ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

The following are the most Architecturally Significant Requirements sorted by Priority

“The expected number of concurrent users are in the tens of thousands with big surges around the major holidays”

“The switch between the existing system and the new one should be as incremental and as seamless as possible”

“Recently they faced a DDOS attack so the system robustness (availability, scalability) is crucial for them along with observability and alerting to be able to instantly pinpoint and address such issues”

“All site traffic data should be collected and analysed for insight regarding user behaviour popular flights and opportunities (e.g: adjusting margins) in addition to audit and legal purposes”

“As they are building a platform FunWithFlights would like to expose their API for re-sellers”

QUALITY ATTRIBUTES

The following are the most important Quality Attributes found in the Architecturally Significant Requirements

PERFORMANCE / SCALABILITY: The system should keep responding without further delays specially during peak times, if this is not fulfilled the company could afford potential user attrition and revenue.

COMPATIBILITY/CO-EXISTENCE: The migration to the new cloud paradigm should be 100% controlled to avoid system interruptions that could generate revenue loss.

RELIABILITY / AVAILABILITY / ANALYZABILITY: System should be very robust and remain stable even under DDOS attack conditions, if system is not available at some time, there will reputation and revenue loss. Also it is important to have a good trace information in order to react to real-time issues very quick.

LEARNABILITY: To find possible revenue increase opportunities the system should be able to learn from user insights and extract meaningful information.

INTEROPERABILITY: Another revenue source comes from external resellers who can use our curated information and sell flights to their users, therefore the system should provide a interoperability API for them

MAJOR ARCHITECTURAL DECISIONS (part 1)

CLOUD NATIVE SOLUTION: First of all, there are great advantages in implementing this solution as Cloud Native such as ready to use components (storages, analytics, monitors, firewalls, etc), but the most important one is the possiblity to perform resource auto-scaling with no or very little effort. Also having public exposed endpoints allow the company to implement data reselling as an alternate income source.

AZURE STACK: The presented architecture can be implemented with very little modifications in any of the major cloud providers, however, I am suggesting Azure because currently the company is familiar with Microsoft technologies (*Monolith code is implemented in .NET*) and also they have already a Enterprise deal with Microsoft which greatly reduces the cost of Azure implementation vs Amazon or Google.

MICROSERVICES ARCHITECTURE STYLE: I chose a microservices approach because it allow us to scale out invidividual services in order to adjust to load peaks, also we can separate persistence by different services allowing us to use polyglot persistence better suited for each case. Additionally it help us to decouple domains and improve development life cycle efficiency and independent CI/CD pipelines which will speed up feature releases.

