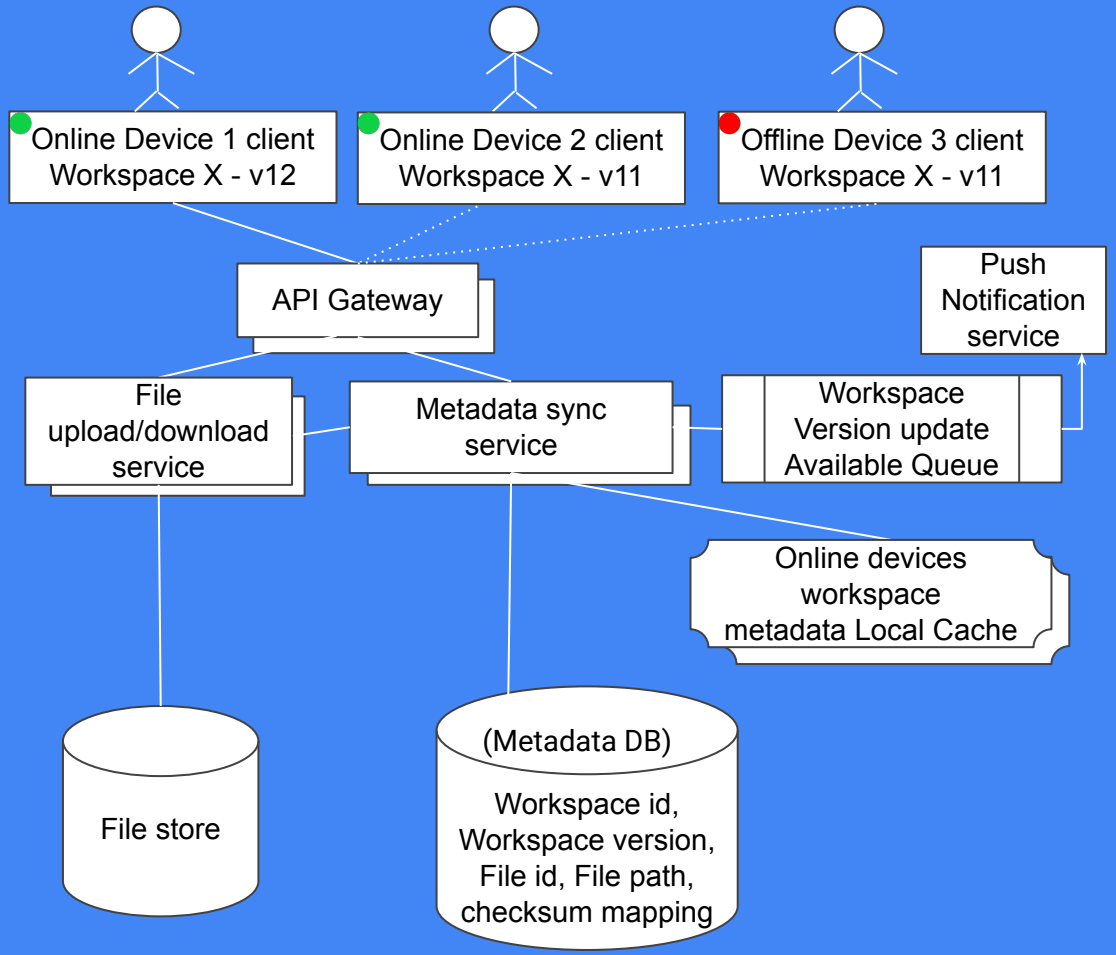


https://www.youtube.com/@15minutesystemdesign



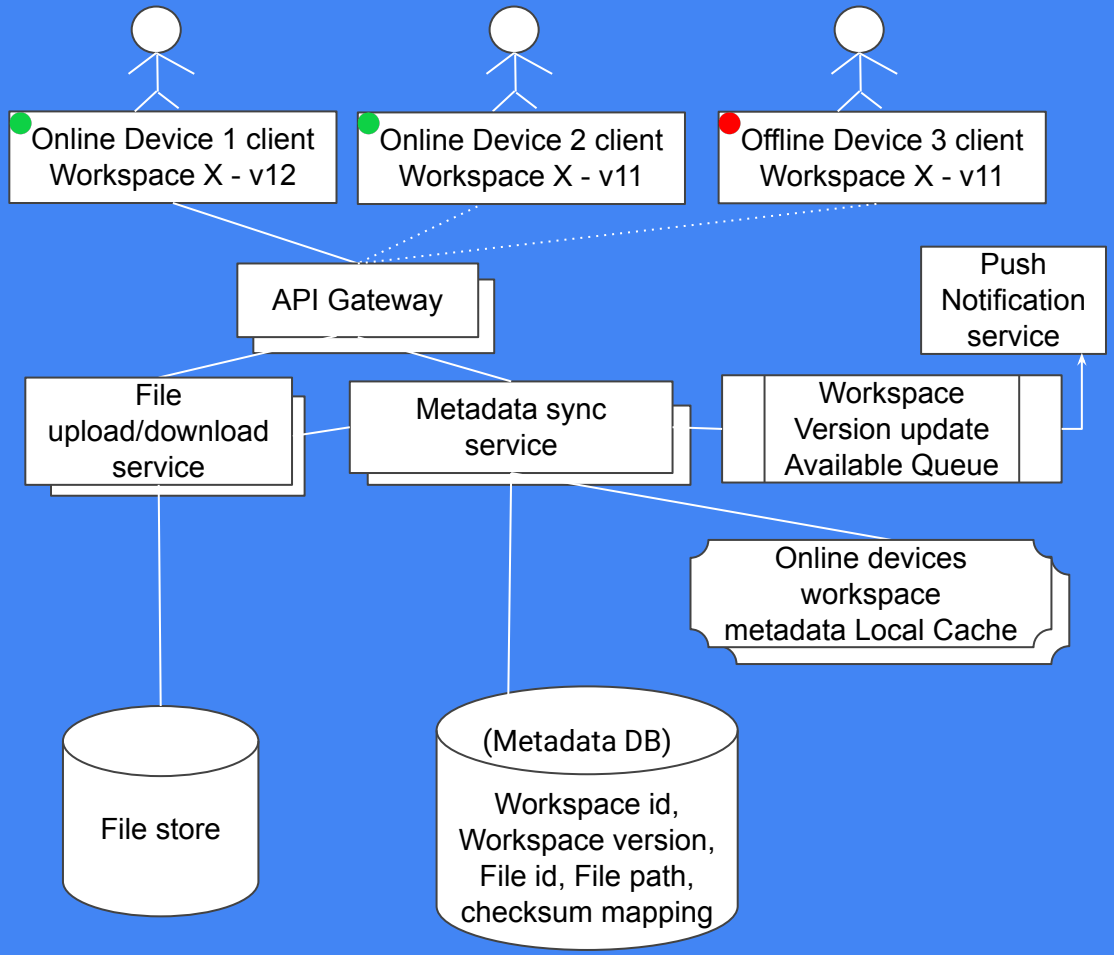
client need to increment workspace version locally then upload the file and metadata changes to cloud

