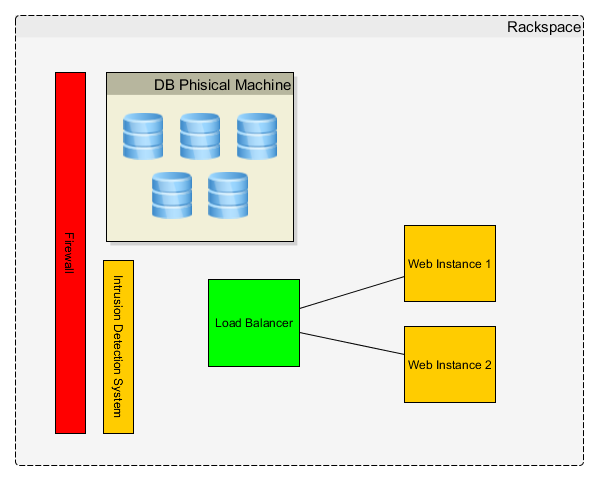
Bristlecone Architecture

# Deployment (*Rackspace Section*)

Currently Bristlecone Application is running in Rackspace private cloud behind a firewall and an IDP (Intrusion detection system).

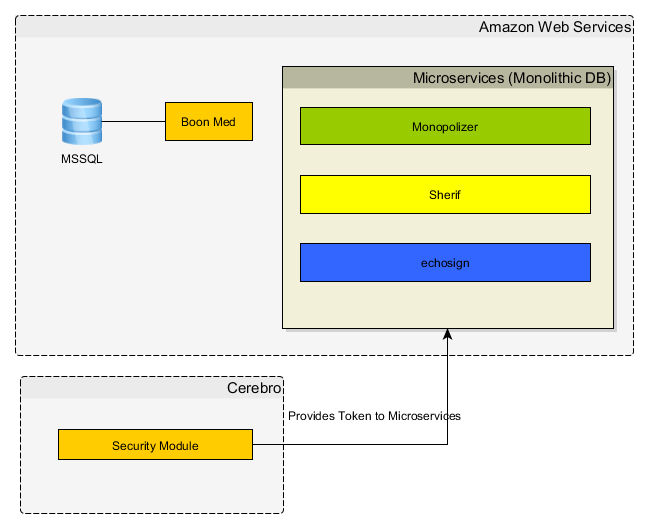
Rackspace provides a physical dedicated machine for all the DBS for the different domain the application has.

* *Medly database =* ***E***
* *Wags database =* ***A***
* *Bristlecone database =* ***R***
* *One Road =* ***N***
* *I Do =* ***D***



# Deployment (*Amazon Web Services Section*)

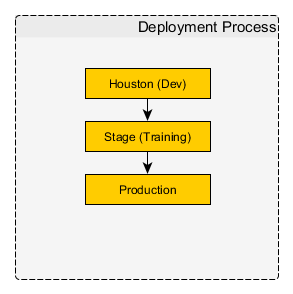
A fraction of the system was moved into AWS infrastructure including three micro services that were moved out from the monolithic code base. The application requests token to ***Cerebro*** ***security module*** in order to access micro services in AWS.



# Continuous Integration & Continuous Delivery

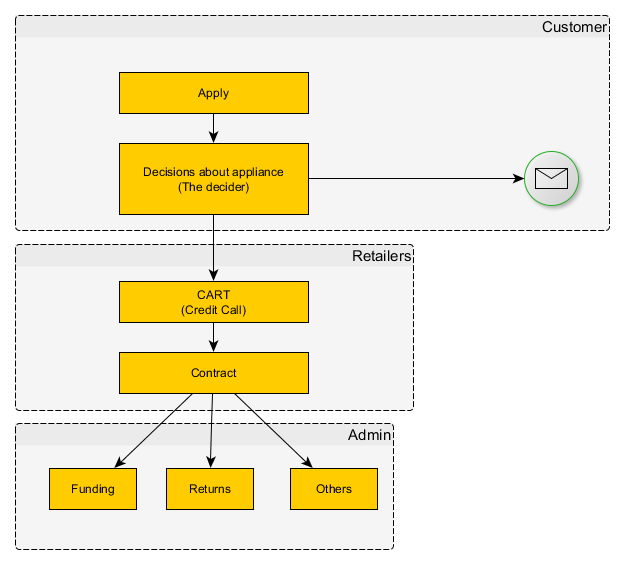
Continuous Integration is working based on ***Team City CI*** for ***bristleconefinancing*** project main branch.