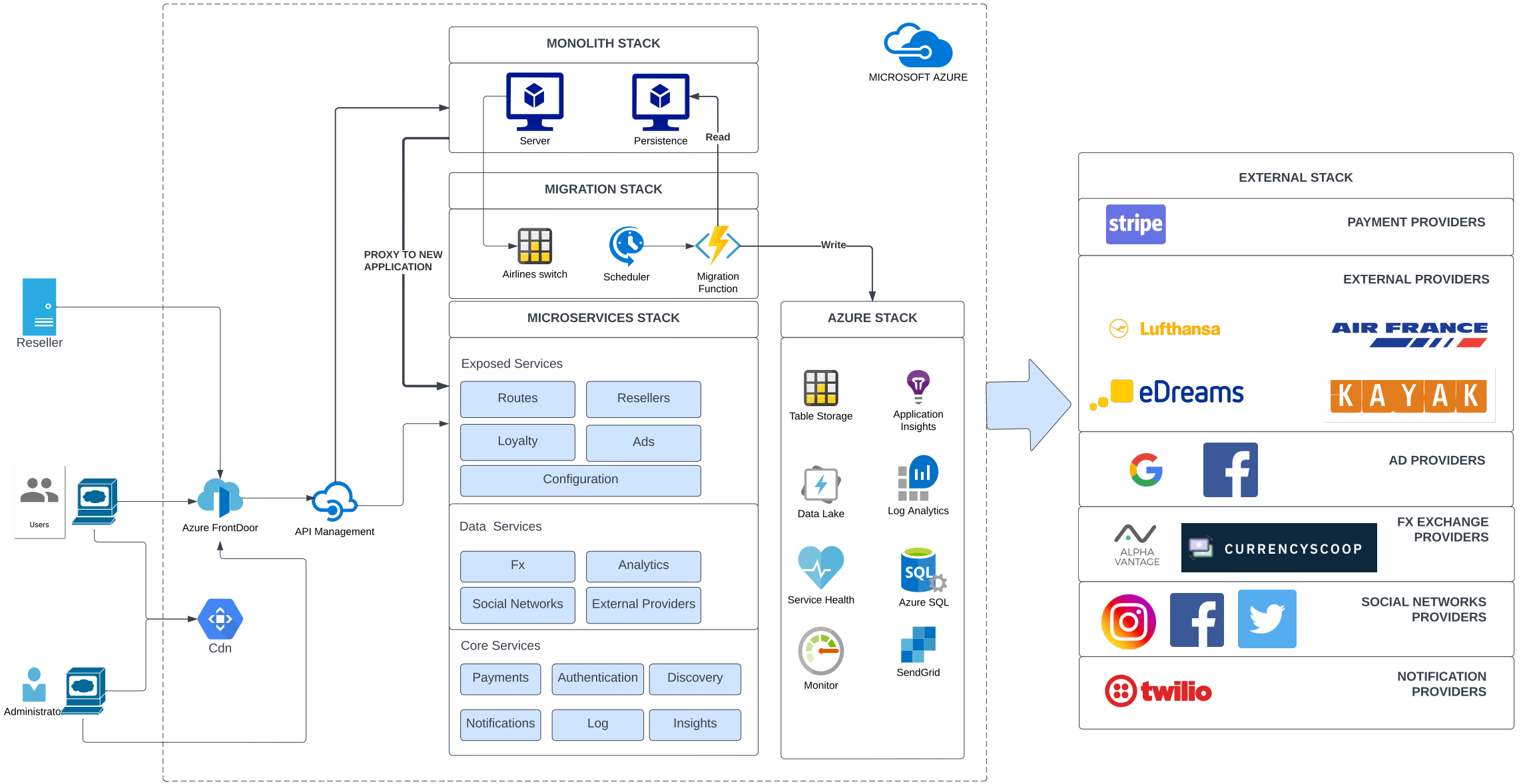
MAJOR ARCHITECTURAL DECISIONS (part 2)

SECURITY LAYER: Having a public exposed service with DDOS attack history is an important risk that needs to be assessed into this solution, fortunately we can rely on our Cloud Provider to add an extra protection layer that will keep the business safe.

API MANAGEMENT LAYER: Having different users and two main applications (*monolith and new solution*) to interact with; this middle layer adding authentication, authorization, redirection and cache features adds a lot of value and simplicity to our proposal and increases compatiblity and interoperability quality attributes.

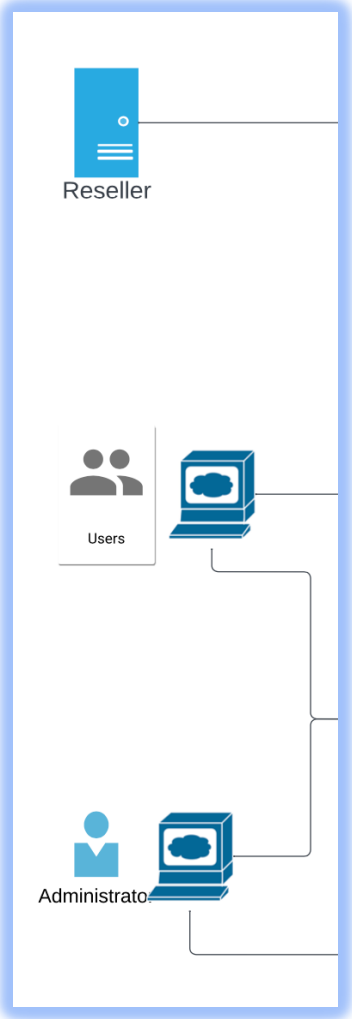
ANALYTICS LAYER: I've added an additional service that will process insights data (*coming from all different services*) and generates business reports helping the managers to create value add proposals (*e.g. launch marketing campaings at a given location, or increase/decrease rates for specific routes*)