https://www.youtube.com/@15minutesystemdesign



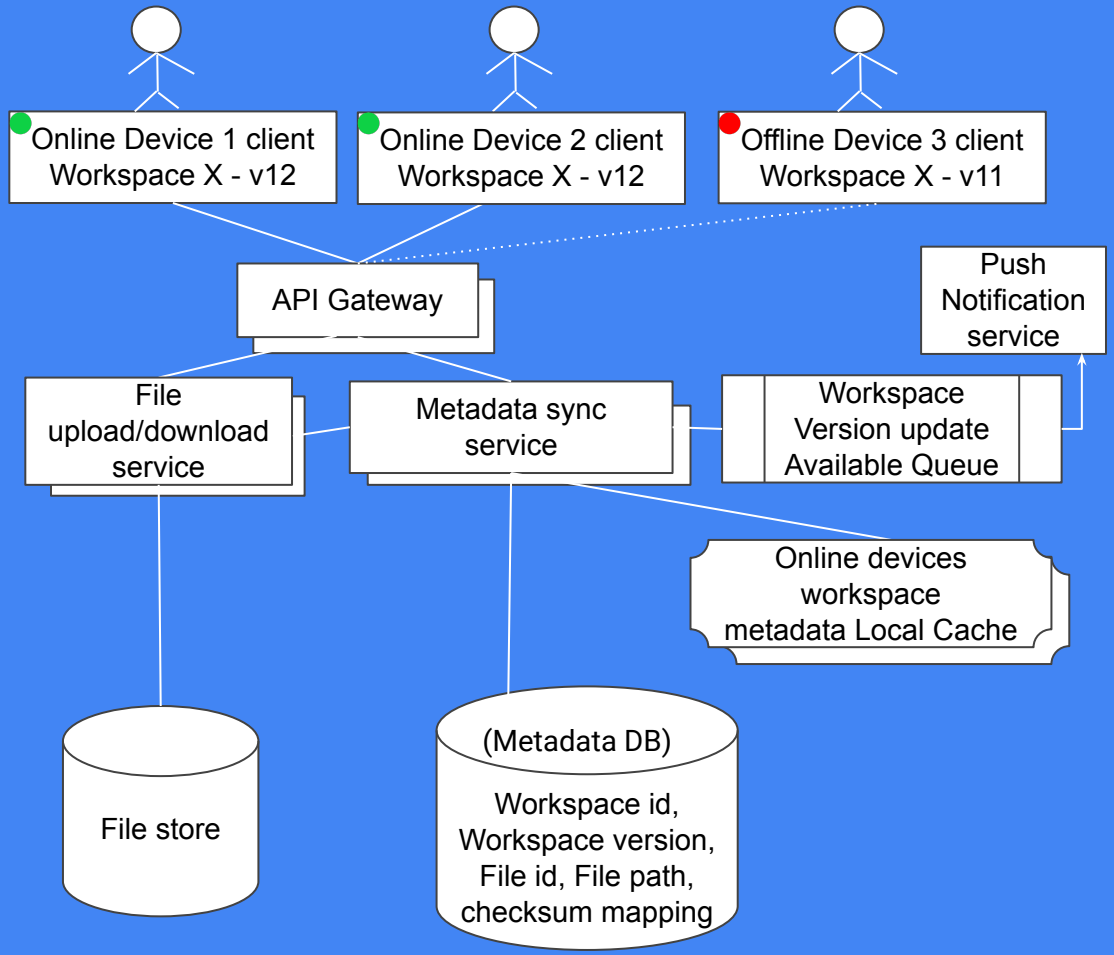
Cloud will update workspace version, file and metadata changes, load all metadata in local cache and then would immediately notify all devices having same workspace via push notification