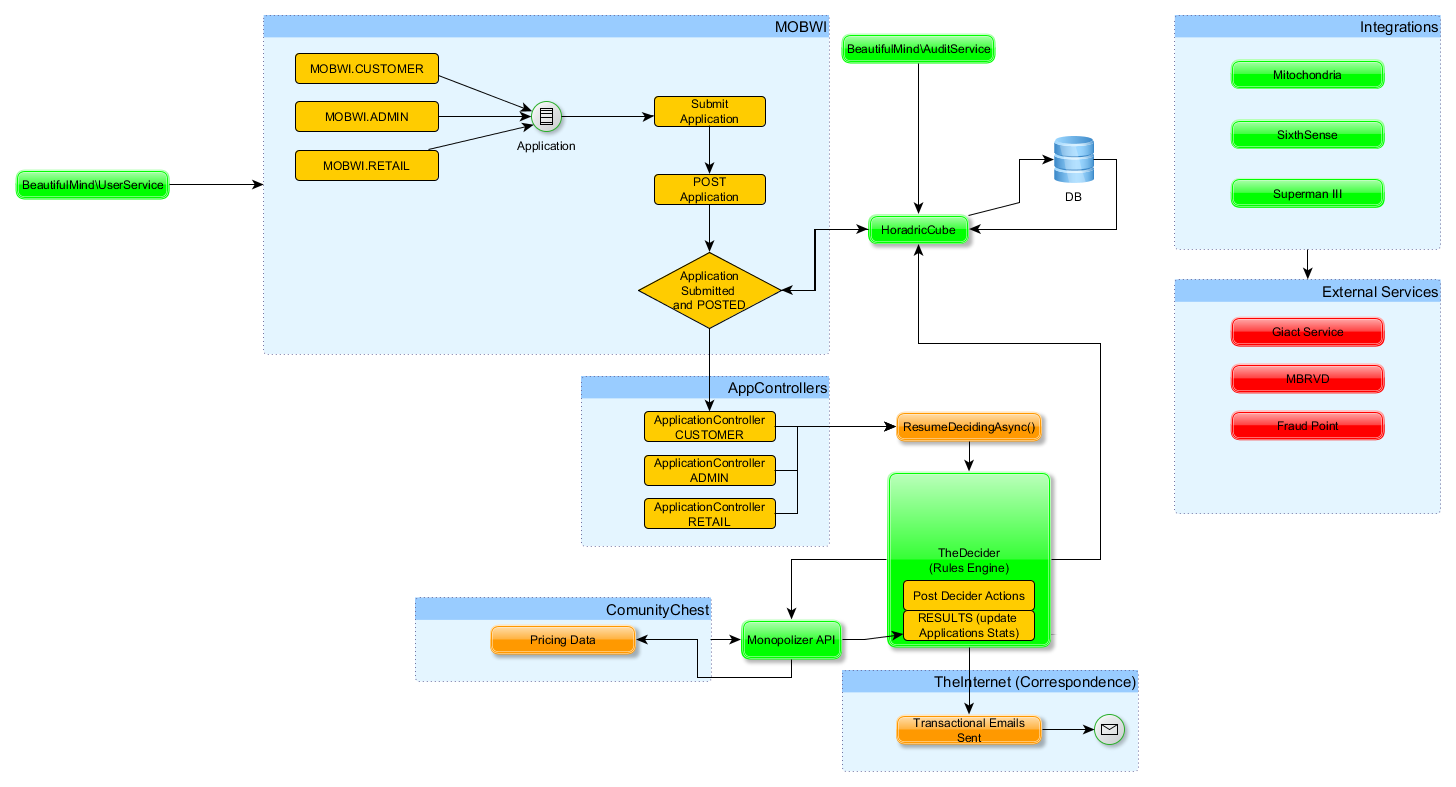
Continuous Delivery is done using ***Octopus*** following the flow below***.***



1. Houston is deployed automatically right after the CI in Team City is finished.
2. The package then is promoted manually to the next environment Stage and finally to Production.

