My implementation of CDN architecture:

We have 3 server roles:

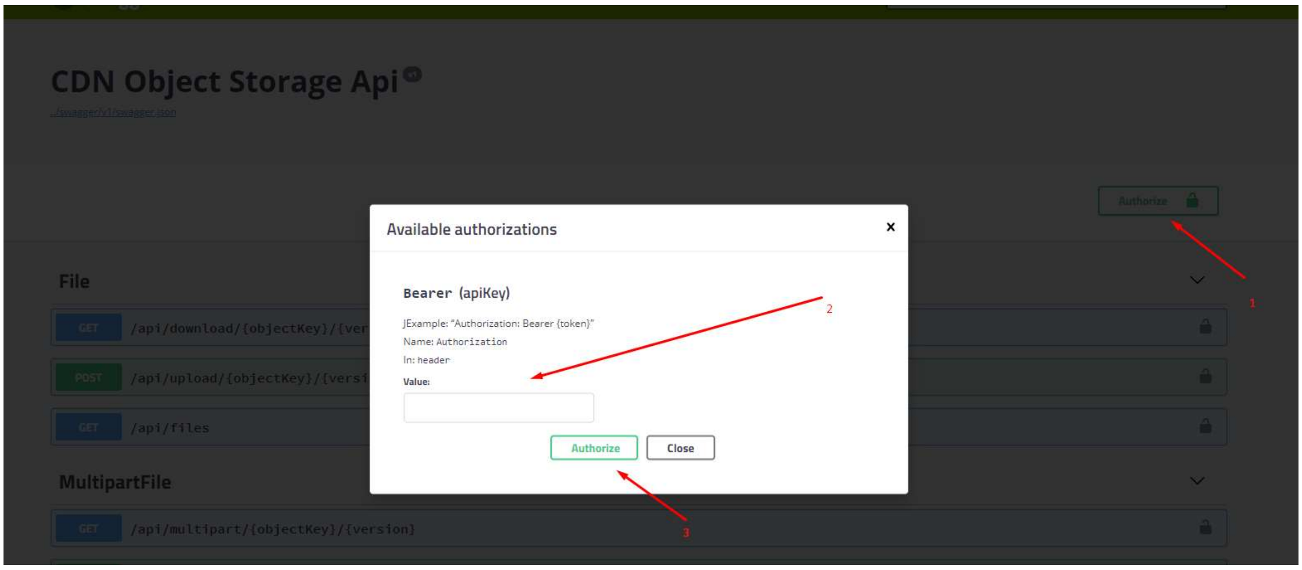
- FileStorage servers (OriginServers) – this servers will contain files which users were downloaded, for long period
- Cache Servers – this servers will be located in different countries, and their goal is just to cache file object. I implemented them like lazy-loading behavior, so for very first request for given file, cache server will download it, and will store id. (more info will be provided in next topic)
- Route Server – this servers are responsible for redirecting client requests to nearest Cache Server. Ideally, they should resolve it on DNS level. But in this solution Route Server is just REST service, which redirects requests.

Also this architecture can be divided on micro services, so we can add microservice which will be responsible for Video streaming.

In my implementation, all parts are custom, but actually, we could use Basic File storage and DNS server resolver, but API for upload/download will be custom. Cache servers implementation is also custom.

File Storage Servers (Origin Servers)

Actually I use the same server for storing file and hosting API, which enables all required operation on objects.
All REST methods are documented via SWAGGER (use this link to make requests http://localhost:54184/swagger)
All methods are protected by token, to Authorize via Swagger click on 'Authorize' button



JExample: "Authorization: Bearer {token}"
Name: Authorization
In: header

List of tokens are stored in appsettings.json, but I will duplicate them here:

```
"Tokens": [ "49ec97b507cc21806861817f2f0564d4", "902c618ee865cba08d350699b8f5b43f",
"40413ed3e7e7a2895362fc3b7308dc11" ]
```

API supports chunked upload (you can test it via swagger, use http://pinetools.com/split-files tool to split file)

1. Start chunked upload (POST http://localhost:54184/api/multipart/test/start)
   Store upload id which you will receive in response.
2. Upload chunks as many as needed (PUT http://localhost:54184/api/multipart/test/0/e85bfdc9-3f39-4324-995c-699d9680a749?partNumber=1). Part numbers should be unique
3. For end multipart upload (POST http://localhost:54184/api/multipart/test/0/e85bfdc9-3f39-4324-995c-699d9680a749)
4. In case if need to abort upload (DELETE http://localhost:54184/api/multipart/test/0/e85bfdc9-3f39-4324-995c-699d9680a749)

   All provided request by me are just sample (do not use them).

For chunked download, use GET /api/multipart/{objectKey}/{version}.
I honestly wanted to use 'ByteRange' header for it, but for comfortable testing I used query string for passing start byte and end byte (example http://localhost:54184/api/multipart/Image.png/2?start=0&end=15)

All other request are simple, don't think that need to show example.

Cache Server

So the main feature of CacheServer is to be near to user, and cache files.

Example of getting object: http://localhost:54598/Image.png

In my implementation, cache server has background job, which in some interval checks Server free space and cleans up it in case of something. This job also updates database with actual value of size.