https://www.youtube.com/@15minutesystemdesign



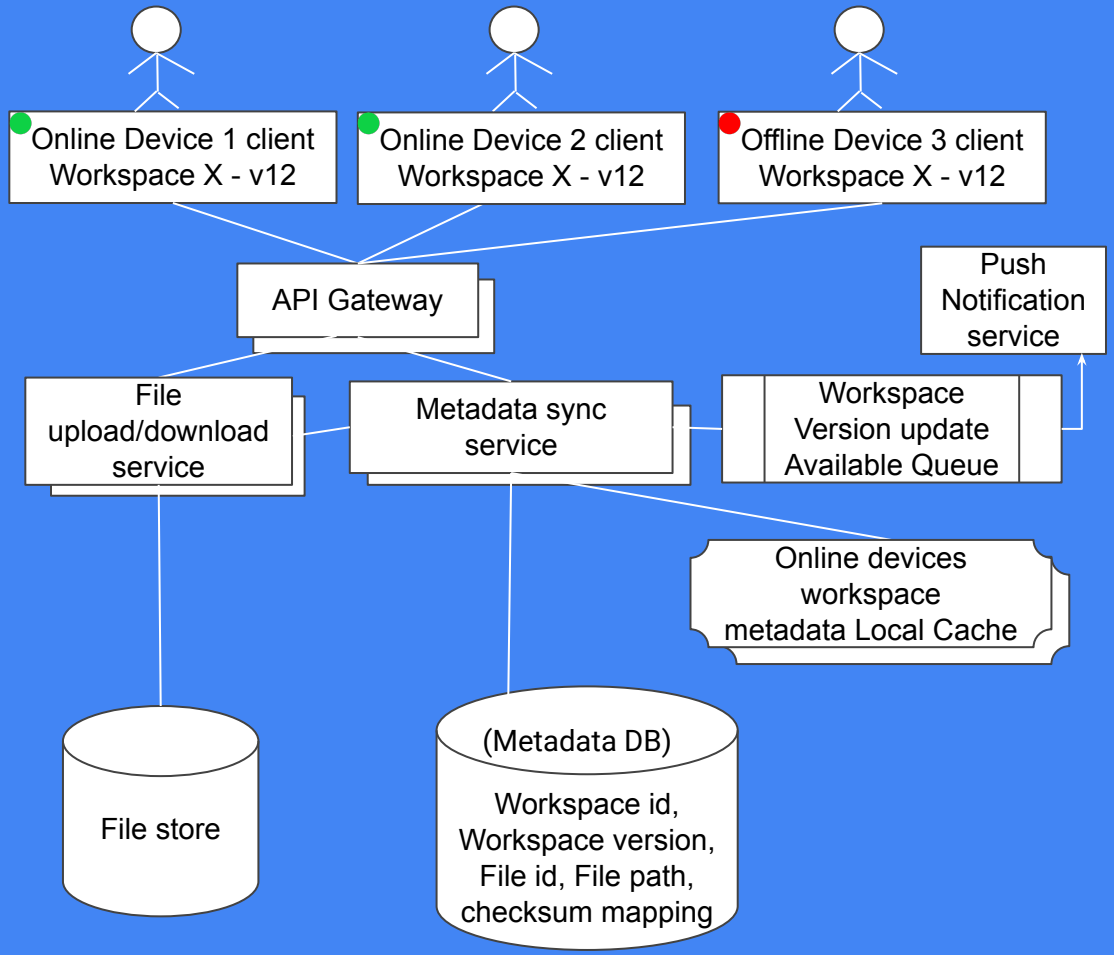
Post receiving notification, devices will compare remote and local workspace version and will send sync request to cloud accordingly

