**Overall Solution:**

Firstly, to avoid over-engineering, I expand my idea step by step. I consider it as a Web-Application project that offers a UI presentation and web API endpoint. UI representation for end-user and webApi endpoint for the scenario which negotiates with other services.

**Solution components:**

I see just one responsibility which take a backup from SalesForce and upload them in Azure and Aws. I ask you one question, is there any possibility in the future that we want to add another uploader? Maybe after a while you want to persist it in GCP or others. There is a narrow border between over-engineering and having an extensible design which becomes clear after asking questions from business experts and having a good understanding of the business. If we have an assumption that it can be extensible, we can use the Open-Close Principle that lets us without changing the code we can put beside the project another upload provider. If we want to make a run-time decision to select one of them, then we can use a strategy design pattern.

Let me expand my scenario, from now on we suppose our app is an instantiated module in a Microservice Service Mesh System and it is hosted behind Api-Getway and end-users can access it by webUi and other Microservices according to their requirement call its endpoint. Then concurrency issues are possible and async programming is useful. Then we should pay attention that if we register a service that has a state, as a singleton then their states get shared among other requests.

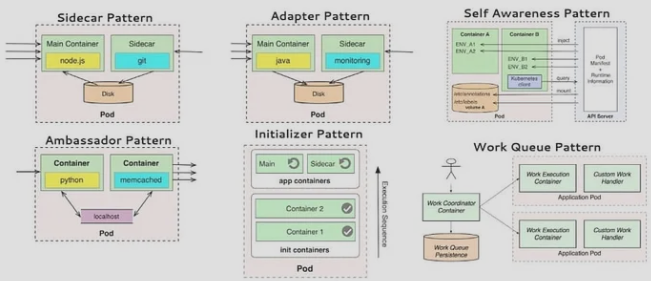
By following our scenario, we can provide a list of below features:

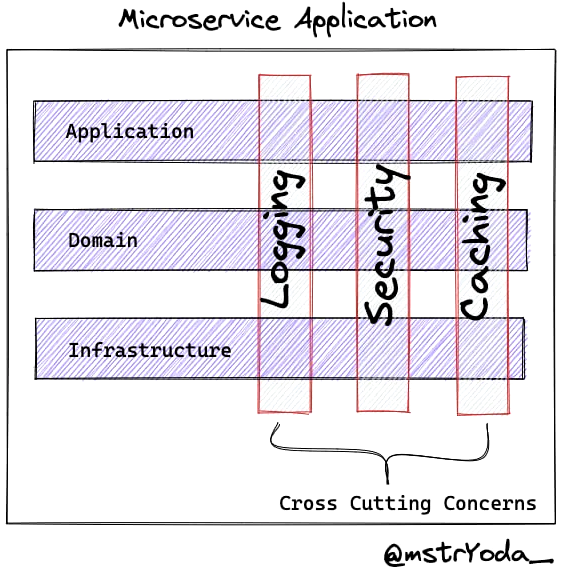
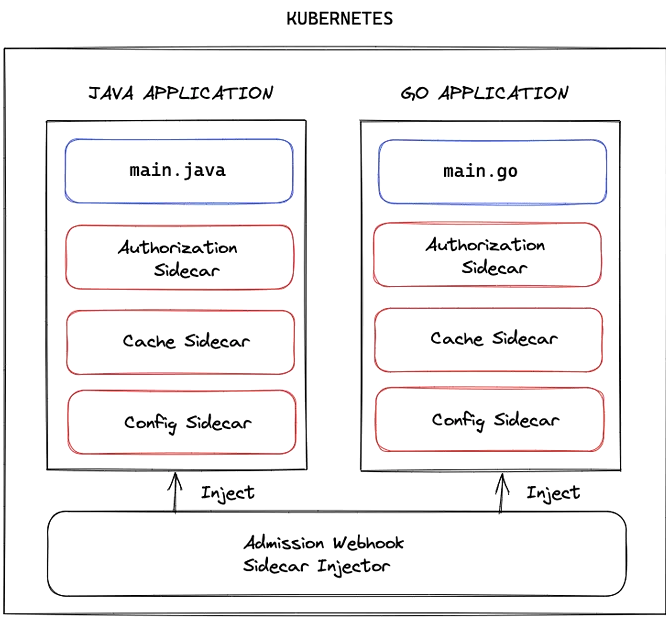
* a report of all ongoing backup processes and the history of all previous requests, enabling the customer to retry the failed ones. All ongoing processes can have an online progress bar.
* Represent some information about the duration of the backup process.
* Ability to define a schedule process which take a Cron expression from end-user. Using [Quartz](https://www.quartz-scheduler.org/)

