Users have desktop client here

1 .And then this has some other data, user pay for storage here

A blackboard with writing on it

Description automatically generated

DAU daily active users here

2.

Average user has 2 clients: one upload and 1 download hte file here

3. These 2 are important concepts

Designt he google drive here

1. And why it is important here

4.

A blackboard with writing on it

Description automatically generated

The above is the back envelop calculation here:

1. 1 user will have 2 clients one for upload and 1 for download here.

A diagram of a service

Description automatically generated

**How to save the data?**

1. one gigabyte file over the network to our ingest server our ingest server is going to actually process the file that was uploaded and it needs to save it somewhere

2. Storing the large data

so in order to actually save our files we're going to use something called a binary large object store where we have it marked here as S3 and that's referencing Amazon S3 which is a off-the-shelf cloud provided binary large object store a binary large object is simply just a group of bytes right it can be any sort of file and that can be uploaded and then these binary large objects also known as blobs are then hosted at a URL that S3 will provide so for example right

Sotring the file information here as well:

we can think of that as being metadata so the metadata like what is the file name what is the file's URL so for here we're going to keep it very simple and we're simply going to have a mapping in our database between the file's name and the URL based on a user right we want to attach a user to that now that the file is uploaded into our database both in metadata and

**Notifiaciton service for another client:**

How are we going to notify another client so once the file is uploaded in our database. the NGS service is going to send a notification to our serving service this serving service is then going to notify another client for the user that this file has

updated so now the client on another machine knows that this file needs to be pulled so looking below then

this 2nd client is able to send a request to our serving service this serving service is now going to look the file data up in our data database right as we mentioned the metadata and it can see if the file exists and can download the file if it chooses to

**Uploading in small chunks**

Instead of uploading 2 gigabytes at once upload just chunsk at a time. Also when you edit a file can upload just chunk for a specific section of the file not for the whole thing.

If user edits a particular seciton of the file you know.

What db for the video metadata?

1. File name, user id, creation timestamp, lastupdated timestamp,

Subscription server?

1.Check if user has enough storage or not, this happens whenever new chunk is added if not then, then this means the user now needs subscription update as said

Scaling the database

let's take a look at our actual database scale so how are we going to scale our database and what are the numbers associated with that let's take a deep dive kind of into actually how much data we're going to use in order to manage files files as we can remember most of the data is stored in S3 that's where our 10 10 petabytes of data is going to be stored but the rest of the data is metadata in terms of Rights ther

A blackboard with text and numbers

Description automatically generated with medium confidence

e are million daily active users on our service and there are 100 updates so this means that we're going to make 10 billion rights per day to our database I actually mentioned the back of the envelope slides we can remember that there was a 500 byte worst case for the amount of metadata for a single file moving on to the second

**Definitely need read replications**

1. But have both multi write and read master and slave nodes since we have so many users here

2. Next using sharding

Decide on a sharding key here

that will require fewer changes down the line now that we understand the scale of all of our Opera operations let's move on and look at a high level diagram of exactly how our service is going to work now of course this whole thing is quite colorful and it has a lot of information in it but we'll go through it step by step so to start we have our user up here and they're uploading something so they're uploading a chunk our chunks are initially going to go into a queue and the reason that is is because our ingest which is horizontally scaled as designated by these three dots our ingest may not be able to handle an influx of

The final diagram

A diagram of a software system

Description automatically generated

**HOw to handle a lot of requests with a queue?**

Chunk uploads right if a lot of people are uploading once so in order to handle that load we're going to place a q in front of that so that way by

default right we can just have you know maybe a few things in this queue and they're just going to be pulled through as normal and you really don't or add that much latency to your requests but when we're in a heavy heavy load scenario lots of people are uploading this

queue really helps us out

**Subscription manager:**

1. Check if subscribed or not here every time new chunk is updated or uploaded here

2. Check if user paid

but we haven't actually notified new clients about these file updates so right as I mentioned right we have our client number one handled here but client number two still needs to understand they need to download so we're going to send a message and place it in a queue these messages are simply going to state that a file has been updated this is going to go to a Notifier service this Notifier service is of course horizontally scaled in order to handle an influx and load but we still have this queue in place in case right we have an extremely high influx this Notifier handles the logic to be able to identify and send information to a single user that a file has updated that they can download now that a client over here understands that they're able to download a file they can send a file or chunk download request this is going to go to our serving service and our serving service is first going to have to go to our metadata database this metadata database they're going to be able to look up the URL that's going to be in the S3 bucket to represent the blob of that chunk it's going to send a request to S3 right get the blob and use this blob in the serving service to send a chunk back to the client now the client has access to the chunk and they're able to replace that part of the file as