https://www.youtube.com/@15minutesystemdesign



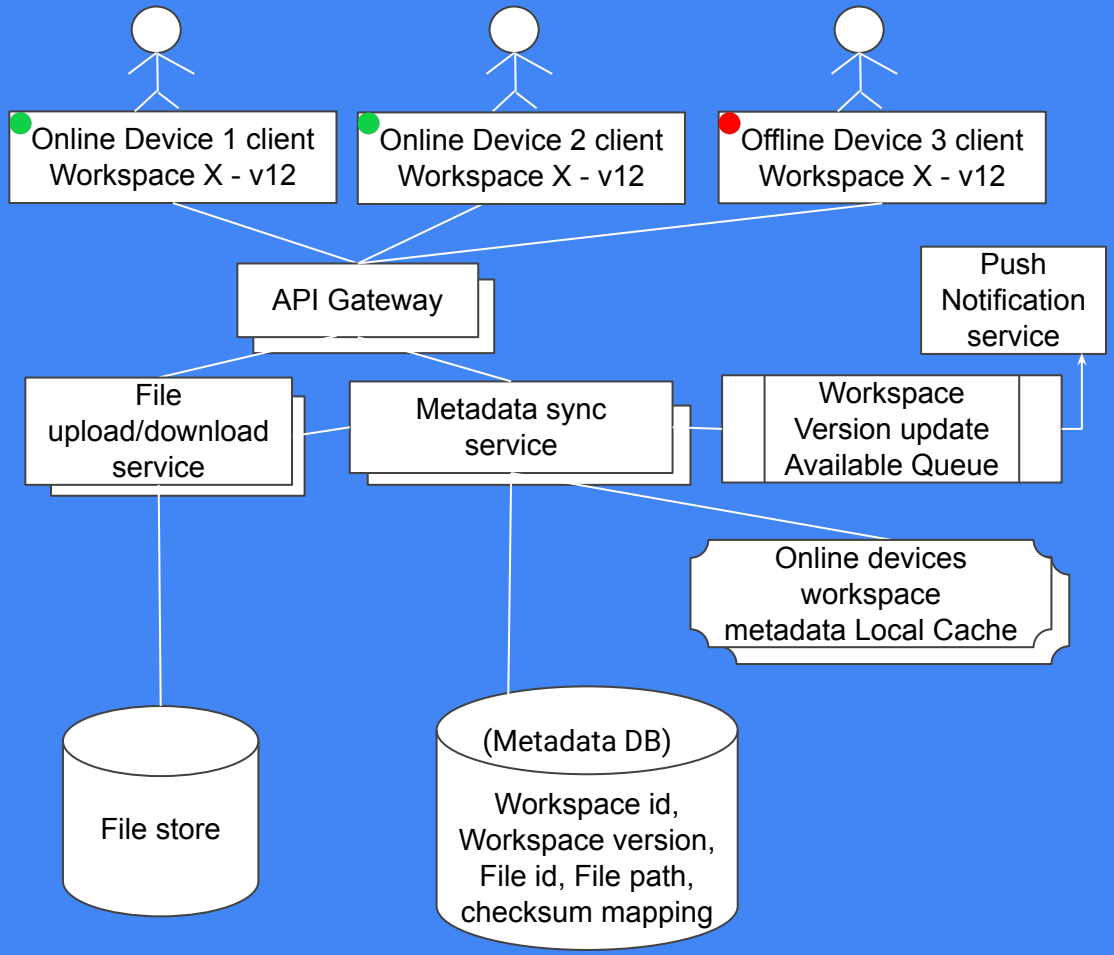
Cloud will send updated metadata as response so that client can download specific file id and update metadata changes locally in subsequent requests

https://www.youtube.com/@15minutesystemdesign



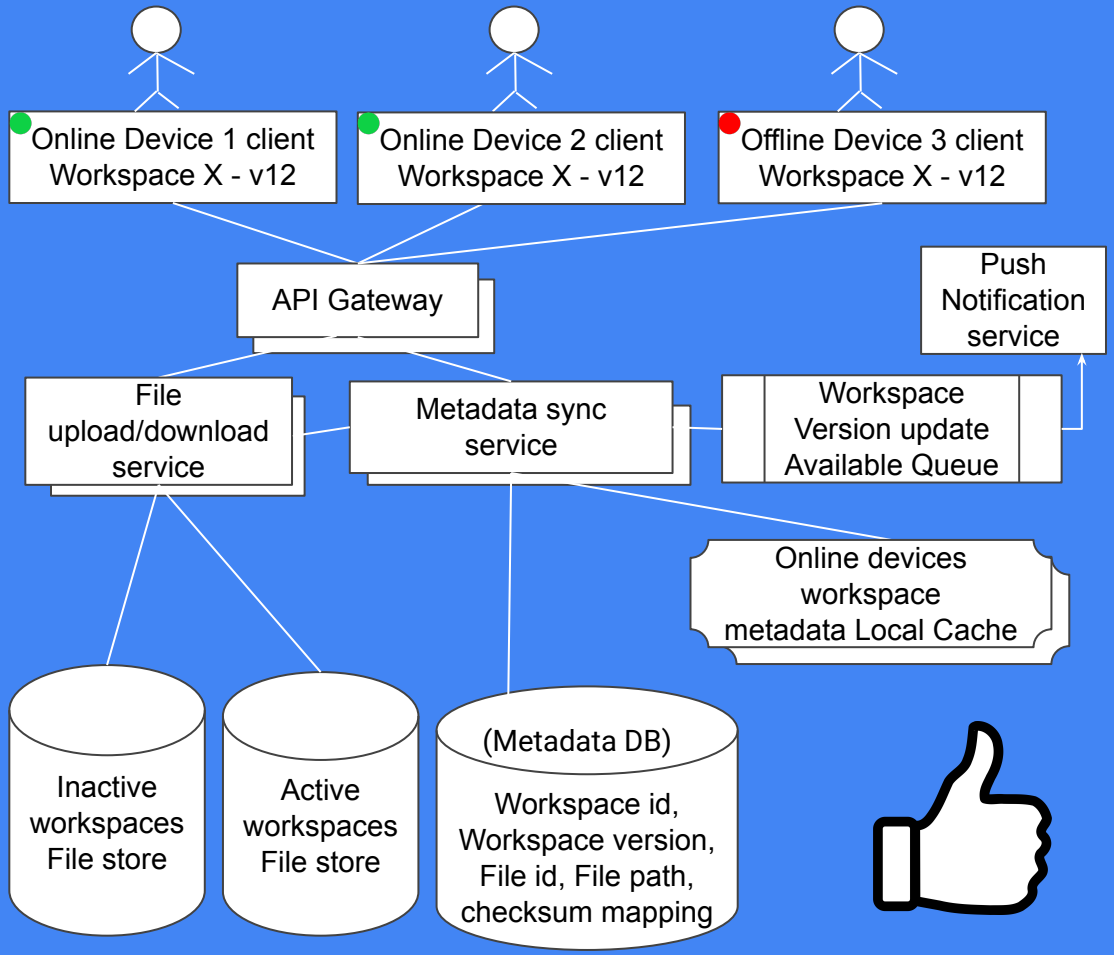
Offline device may send sync request to cloud once device comes online and repeat the same process

