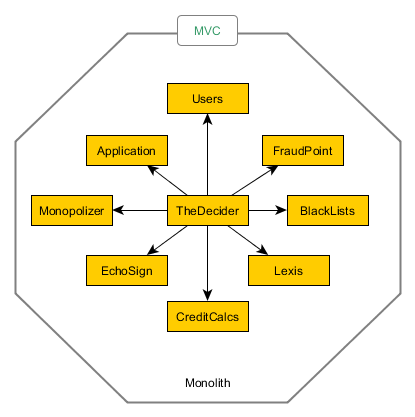
Development Plan

# Refactoring Order

1. ***Applications => Applications Microservice***
2. ***EchoSignServcie => Microservice Platform Build***
3. ***Monopolizer => Microservice Platform Build***
4. ***FraudPoints => FraudPoints Microservice (for integrations)***
5. ***BlackListIps => Black Lists Microservice***
6. ***Lexis => Lexis Microservice***
7. ***CreditCalcs => Credit Calcs Microservice***
8. ***The Decider = > Rules Microservice***
9. ***Login Microservice***

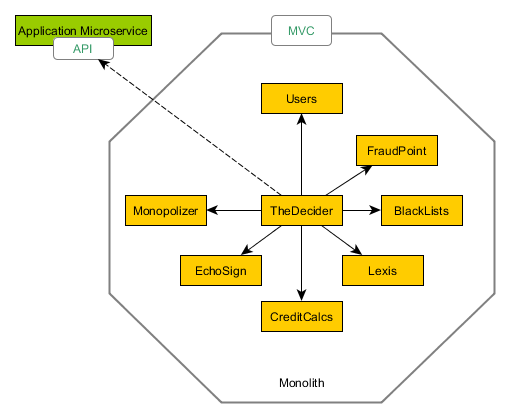
# Remove *TheDecider* coupling

The image below reflects the coupling between ***TheDecider*** and the rest of the modules.



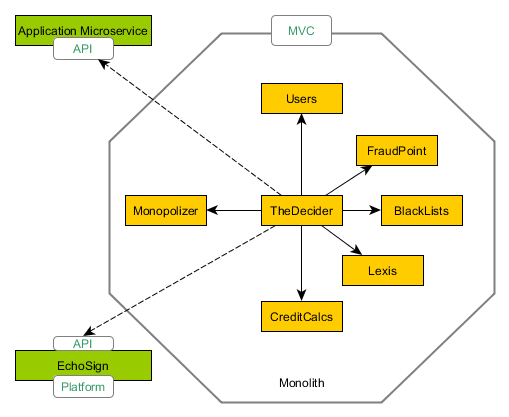
# 1 . Applications Modules to Applications Microservice

* Isolate Applications into its own microservice.
* Probably move CreditCalcs into Applications Microservice scope.



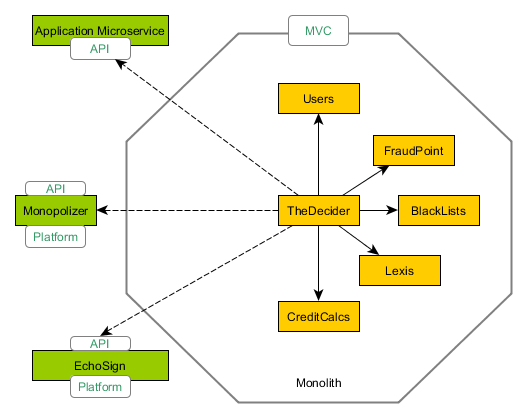
# 2 . Remove Echosign dependency on CommonConfiguration (Tight Coupling)

* This step includes removing monolith libraries from EchoSign and moving ***CommonConfiguration*** into a nugget package so every new service has not to have a submodule installed in its repository. That makes development easier.



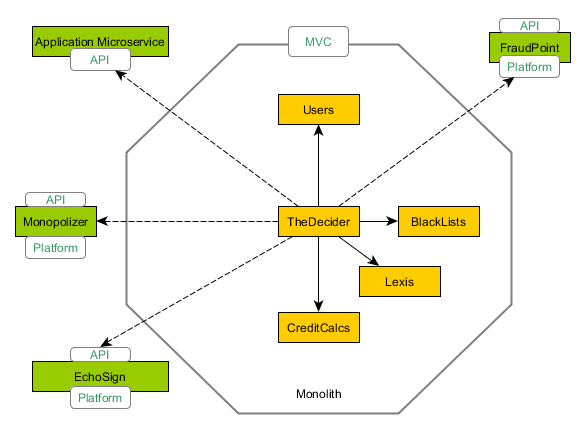
# 3 . Remove Echosign dependency on CommonConfiguration & CommunityChest (Tight Coupling)

* Remove ***Monopolizer CommunityChest*** Dependency
* Replace the reference to the **CommonConfiguration** submodule by the nugget package implemented in step 2.



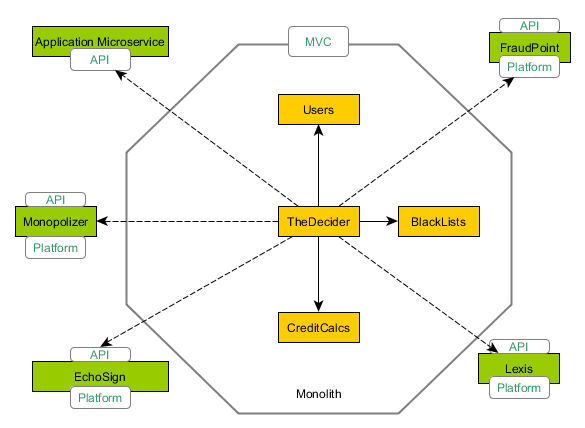
# 4 . Move FraudPoint to a Microservice

* This is a ***technical capability*** that represents “*integration*” and it should be moved out from TheDecider.



