Societe Generale speeds real-time market quotes using Azure Service Fabric

April 29, 2019

Authored by Morgan Conti and Jean-Martin Zarate from Societe Generale UK, in conjunction with Eric Grenon from Microsoft

Digital technology is at the heart of Societe Generale's mission to serve clients and to advance the economy. Every day, it serves 31 million individual clients, businesses, and institutional investors around the world by offering a wide range of advisory services and tailored financial solutions. As part of the company's digital transformation initiative, Societe Generale is expanding the role that Azure Service Fabric (https://azure.microsoft.com/en-us/services/service-fabric/) plays in its environment.

This article is part of a series about customers who've worked closely with Microsoft and utilized Service Fabric over the last year. We explore why they chose Service Fabric and take a closer look at the design of their applications.

In this installment, we profile Societe Generale and the design and architecture of its market dataset platform, called Geysir.

Scaling markets in real time

As part of its company-wide digital transformation, Societe Generale is structuring IT developments as reusable building blocks exposed through APIs as services. Among these, a financial market dataset platform, code-named Geysir, provides quotes describing financial instruments to several internal clients. Geysir can be thought of as a market provider, where a *market* is a coherent set of financial data, such as the price of gold, the conversion rate of a dollar in euros, or the London interbank interest rate. The custom markets that Geysir provides can have different currencies, instruments, cut-offs, and other mathematical model parameters.

Several teams are using Geysir for different purposes. Traders use it to provide real-time market data to price financial products. The middle office needs snapshots of a market to value a portfolio or compute its risks and profit and loss. Financial engineers use it to retrieve past values to back-test strategies.

Geysir's predecessor was designed to distribute a single, real-time market in a push workflow. As more internal tools began to request more quotes from this platform, the team was faced with a growing need to generate customizable markets. Although the platform was stable, it wasn't designed to keep pace with multiple custom market quotes. That's when the team began gathering requirements for Geysir, as a new solution with greater capacity, flexibility, and scale.

Today, a single snapshot market builds approximately 30,000 instruments. But the forecast is to have at least 30 different snapshot markets building more than 40,000 instruments each. In a real-time market, 218,000 instruments are built continuously—and this number will reach 1 million instruments by the end of the year.

Markets as a service

Written in C#, the predecessor of Geysir was a set a microservices designed to offer an API to client applications for composing a market from different data sources with customized refresh rates. Some services were dedicated to retrieving live quotes, transforming them to the company's internal model, and pushing the transformed model to a cache (MongoDB). Other services, at a given interval, were picking the

new data from the cache, transforming this data to the clients' format, and then publishing the result to the clients.

The team had to design a new solution to address the shortcomings of the former platform:

- Fault isolation. The old architecture was not truly isolated. Some connectors were in the same services, and a failure of one affected them all.
- Reliability. Only one instance of the services was deployed on a single virtual machine, without any orchestration mechanism.
- Scalability. The single-service microservices architecture had no way of managing scalability other than scaling up the machines the services ran on.

The team also wanted to add a set of new features, such as:

- Segregation of business capabilities and support for multiple custom markets. The new implementation needed features for market data retrieval, transformation processing, and a filtering and querying engine.
- Zero downtime upgrades to ensure a strong service-level agreement (SLA).
- Platform consistency. The goal was to build a consistent microservices ecosystem and to create a single platform capable of deploying, running, and scaling multiple versions of the application, side by side.

Service Fabric as a platform for microservices

To rearchitect the system, the Geysir team needed a new approach, but the developers didn't have the time or bandwidth to gain expertise on distributed cache and all the other necessary technologies.

They consulted with another development team at Societe Generale, one which was responsible for a financial simulation platform (https://customers.microsoft.com/en-us/story/societe-generale-complex-financial-simulation-platform-expands-on-azure-service-fabric-architecture) . This platform was the main Geysir client, relying on Geysir data for its computations, and was built using Service Fabric. The simulation platform team showed the Geysir team what Service Fabric could do, and it looked like the natural solution to meet the design requirements. Service Fabric was compatible with the large, existing C# code base.