https://www.youtube.com/@15minutesystemdesign



We may also separate out active workspaces and inactive workspaces data to have better performance

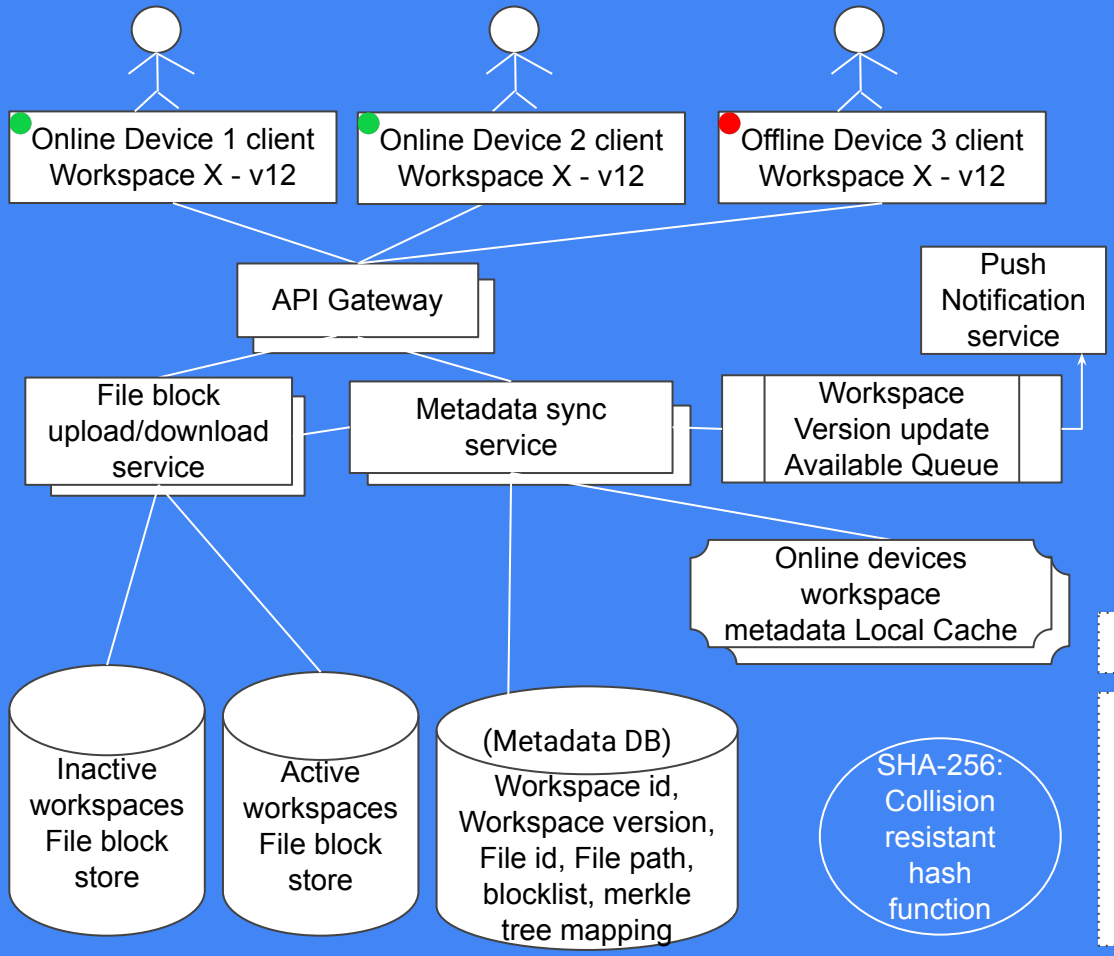
https://www.youtube.com/@15minutesystemdesign