All these features could be configured in appsettings.json

```
//Inverval for checking free space
"CleanUpCheckInterval": 60,

//If free space is less than provided value, cleanup will be done
"CleanUpWhenAvailMemoryLessThanPercents": 20,

//Age of objects which needs to be deleted during cleanup
"CleanUpObjectAgeInDays": 14
```

Could be configured in

In case if there will not be free space, and any objects to cleanup, CacheServer will redirect request to another nearest to client CacheServer. (so should not be any failures).

The problem with free space on cache servers could happed sometimes, and we can play with these parameters to reach the best performance/storage cost. It depends, if we need good performance, need to have CacheServers with a lot of SSD memory. If need to not pay a lot of money -> need to do cleanups very often….

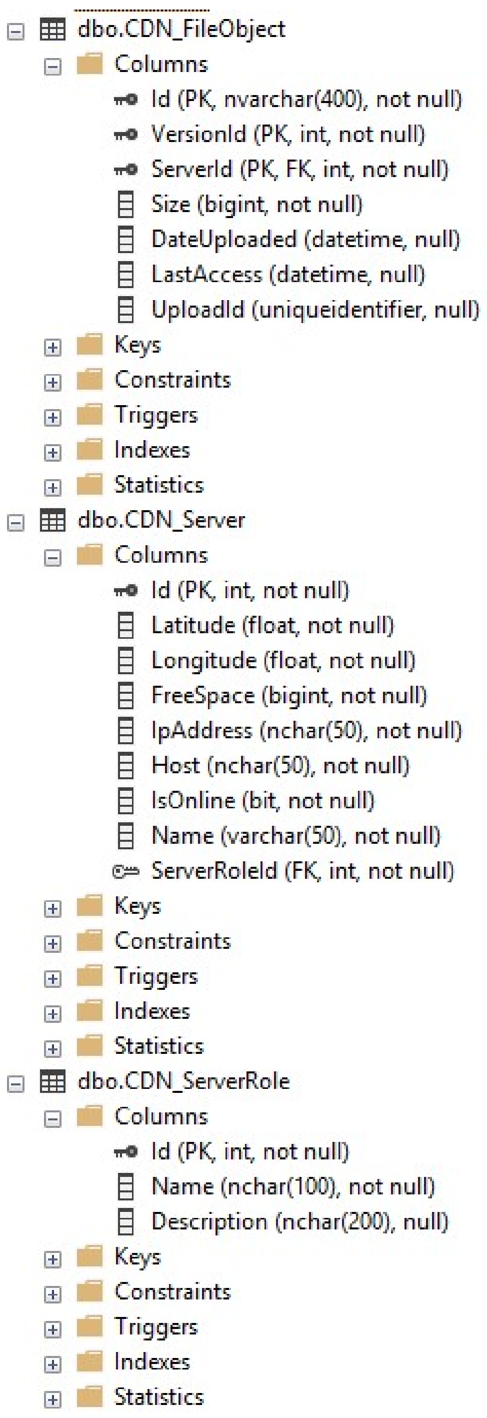
Route Servers

The main idea Is just redirect requests to nearest cache server. For testing you could pass latitude and longitude in query string (but I implemented solution, to get user location by IPAddress, I used free subscription of `http://api.ipstack.com`)

Project Structure:

- CDN.RouteServer.Api – this is REST client for redirecting requests to nearest server.
- CDN.OriginServer.Api – this is like API which allows to upload files (including chunk upload/download). It supports file-versioning
- CDN.CacheServer.Api – this is server which is responsible for caching objects, to deliver them fast.
- CDN.Domain – this lib contains all common logic.

Database Structure



```
dbo.CDN_FileObject
  Columns
    Id (PK, nvarchar(400), not null)
    VersionId (PK, int, not null)
    ServerId (PK, FK, int, not null)
    Size (bigint, not null)
    DateUploaded (datetime, null)
    LastAccess (datetime, null)
    UploadId (uniqueidentifier, null)
  Keys
  Constraints
  Triggers
  Indexes
  Statistics
dbo.CDN_Server
  Columns
    Id (PK, int, not null)
    Latitude (float, not null)
    Longitude (float, not null)
    FreeSpace (bigint, not null)
    IpAddress (nchar(50), not null)
    Host (nchar(50), not null)
    IsOnline (bit, not null)
    Name (varchar(50), not null)
    ServerRoleId (FK, int, not null)
  Keys
  Constraints
  Triggers
  Indexes
  Statistics
dbo.CDN_ServerRole
  Columns
    Id (PK, int, not null)
    Name (nchar(100), not null)
    Description (nchar(200), null)
  Keys
  Constraints
  Triggers
  Indexes
  Statistics
```

I will include SQL Script for creating tables (in case of something)

Database contains 3 tables

- CDN_ServerRole just describes server roles
- CDN_Server contains all servers, their host urls and disk space
- CDN_FileObject contains all object info, size, last access time, versionId, and server where it’s present
  Composite key Id + VersionId + ServerId

By the way, all projects are done on .NET Core 2.0 platform, so should be cross-platform ☺

To run app, you need install .net core sdk
Cd to folder with csproj file, then need to run via cmd ‘dotnet run’

Enjoy ☺