Even though there was no requirement for Geysir to use the same technology as its main client, the platforms would continue to work closely together. Both teams could benefit from sharing knowledge and components.

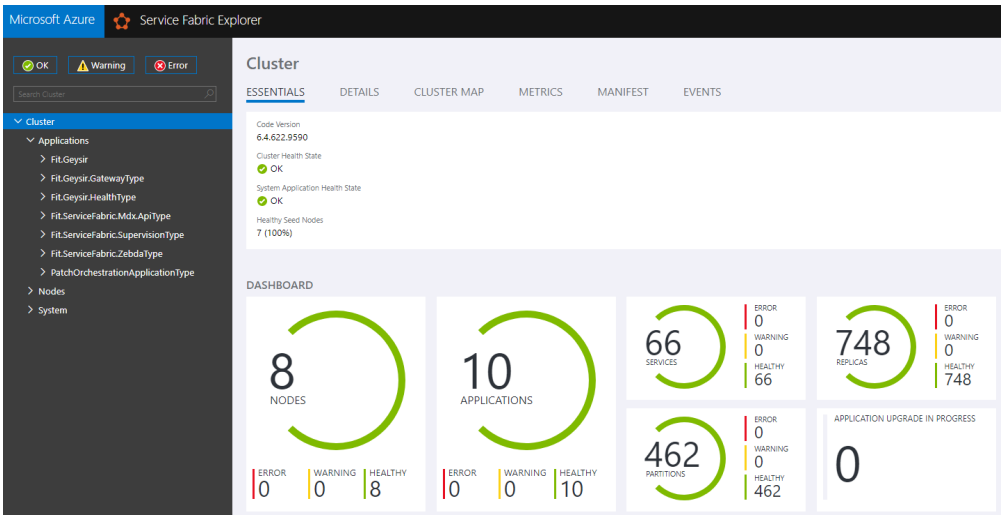
Without Service Fabric, the team would have to learn and deal with:

- Service orchestration (using, for example, Swarm and Kubernetes).
- Microservice communication (such as through Fabio and Consul).
- Microservice monitoring and health (using Consul).
- Distributed cache (using Redis).
- Key value store (in MongoDB or Azure tables).
- Queuing (through RabbitMQ or Azure Service Bus, for example).

With Service Fabric, the team can create Service Fabric clusters on any virtual machines or computers running Windows Server. For a team of three who were supporting a legacy application and developing its successor, the fact that Service Fabric manages most of the complexity involved in developing and managing distributed applications was a big bonus.

For security reasons and because the application is for internal use only, the Geysir team set up an on-premises cluster of virtual machines to host the applications. The production cluster has eight large nodes in three locations, with 10 applications, 60 services, and more than 700 replicas.

> " It is easy to introduce a new technology but more complicated to master it, maintain it, and transmit the knowledge as employees come and go. This was especially challenging for our small team, where the departure of a person would have a big impact on the project. With Service Fabric, we leverage our C# knowledge, and people who may not know anything about it can quickly be onboarded and add value."
>
> —Morgan Conti: Developer and project manager
> Societe Generale UK