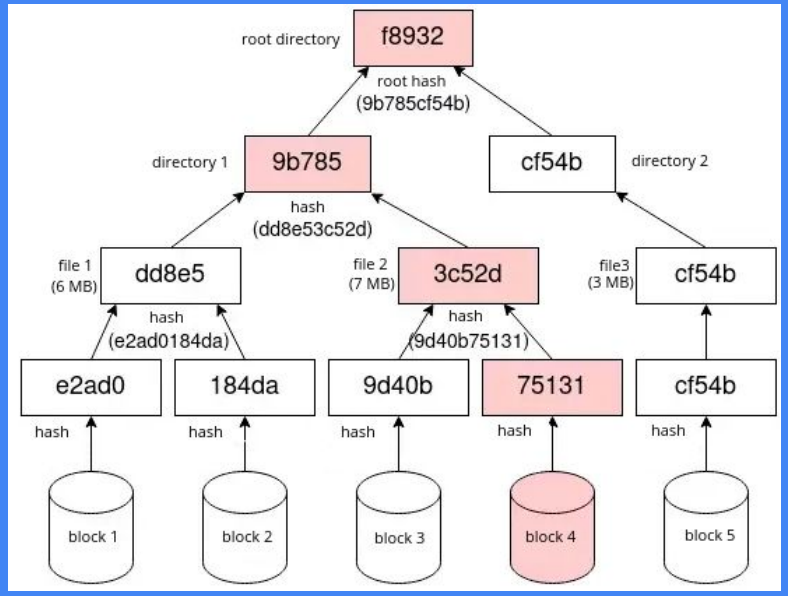


Issue 1: For small changes, uploading and downloading entire file would be inefficient. Can we do it efficiently?

Issue 2: how can we ensure atomic data operation to guarantee data consistency?

https://www.youtube.com/@15minutesystemdesign





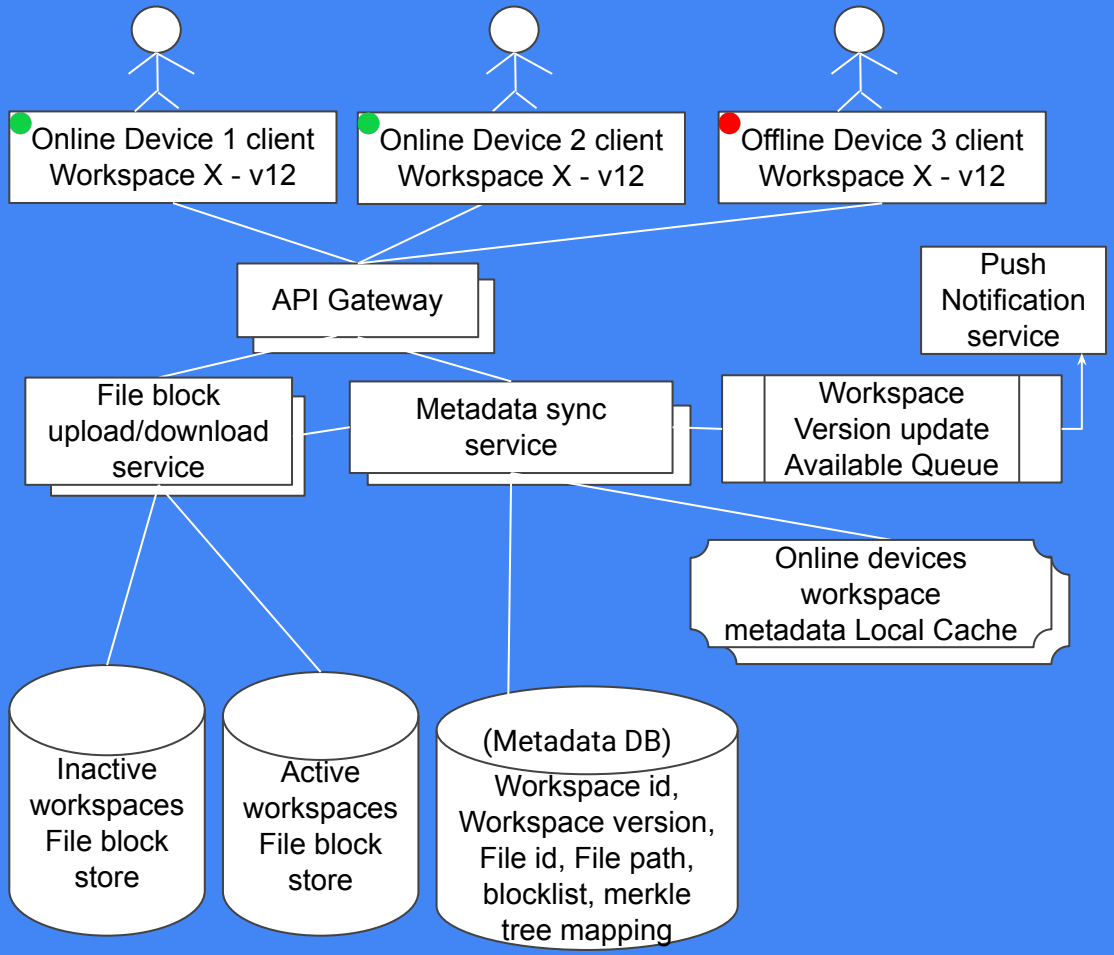
Audit trail: f8932 -> 9b785 -> 3c52d -> 75131 -> block4_id