Fun with Flights – ARCHITECTURE DIAGRAM



Fun with Flights – DOCUMENTATION

USERS



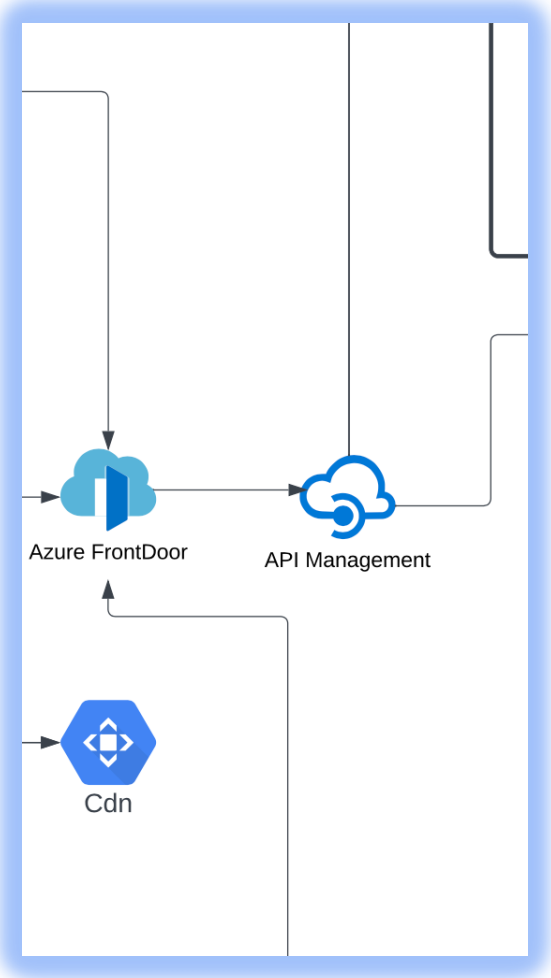
REGULAR USERS: Can use the system anonymously to search for cheap flights, they can adjust

REGISTERED USERS: All functions from regular users plus some extra features such as loyalty program level, discounts and benefits (*also they have a specialized chatbot to support enquiries and suggest special promotions*). Also a user can get a voucher of 10% when they need to cancel their non-refundable flights in advance. Our system can communicate with them using different options such as e-mail, SMS or WhatsApp (*all of this can be configured in the preferences section*)

ADMINISTRATORS: They can configure different parameters such as different flight providers and their configuration (*such as profit margin*). Also they can manage resellers and parameters such as their monthly quote and margin.

RESELLERS: They are API users who don't interact with the portal and invoke directly our Resellers Service via well documented REST API (*using level 3 Richardson maturity model*)