needed on their machine overall right we have everything working but we still need to handle how does a user when they run out of space so for example I upload a file but it gets denied how exactly do I how exactly do I pay for more storage space so we have the ability for a user to go to our subscription manager service and here they're able to send you know enter their credit card information and this will be saved in the database that they have paid for extra storage for example their monthly fee overall right this is a great high level diagram exactly how the services work there are some smaller details that need to be discussed but I think the most glaring one is we mentioned that this database was sharded right we can see that it's horizontally scaled here right we have a Shard one all the way to Shard n so let's review exactly what is needed in A Shard key right A Shard key needs to have high cardinality it needs to have a low frequency and it needs to fit our query patterns very well looking at those constraints if we take a look at our metadata table we can pretty easily see that we're going to need a combination of user and file ID now let's consider if we just did each one of these independently if we just did the user key this means that we would distribute the users across our shards but what if one user has more files than another so for example this chart is very much stacked up with a lot of files versus this one only has a few as well as if we distribute by file ID if we ever want to look up files by user right we can be doing a scatter gather request and going to all of our partitions which is not helpful in general most of our clients are going to be signed in with the same account right so this user is going to upload but they are on for example the Bob user account and this user is going to be this client is going to be downloading something but they are also signed into the Bob account to be accessing the same files we don't have any file sharing between accounts right now so this means that we need to be able to look up files very easily in terms of a user as well so overall this means that a combination of the two is great because it supports our query patterns very well as well because for example if I want to download my file for the first time I'll send a file request to download and then I can go to the metadata database and look up all of the blobs that I need to download and they will be grouped together then we can go to S3 all at once and stream this information back through the serving service back to the client now we also need to consider our user data for our user data the most obvious Shard key is user right there is no real reason to place user data scattered across a bunch of nodes right if we need to look up information about the user it's good to keep them all on OneNote overall that will be our short key okay so to get into wrapping up on this problem there are some further considerations that you can make in the high level diagram we just saw and in general the the design first of all how exactly are we going to be able to orchestrate the service so as we mentioned right there's a lot of horizontal scaling in our service so how exactly is that going to be orchestrated right how are we going to spawn new instances how are we going to destroy them and how are we going to manage load in general next how are we going to do connection management for each of the clients right we said that right for example the notification service was able to send a requ send you know notification to the client and this client would be able to be updated but this means that we need some sort of bi-directional relationship it isn't right we can't just do this between standard HTTP because there's no way for our server-side service to get information over to our clients there needs to be some sort of connection Management in our service to handle that next up how is the desktop client design right we mentioned that we would just for example send a chunk back to a client but how exactly is the client going to be able to replace that Chunk on their machine or for example how is a client going to understand to break up these files into chunks that is all part of the desktop client design and can be another element of this interview next in general right we only considered a few failure cases but there are always tons of failure cases in terms of things you know randomly shutting down right what if there's uh like major issues with our platform right what if S3 goes down right we are depending on an external service so all kinds of things like that how exactly would you handle those failure scenarios in your implementation finally there are tons of examples of different considerations that need to be made about this service right but they're all what I would say implementation details so right the high level diagram as well as the other implementation details we made around subscriptions and the chunking are very useful and I think that those are very uh you know integral details to the problem but there are additional implementation details that are able to be worked out in addition to the ones up here so to look at overall what we achieved in this video we first looked at a very naive solution and then moved into chunking that way we could minimize our bandwidth right which is I would say a very very essential understanding about this problem and a major integral part of what an interviewer will be looking for for in asking this question next we actually crunch them back at the envelope numbers and that was helpful for us to understand you know do we need to scale our database horizontally right which we did and that was the best solution for us right how exactly are we going to implement scale in that and it honestly got us to understanding that we even needed to do orchestration or connection management for that matter right in general this made us understand that we needed a scalable service and there are tons of numbers and opportunities in this section they are able to you know do as exercises on your own then since we were using so much data with those numbers we crunched we looked at actually managing the money right we need to actually handle subscriptions for our service so that way our service can make money and not just store files for free and how we can actually Implement that as part of the ingest service right going over to our subscription service which can then check the database and ensure right that everything that the user is paying for it is able to be satisfied finally we looked at the high level design and that was very useful in understanding the exact the exact flow right for example how a user was able to upload something to our service how we used a queue there for example and things like that overall though I invite you to look back through the video and especially through these further considerations as exercises you know on your own to be able to better understand the service but overall I hope this gives you a great start and understanding how to implement a Dropbox clone in a system design interview and can help you prepare if you enjoyed that video you can get a lot more content just like this on interviewpen.com WE publish two to four videos a week really it's just an arbitrary number it's whenever I can sit down and do a video because these videos take a whole day to do and we're always online to answer any questions you may have join our Discord join our newsletter the blueprint where you can get more weekly data structure and algorithm and system design kind of topics and subscribe and like this video if you actually like this video and it helped you and also tell a friend that we exist that's all