Maintaining workspace level Merkle tree will enable us to have efficient data transfer and atomic data operation

---

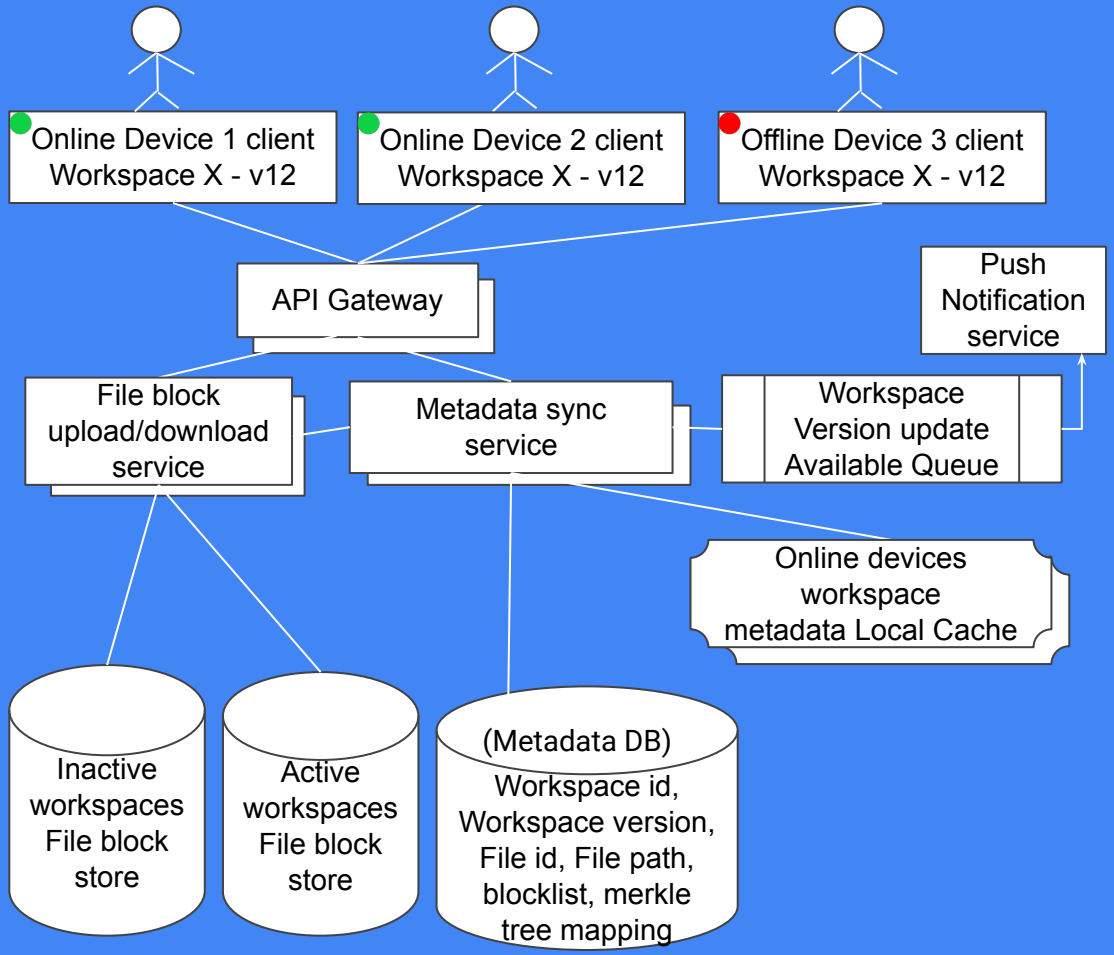
Searching in Merkle tree - O(logn)
Insertion in Merkle tree - O(logn)
Deletion in Merkle tree - O(logn)
Update leaf node in Merkle tree - O(logn)