AZURE ENTRY COMPONENTS



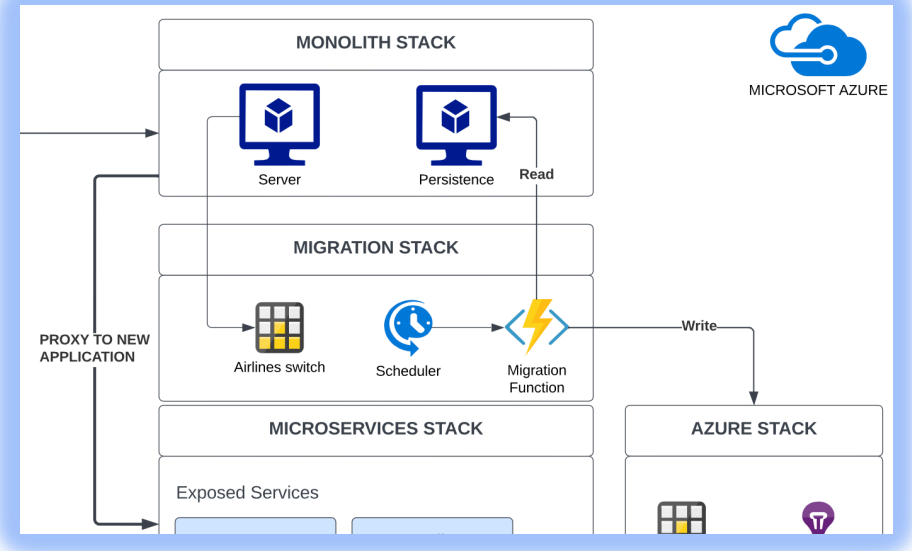
AZURE FRONT DOOR: This component adds protection against DDOS attacks and many others thanks to its state-of-the-art Web Application Firewall. This component is connected to Azure Monitor in order to troubleshoot and pinpoint security and access problems.

CDN: This a standard CDN with basic protection against DDOS offered by Azure by default, this component helps to faster loading times at different world locations by offering static cacheable resources.

API MANAGEMENT: Allow unified access to different microservices and also to old monolith servers, once the migration phase is finished we will just need to modify a configuration here and the switch will be completed.

This component also helps to manage authorization so resellers can only access Reseller service, and can cache some dynamic responses based on some parameter configuration (*e.g. cache all routes for same search for minutes*)

AZURE MIGRATION STACK



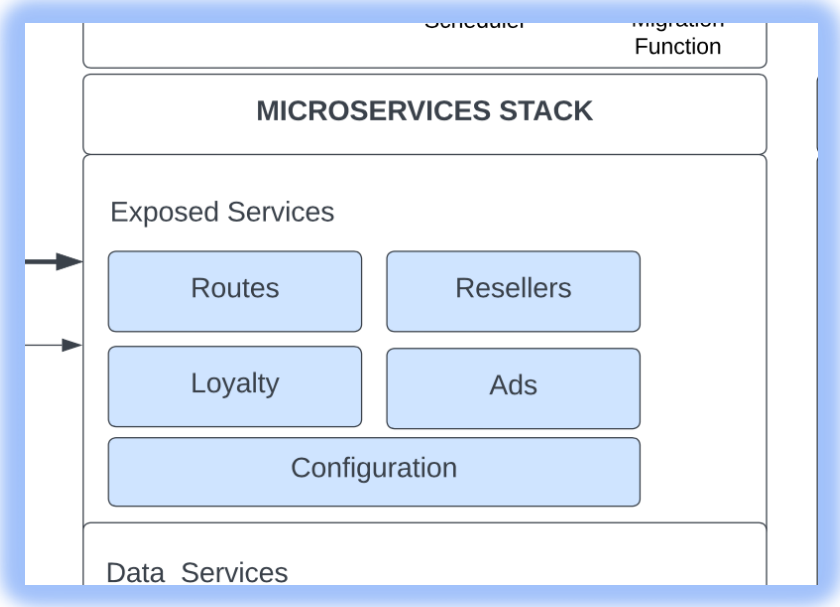
MONOLITH STACK: Current server traditional implementation using an app server and a persistence server. There are some small modifications that need to be done:

- Move on-premise servers to Azure (*or use Azure hybrid cloud*) and for application server we should use scale sets to provide IAAS scalability.
- Integrate with airlines switch table storage which will define if a specific external provider request should be managed by the monolith or by the new implementation. This will allow us to start a pilot and then increase coverage until migration is finished.

MIGRATION STACK: In addition to our airlines switch table storage, we have here a scheduler that will trigger regularly the Migration function that executes transformation logic between monolith persistence and new Cloud native persistence.

Once the migration phase is finished all these components can be safely decommissioned

MICROSERVICES STACK – EXPOSED SERVICES



This layer represents the outer layer services in the new implementation and it's composed of the following services:

ROUTES SERVICE: Invokes External Providers service and aggregates information, this is one of the main services which are autoscaled out based on load and usage rules.