# Application Big Picture



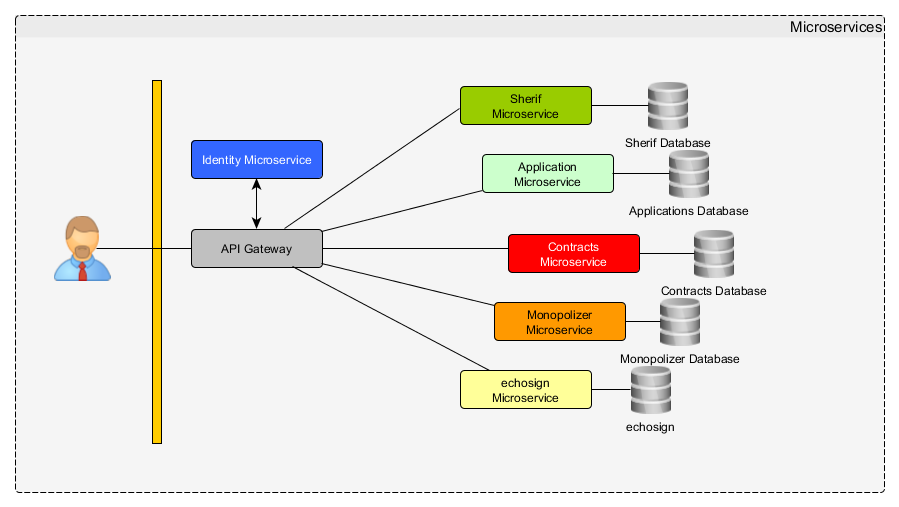


# Monolithic to Micro Services Architecture

Moving from *Monolothich* to *Microservices* will require some incremental architecture changes in order to understand and finally isolate the different business capabilities. So far the business/technical capabilities detected are:

1. Create Applications (Applications Domain -> ***Applications Microservice***)
2. Create and Operate with Contracts (Contracts Domain -> ***Contracts Microservice***)
3. A technical capability for Identity (***Identity Microservice***) which is useful to allow delivering token to authored client parties specified.
4. Refactoring already existing micro services to have their own databases.
   1. Monopolizer
   2. Echosign
   3. Sherif

The final micro services architecture could looks like the picture below:



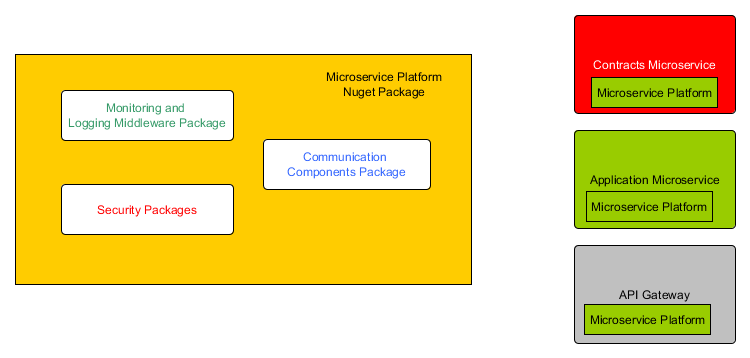
The micro services we need to focus are:

1. ***API Gateway:*** The API Gateway acts like an adapter between applications and the system of micro services. It usually covers all of the system functionality by delegating everything to other micro services.
2. ***Identity Microservice:*** This is a micro service that provides tokens to the API Gateway and also provides security ***microservice-to-microservice*** by using scopes.
3. ***Application Microservice:*** This creates application for a customer. This operates with Application Domain Entity and its business invariants.
4. ***Contracts Microservice:*** This operates with Contract Domain Entities and its business invariants.

# Monolithic to Micro Services Architecture (.NET Micro Services Platform)

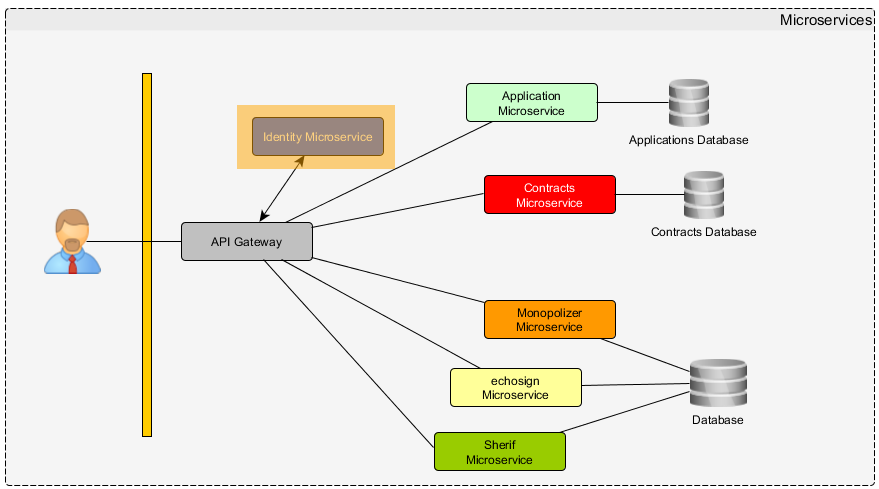
In order to have a reusable .NET Micro Services Platform so developers could have a common infrastructure to work on we can use a Nugget packages an upload them into a local Nuget server.