https://www.youtube.com/@15minutesystemdesign



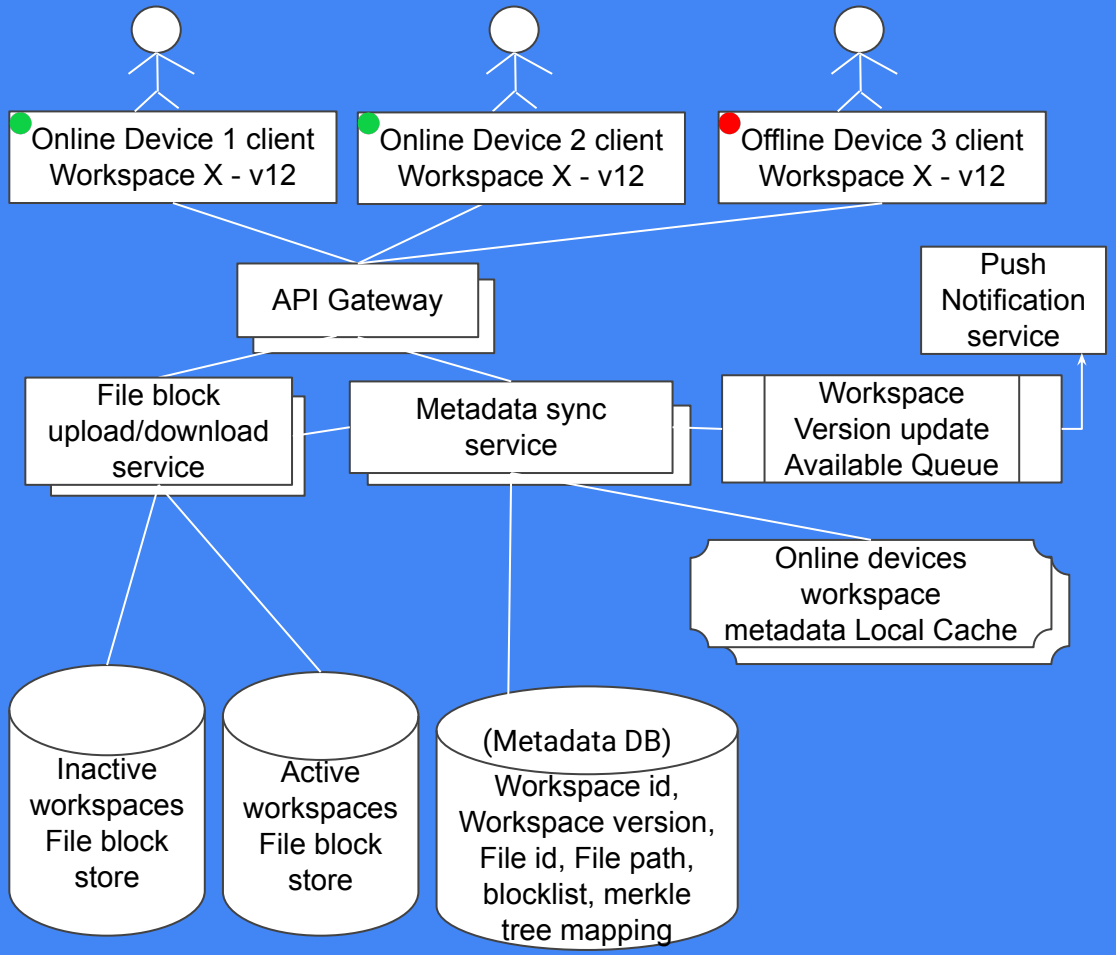
Client side responsibilities at workspace level:
(a) breakdown workspace files into blocks
(b) allow block to shrink/grow independently
(c) track changes in file content and directory structure
(d) keep blocks updated and stored separately
(e) compute workspace level merkle tree, blocklist
(f) check periodically if cloud has any version update available for current workspace

https://www.youtube.com/@15minutesystemdesign



Upsync algorithm:
Assuming client have updated blocks, merkle tree and workspace version locally
(a) client sends ws_id, device_id, ws_version to cloud
(b) cloud accepts sync request and provide upsync_id
(c) client sends upsync_id, block id, block content and merkle tree audit trail for each updated block
(d) cloud updates block content and metadata
(e) cloud verifies root hash using audit trail
(f) cloud confirms upsync completion to client
(g) cloud publish event for push notification service to notify other devices for "workspace_id version update available"

https://www.youtube.com/@15minutesystemdesign



Upsync algorithm:
Assuming client have updated blocks, merkle tree and workspace version locally
(a) client sends ws_id, device_id, ws_version to cloud
(b) cloud accepts sync request and provide upsync_id
(c) client sends upsync_id, block id, block content and merkle tree audit trail for each updated block
(d) cloud updates block content and metadata
(e) cloud verifies root hash using audit trail
(f) cloud confirms upsync completion to client
(g) cloud publish event for push notification service to notify other devices for "workspace_id version update available"

Downsync algorithm:
Assuming client have been notified or checked that workspace version update is available over cloud
(a) client sends ws_id, device_id, ws_version to cloud
(b) cloud accepts sync req and sends back downsync_id, block id, block content and merkle tree audit trail for each updated block
(c) client updates block content and metadata
(d) client verifies root hash using audit trail
(e) client confirms downsync completion to cloud

https://www.youtube.com/@15minutesystemdesign

Like, share, Subscribe

https://www.youtube.com/@15minutesystemdesign