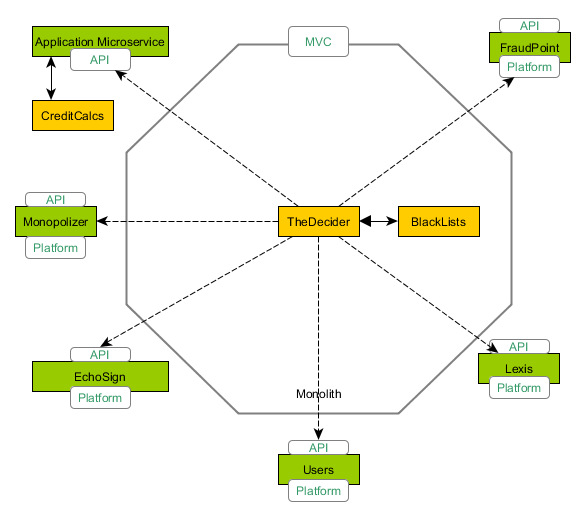
# 5 . Move Lexis Module to Lexis Microservice

* This is probably another technical capability that could be moved into a microservice.



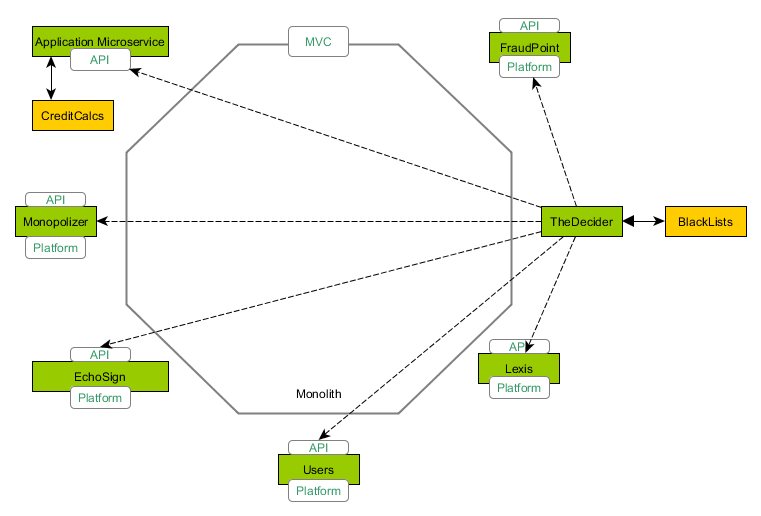
# 6 . Create Identity Microservice

* Review the ***Document [04 – Identity Microservice.doc]***



# 7 . Create Rules Microservice

* Remove ***TheDecider*** outside the monolith since all hard dependencies have been already moved. This releases the rules engine from the monolith.



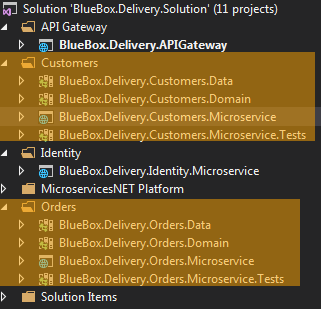
# Summary

* ***Applications*** w/ ***CreditCalcs*** moved into a Microservice.
* ***Echosign Microservice and Monopolizer*** are refactored to use a nugget package.
* ***FraudPoint*** integration is refactored into a Microservice.
* ***Lexis*** is refactored into a Microservice.
* ***Login Microservice*** is created.
* ***TheDecider*** is released from all dependencies and is moved into a Microservice.
* ***The monolith started becoming smaller. The expectation is to convert the MVC project into an API Gateway.***

# Project Structure

Every Microservice should have:

* An own Data layer
  + Commonly will use ***Repository Pattern*** with Concrete Types and ***UnitOfWork*** to manage ACID transactions.
* An own Database (Any type is just fine: NOSQL, SQL, etc.)
* An own Domain Layer
  + Refer to DDD way.
* An own Unit Tests Layer
  + Using xUnit
* A REST API
  + This will be used to establish an HTTP communication. Keep it as small as possible so in the future if we just want to move it into an Even-Driven architecture it just replaces that layer. Keep the business into domain as much as possible.

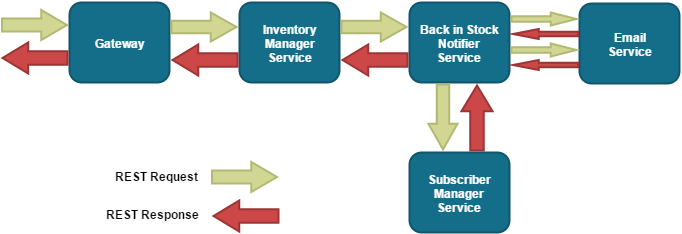


# Application Point-to-Point Communication

During modernization phase the application is hosted into two cloud solutions:

* *The monolith lives in RackSpace.*
* *The microservices system lives in AWS Amazon.*

In order to keep that integration we will continue using HTTP protocol for point-to-point communication. This will produce a temporal coupling between services since we require creating client classes in order to establish communication with other microservices.



# Application Message Broker Communication

Once the application leaves the monolith a perfect refactoring could be change the communication channel into an Event-Driven microservice system. This will remove any coupling, services do not know about the other services.

* Gateway is REST
* All on-premises microservices will keep communicated with Events using a Service Broker like RabbitMQ/NServiceBus/AWS SQS.

