What I’ve learned creating .Net Core microservices app for Trucks

**EF Core**

* EF Core 5.0 doesn’t need a link table nor **HasMany** commands defined in code. It figures it out itself
* EF .**Include**() has linq methods such as **Where** so you can query based on the linked class (ie all trucks with category of 1)
* Newtonsoft.ReferenceLoopHandling needs to be defined in startup when u don’t create a dto to show theres not a cycle between linked classes of EF tables.

context.Trucks.**Where(t=>t.Categories.All(c=>c.CategoryId == categoryId)**)

* .Include(t=>t.Categories).**AsSplitQuery**().ToListAsync(); Here Im both querying a M2M relationship and splitting the query up for performance. (only works with related data)

**Logging**

* Logging: set it up in appsettings.json using ‘namespace’:’loglevel]

**Architecture**

* Duplication code sometimes is better in services then having as opposed to having a reference to a custom lib (such as for logging) as this causes coupling.

**AutoMapper**

* In startup: services.**AddAutoMapper**(AppDomain.CurrentDomain.GetAssemblies());
* **CreateMap**<Models.PhotoDto,Photo>().**ReverseMap**();

OpenApi

* <https://localhost:5001/swagger/v1/swagger.json> to get OpenApi client classes

gRPC

* Uses Http2 and binary serialization. Full duplex streaming. Needs to know all client details upfront. Good for internal API’s



**Asp.Net Core**

**Use, UseWhen, Map, MapWhen** and **Run** are used in Startup.Configure to setup the pipeline with steps(use), return(useWhen) and non return(Map) branching and terminate(run). Also UseMiddleware<T> to run custom middleware.