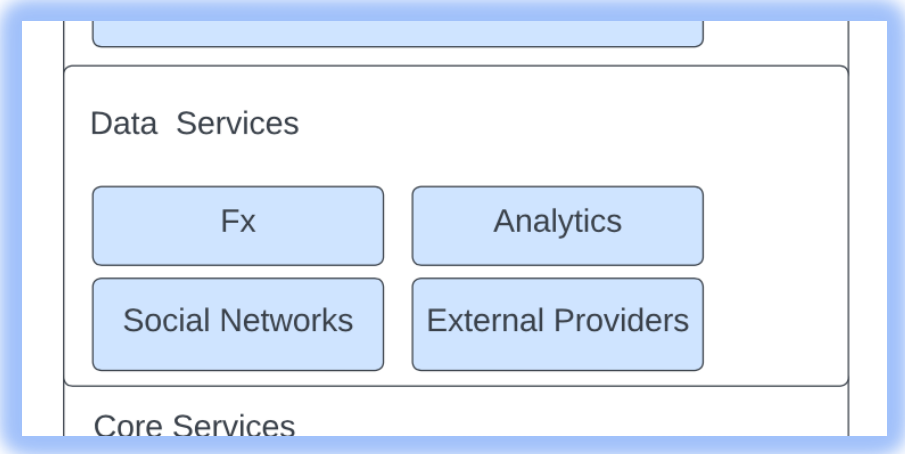
RESELLERS SERVICE: This is a lightweight service that acts as a communication layer between resellers and Routes service, it also manages quote usage and can increase margins based on configuration.

LOYALTY SERVICE: This service manage all request related to registered user loyalty data, levels, promotions, discounts and chatbot communication.

ADS SERVICE: Very lightweight service that defines exactly which ad information will be displayed in the portal.

CONFIGURATION SERVICE: Manages all configuration and parameters for the application, including resellers, providers and margins.

MICROSERVICES STACK – DATA SERVICES



This layer represents the data consuming services in the new implementation and it's composed of the following services:

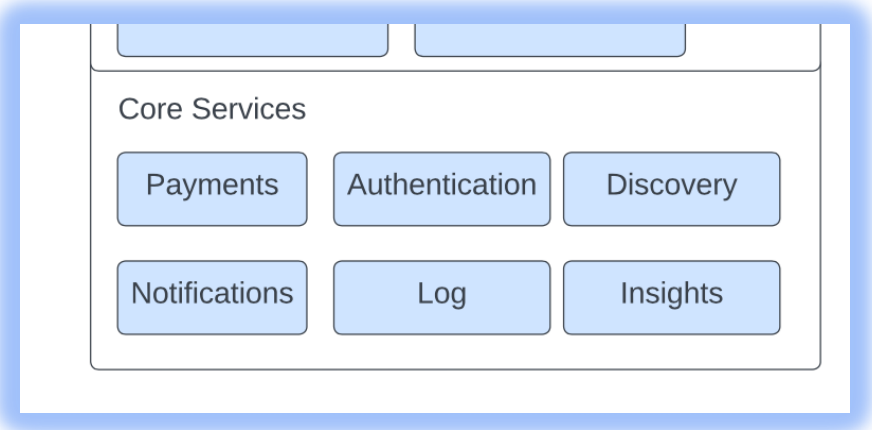
FX SERVICE: Returns recently cached information regarding money conversions in order to display quotes in user's currency. (*recalculates every hour*).

ANALYTICS SERVICE: This is a batch worker service that constantly process insights information to generate daily reports reflecting most popular flights, usage trends, etc.

SOCIAL NETWORKS SERVICE: Lightway service that allow loyalty service to publish promotions regarding last minute ticket resell opportunities and attract customers.

EXTERNAL PROVIDERS SERVICE: This is a core service that can return quotes and fares from one external provider at a time. It can run several requests in parallel to speed up processing. Given this is on of the most important services, it has autoscale rules that span several instances of the service at the same time to adjust for example to holiday peaks and then scale down rules when resource consumption is low.