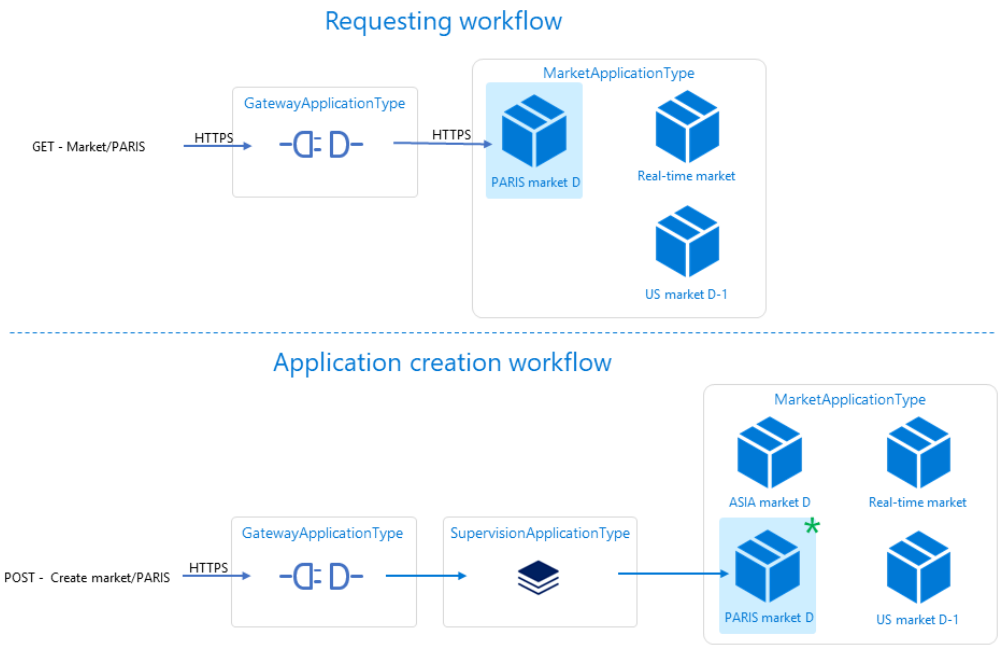
Geysir architecture

The cluster hosts the following applications:

- **MarketApplicationType.** An instance of this application represents a market. It stores how the market is defined and refreshed, and it requests and stores the quotes that are used to generate the market. Each instance also has its own API to expose data, but the endpoints are restricted to the cluster and not exposed to client calls.
- **GatewayApplicationType.** This application unifies the multiple APIs into one that proxies the requests to the desired instance of MarketApplicationType. This allows the team to have a unique public endpoint for the API and yet provide various markets with multiple versions for their clients. It is the unique entry point for the applications and services on the cluster and manages authentication and authorization for all API calls.
- **SupervisionApplicationType.** This application exposes an API to list, create, and delete instances of MarketApplicationType on demand. SupervisionApplicationType instantiates a new instance of MarketApplicationType with the click of a button or with an automated call, or it provides the list of available markets.
- **Guest applications.** Several guest applications are used to port existing data source connectors into the cluster. The team used the Service Fabric functionality to quickly and easily elevate the connectors to the same level of availability, reliability, and scalability as the rest of the applications, without having to rewrite any code.

The key to supporting multiple custom markets is the MarketApplicationType application. While designing it, the team took advantage of several Service Fabric features. Reliable Collections proved helpful because it enabled the small team to write highly available, scalable, low-latency cloud applications as though it was writing single process applications. Developers can program to the Reliable Collection APIs and let Reliable Collections manage the replicated and local state.

All services in MarketApplicationType are partitioned to ensure a good distribution of the load, and most have multiple replicas or instances.



Stateful services

Reliable Collections are used to manage the workflow of the MarketApplicationType application, ensuring that the availability and reliability goals for Geysir are met. If a node goes down, a service managed by Service Fabric is moved, or some other event happens, the impacted service resumes from the point it left off with its previous state.

Service Fabric offers a stateful programming model available to .NET (and Java) developers via Reliable Collections. Specifically, Service Fabric provides **ReliableDictionary**, **ReliableQueue**, and **ReliableConcurrentQueue** classes. These classes are used when state is partitioned (for scalability), replicated (for availability), and transacted within a partition (for ACID semantics). To ensure that the workflow has a consistent state at all times, the team used a combination of the reliable dictionaries,

queues, and transactions (https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-reliable-collections-transactions-locks) provided by Reliable Collections.

Other Reliable Collections are used only to manage a distributed cache that exposes the end data to clients. The collections are in memory, which helps boost performance—an easy win that spared the Geysir developers from having to manage the complexity of keeping a distributed cache synchronized across the cluster.