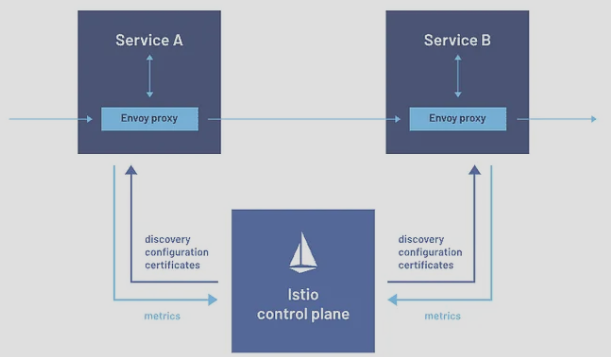
Also, we have some cross cutting concern such as:

* Audit log
* Activity log
* Application log
* Monitoring
* Authentication, authorisation
* Configuration management
* Automatic instantiation
* Security
* Cache management
* Circuit Breaker

By following our assumption, we have a microservice that is equipped with a Service Mesh System (Side-Car Pattern) then we have already had a solution for all of them except Automatic instantiation that Dockerization and Kubernetes can handle.





Also, if we use one of the Cloud Platforms such as Azure, it can provide us with many tools and facilities such as its log that causes having a full-featured dashboard for Monitoring.

**Backend Structure:**

By taking into consideration all the above points I will empower the backend side by .net 8 and use the advantages of:

* IHostBuilder
* BackgroundService which named it as a worker.
* Channel as an internal queue
* The worker listens to the channel.

1. **Request**: A request comes from UI by calling the backup endpoint which is located in BackupController, it is a Restful API then uses post-http-verb. And support versioning

Post httpVerb: <https://salesforceBackup.NewYorker.com/api/V1/backeup>

BackupCommand: {

SalesforceUsername

SalesforcePassword

SecurityTokenforSalesforce

Salesforcehostnameforyourorg

AWSaccesskey

AWSsecretkey

Azureaccountname

Azuresharedkey

}

1. **Processing Flow:** BackupController dispatch BackupCommand and in BackupApplicationHandler we have BackupCommandHandler which is responsible for handling this command and trying to download ScalesForcePage Content, then Put them in the Channel, then reply to the consumer that ”Backup Is Processing”. On the other hand, the worker is listening to the Channel, and as soon as a newly downloaded file comes, try to upload it to one of the Upload providers. When all files are uploaded, then push to the consumer that it gets finished.
2. **Scheduler**: Using Quartz for the scheduling mechanism which just needs to dispatch BackupCommand, it gets handled by BackupApplicationHandler
3. **ILogger**: If we use the could platform, ILogger is wrapped by the Could logger provider, Otherwise, I prefer wrapping it with Serilog, and making an Elastic configuration for Serilog, and then logging them in Elastic. Then using Prometheus for Monitoring.
4. **Business Flow form End-User Side**: list of End-User functionality

* make a Query on BackupRecord Object.
* Backup request
* Retry Backup Request

1. **Database**: for keeping the track of BackupRecord Object, correspondingly we have BackupRecordEntity. I couldn’t extract more objects in this domain that need persistence, we can impose more complexity on the business but keep it simple (KISS Principal). Per each backup request, we have one BackupRecord which has a state that show its status. And keep the data of the backup request, that can be used it for retry if the process fails.

**In** BackupApplicationHandler BackupRecordRepository is injected and as soon as the command is going to be handled, firstly we add it to BackupRecordRepository

1. **SQL server and EF Core:** I prefer using SQL server and EF Core, we decorate our command handler with a decorator in which the Unit of work is injected and calls Savechange() at the end of each method of the command handler. That guarantee has a transactional behavior in our command handler.
2. **Checking Idempotency:** BackupRecord has an Id property which is fed by the hash of BackupCommand instance, then we use this Id as an Idempotency key, and in our BackupCommandHandler we would check if we have an ongoing process with this Id or not, it means we only let one ongoing backup process is going on for specific id.
3. **Authentication and Authorization**: we can divide this subject into two different parts.

First, for the UI side, I suggest using [Basic Authentication in ASP.NET Web API](https://learn.microsoft.com/en-us/aspnet/web-api/overview/security/basic-authentication)

For restful Api Side, as we assume it is part of our service Mesh, generally they are equipped by SST for oAuth and OpenID such as using IdentityServer4. Then the only thing we need to implement is adding their middle-ware to our project and defining a new scope for our webApp in SST.

**Frontend Structure:**

There are a range of products and technologies that we can use.

* Asp net MVC MVC patteren
* Angular Web Mvvm pattern
* React Web Mvvm pattern
* VueJs Web Mvvm pattern
* Wpf Windows MVVm Pattern

I think MVC with SignaR as a push Notification satisfies our customer requirements. But if we want to have a Single Page App (SPA) I prefer using Angular