1. Monitoring Middleware
2. Correlation Token Middleware (add a correlation token from API Gateway if not exists in the request)
3. A client factory class to adding common headers to the outgoing requests
4. Data Access Base Classes like repositories interfaces, etc.
5. Owin Classes if needed



**A tiny server could be required from Bristlecone Holdings to host a nugget server.**

# The Identity Micro Service



The ***Identity Micro service*** is in charge of providing *Tokens* to the interested parties (i.e. clients) it hosts users, resources and clients. It also owns a user’s database.

Clients could be a web client (like a SPA) or micro services trying to communicate other micro services.

The API Gateway could use it to generate a cookie or just return a JWT (Json Web Token) to the application which is usually saved in the local storage.

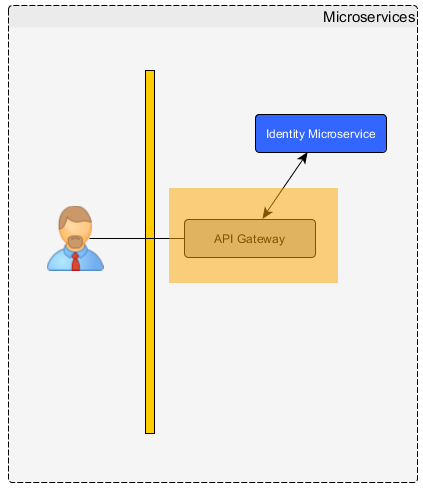
A micro service could ask for a token to access other micro service as well as validate incoming requests.

Live Samples:

<https://www.screencast.com/t/eUONhgyIU0>

<https://www.screencast.com/t/mEruMPdfJ>

# The API Gateway (Public API)



The API Gateway acts as an adapter between applications and the system of micro services covering all system functionality. It delegates everything to other micro services. The application only needs to know about it. Some disadvantage it presents it is that tends to grow quickly.

# The Applications Micro Service

It’s the micro service responsible of creating **Applications** from Customers and maintains its own database.

* ***(TBDs below are important to discover additional micro services. Remember that micro services should not be too big, it’s common at the beginning to start bigger then refactor to smaller).***

***Tables to own:*** TBD

***Micro services Dependencies:*** TBD

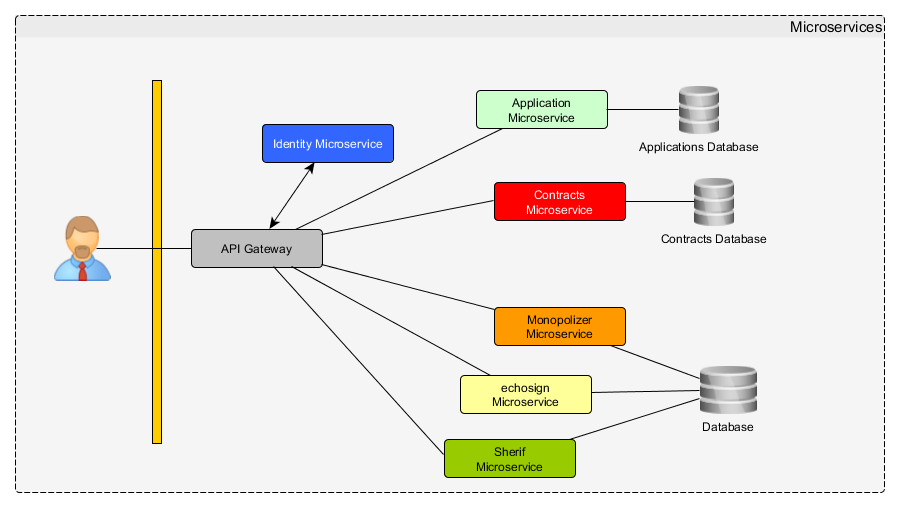
# The Contracts Micro Service

It’s the micro service responsible of creating ***Contracts*** from Customers and maintains its own database.

***Tables to own:*** TBD

***Micro services Dependencies:*** TBD

# Final Architecture after Separating Applications/Contracts



# Database and Data Access

We can continue using Entity Framework (given the application will be built on .NET Core we will have to use Entity Framework Core 1.0 which does not have all full features from the full .NET core version).

***Framework***: <https://docs.microsoft.com/en-us/ef/core/index>

# Notes

I see additional micro services candidates like:

* *Correspondence*
* *Might be the Hang Fire stuff (I have to study deeper them)*
* *Audit (Part of the BeatifulMind Service)*
* *Users (Part of the BeatifulMind Service)*
* ***TBD***