In Geysir's architecture, only stateless services perform actions, so the MarketApplicationType has three stateful services. Their exclusive role is to store key entities and their state:

- **Request state service.** A *request* describes how a call to a data source is performed—for example, an API endpoint and its parameters. Along with the form of the request, the service stores a list of the requests that must be performed, those that are currently running, and those that are matched with their most up-to-date results. This service is partitioned by request key.
- **Definition state service.** A *definition* describes how a quote is built and where to source the data. A definition can be linked to several requests, and definitions can have requests in common—that is, there's a many-to-many relationship between definitions and requests. A definition also specifies how often the requests need to be performed, whether the requested data is to be filtered, and, if so, how. This service is partitioned by definition key.
- **Instrument state service.** *Instruments* are the result of processing definitions—data has been requested and built into a list of quotes. The instrument state service is essentially a distributed cache that keeps a version of the market for quick access from the API. It is partitioned by instrument key.

Stateless services

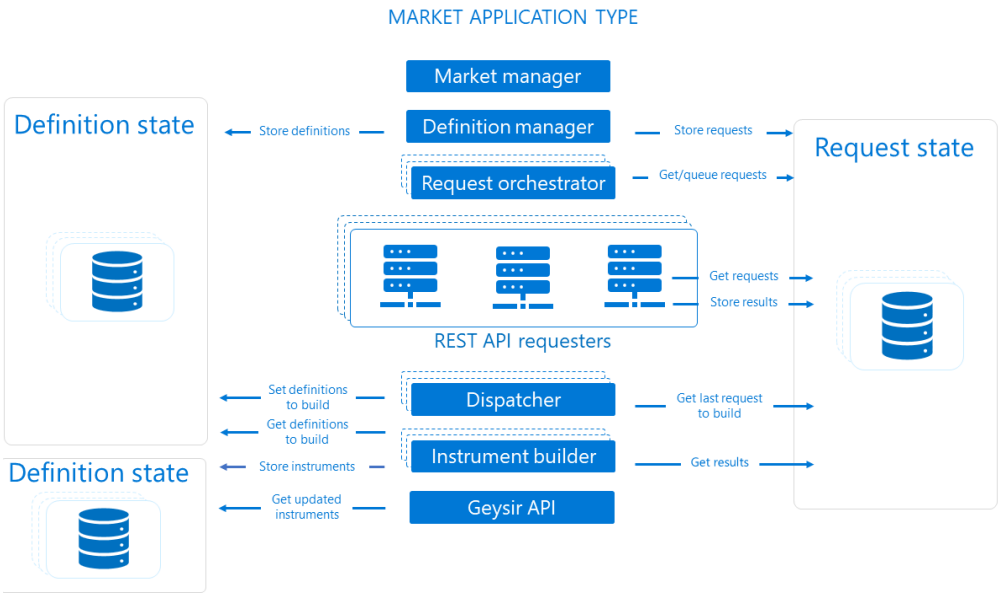
Stateless services are the engines of the application. They do not store any state information Each time a stateless service needs to interact with a state, it communicates with one of the stateful services.

As a platform, Service Fabric is completely agnostic with respect to communication between services. The Geysir team used the Reliable Services application framework to build the communication components and used the Service Remoting V2 mechanism for communication between services. At any time, a service can move across the cluster, be upgraded, or fail, so Service Fabric provides the Naming Service REST API. It keeps track of the current locations of services and associates the service names with the addresses of the currently running instances. When other services ask to communicate with a given service, Naming returns the current addresses. The Remoting stack then makes the connection appropriately. The Geysir developers

appreciate that all the complexity of calls between services and partitions is hidden behind the Service Fabric client.