MICROSERVICES STACK – CORE SERVICES



This layer represents the baseservices used by the upper layers, and it's composed of:

PAYMENTS SERVICE: Interacts with payment brokers such as stripe (which offers all the payment options we need).

AUTHENTICATION SERVICE: This service provides user information, oAuth tokens, claims, etc.

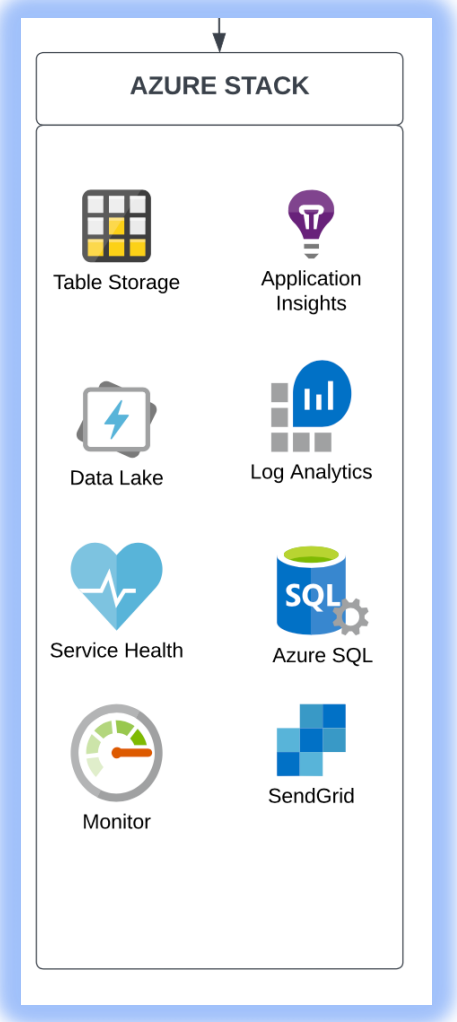
DISCOVERY SERVICE: This service records all service endpoints and act as a mediator between different services by providing actual service urls and decoupling service configuration. It also allow developers to easily run some services locally and some services on the cloud.

NOTIFICATIONS SERVICE: Lightweight service that connect with notification providers such as Twilio.

LOG SERVICE: Core service that keeps tract of all requests (*through correlation id*) and their interaction through all application services, by storing all traces eventually in Log Analytics

INSIGHTS SERVICE: Another core service that add contextual information to user behaviour data that is stored in application insights

AZURE STACK



This layer represents the main azure services used in the solution, which brings added value as a cloud-native solution:

TABLE STORAGE / AZURE SQL: Provides persistence for different services (*each service has their own persistence models*)

APPLICATION INSIGHTS / SENDGRID: Stores user behaviour information that is analyzed later by the Analytics Service, once the information is processed reports are sent using SendGrid and then informationis moved to DataLake.

LOG ANALYTICS: Repository for trace data, after some time it is moved to Data Lake

DATA LAKE: Information here is stored up to 5 years and can be used for any legal or audit purposes.

SERVICE HEALTH: Keeps track of different components health status, it provides alerts and incident management.

MONITOR: This component provides query capabilities over trace information stored in Log Analytics, security reports, etc.

EXTERNAL STACK

This layer represents the external services interacting with the new application:

PAYMENT PROVIDERS: Stripe is a world leader payment provider that allow us to use VISA/MASTERCARD/AMEX,APPLE PAY, GOOGLE PAY, PAYPAL and even Cryptos.

EXTERNAL PROVIDERS: Selected airlines or external providers offering fare informations, this also includes new small mergers.

AD PROVIDERS: Possible Ad providers who can post advertisements in our site

FX EXCHANGE PROVIDERS: Different external APIs providing up-to-date currency exchange rates.

SOCIAL NETWORK PROVIDERS: Set of trending social networks to post last minute promotions (user canceled trips)

NOTIFICATION PROVIDERS: Twilio is an All-in-one service providing e-mail, sms, Whatsapp notifications and more.

Fun with Flights

THANKS!