Each of the following stateless services performs its own task:

- **Market manager** service dynamically instantiates all the other stateless services based on the specifics of the market, whether it retrieves market data in real time or as a snapshot.
- **Definition manager** service oversees loading the definitions from an external store or generating them from a set of rules. These definitions are then pushed to both the definition and request state services. This one-time job is performed at the creation of the application instance.
- **Request orchestrator** notifies the request state service of the requests that must be performed, considering their respective refresh rate (when polling data from API, for example). It has as many partitions as the request state service. Each partition of the request orchestrator is paired with and drives a corresponding partition of the request state service.
- **REST API requester** oversees the call to an API, managing any error that may arise and normalizing the response. This service is deployed for each API type to respect the fault isolation principle. Currently there are five different data sources, so there are five types of requester services. Each requester has the option to scale across several instances. Like the request orchestrator service, its partitions are paired with the request state service partitions.
- **Subscription feeder** service is dedicated to the data sources working by subscription (push-based), as opposed to the requester service, which interacts with pull-based data sources. The developers used the .NET Reactive Extensions to manage the events. The push-based sources don't need a refresh rate, so the subscription feeder server doesn't interact with the request orchestrator service. It normalizes the data on the fly and pushes the result directly to the request state service. Its partitions are paired with the request state service partitions.
- **Dispatcher** service picks up the notifications of completed requests from the request state service and then notifies the definition state that the corresponding definitions are ready for their instruments to be built or rebuilt. Since it interacts primarily with the request state service, their partitions are paired, but it also interacts with the definition state.
- **Instrument builder** service retrieves the corresponding results for each definition that needs to be built from the request state service, applies filtering, builds the instruments, and then publishes the instruments to the instrument state service. Its partitions are paired with the definition state service partitions. But it also reads data from the result state service and updates the instrument state service.
- **Geysir API** service is the only one exposed to Geysir's internal clients (through a GatewayApplicationType instance that acts as a proxy to unify the several instances of Geysir API). It exposes the data using two communication modes:
  - RESTful API, which is used by most Geysir clients.
  - GraphQL, which allows users to access specific subparts of the market, because a market is a big object. It also exposes additional fields, such as the last update time or the source of the market data.



Visualizing and improving performance

The Geysir team uses the open-source Service Fabric Explorer to monitor the cluster. It gives a great overview of the status of each application and service. To expose information from the data stored in its Reliable Collections, the team also implemented a custom monitoring service. This service exposes KPIs, such as the size of the queues in the stateful services (for example, the number of requests that need to be processed, are being processed, and have been processed). Having these KPIs allows the team to analyze the state of the data in depth and to improve the performance of its code based on both a service-level and data-level overview.

These metrics allowed the team to make several improvements to Geysir:

- Optimize cluster usage by destroying services that have nothing left to execute. For example, after a snapshot market has retrieved all its data, the definition manager, requester, dispatcher, and instrument builder services are not needed and can be automatically deleted.
- Identify deadlocks in transactions. The team hadn't considered the possibility of locks arising until it saw some slow performance metrics. The developers used the Reliable Collections lock mechanism to address the issue, and performance improved. Plus, moving to Service Remoting V2 allowed them to read data lock-free on secondary replicas, which also greatly improved performance.

- Call the secondary replicas of stateful services to read data that isn't necessarily up to date but has been recently synced (to compute the KPIs, for example).

Next steps

The Geysir team has several planned developments to further improve the application:

- Add autoscalability to optimize the platform's performance and cluster usage by adapting the number of instances of the stateless services based on the quantity of operations to process and how quickly they are usually handled. This would enable Geysir to automatically adapt to the peak and off-peak periods or to the volume of data processed per custom market.
- Enable market backup and recovery. The goal is to relieve the cluster by removing instances but to reload the market if needed for accountability purposes.
- Move from the on-premises cluster to an Azure cloud. The move would enable better coordination between Geysir and the specific Societe Generale simulation platform (https://customers.microsoft.com/en-us/story/societe-generale-complex-financial-simulation-platform-expands-on-azure-service-fabric-architecture) already using Geysir as a market source, which runs in Azure. It would also enable the team to take advantage of more Azure services.
- Use chaos testing to improve the platform's fault tolerance and robustness.

This project, developed in collaboration with Microsoft, is a milestone in Societe Generale's digital transformation agenda, in which moving to the public cloud is a priority.

> " For a team of three who must support a legacy application and develop its successor, the fact that Service Fabric manages most of the complexity involved in developing and managing distributed applications is a big bonus."
>
> —Jean-Martin Zarate:
> Societe Generale UK