FunRock takes mobile strategy games to the next level on Azure

April 20, 2018

Authored by Adam Tibbing, CTO and co-founder at FunRock, in conjunction with Mark Fussell from Microsoft.

This post is part of a series (http://aka.ms/TCP) focused on customers who've worked closely with Microsoft on Azure Service Fabric (https://azure.microsoft.com/en-us/services/service-fabric/) over the last year. We look at why they chose Service Fabric, and we take a closer look at the design of their application.

In this installment, we profile FunRock (http://funrock.com/) , their mobile strategy game back end running on Azure, and how they designed the architecture using Service Fabric.

Based in Stockholm, Sweden, FunRock is a fast-growing developer and publisher of mobile strategy games that are uniquely customized for markets primarily in the Middle East and North Africa. The story lines, environments, sounds, music, graphics, and support are all adapted to the target region's culture and language, where FunRock has local knowledge and presence. The team consists of experienced game developers who have created successful game titles in the past.

The company shares a mutual mission: To take the massively multiplayer online (MMO) strategy game genre to the next level.

FunRock has been relying on the flexibility and scalability of cloud hosting for years. About two years ago, in anticipation of future growth and the loads they expected, they moved their back end from Amazon Web

Services to Azure and Service Fabric.

Overview of Heroes United

In MMO strategy gaming, the greatest technical challenge is concurrency. Handling up to hundreds of thousands of concurrent players requires a specialized architectural approach. FunRock's latest title, *Etihad Al Abtal* ("Heroes United"), takes place in a desert wasteland where the Arabic nations unite against a common enemy.

Players take the role of commander of a small plot of land, where they gather resources and construct a powerful base with help from the different nations and the guidance of their generals. Players can explore the wasteland for resources and join other players in mighty alliances.

Unlike most other mobile strategy games, Heroes United takes place in a truly open world. Players can pan around and explore the world in a strategy view and in a fully-zoomed view. Since the world is divided into smaller areas, and each area manages itself, the world can grow almost infinitely without affecting performance.

A traditional back-end architecture usually handles this kind of load using shared databases, load balancers, and caches. Planning and managing an architecture like this takes time and resources away from product and feature development. FunRock was motivated to try a different approach to MMO and turned to Service Fabric Reliable Actors (https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-actors-introduction) , an implementation of the Actor design pattern.

The actor model is a good match for a project of this type. Each entity in the game--a connection, player, world area, player base, and so on--is isolated. It doesn't know anything other than itself and its own data, and how to interact with other entities. The actor model is specifically designed for concurrent or distributed systems in which a large number of actors can execute simultaneously and independently of each other. Actors can communicate with each other, and they can create more actors.

Moving Heroes United to the Service Fabric platform was an easy decision. Service Fabric handles the heavy lifting, so the game developers can focus on design. Not only did Service Fabric provide an easy-to-use microservice architecture, it gave FunRock other key features including automatic scaling, communication, high reliability, and load balancing between nodes.

Migrating to Service Fabric

Porting the traditional architecture of a production environment to Service Fabric proved to be a two-stage project.

The first step was to move the current back-end gaming architecture, which used SQL Server databases for data persistence, into Service Fabric. The move was much simpler than the developers expected, since Service Fabric can host any executable. FunRock simply had a communication service and a monolithic application service, which could be balanced over several nodes. Data persistence is handled by a single Azure SQL Database (https://azure.microsoft.com/en-us/services/sql-database/) instance, a platform-as-a-service (PaaS) offering that helped ease the transition to Azure.

The next step was to slowly take the existing application service apart into multiple microservices and actor services, and migrate the data in SQL Server to the actor model. This migration happened very gradually over several months.

FunRock realized they needed a few tools to manage and visualize the data. In a standard relational database, data can be easily viewed, modified, and backed up using existing software. For the new platform, FunRock designed a tool for viewing and modifying the state of any given actor instance--simply by providing the ID and type of actor.

The unique nature of the MMO environment introduced a few challenges. Service Fabric supports actor service backups (https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-backup-restore) , but FunRock had to work out issues with consistency, point-in-time restore, and restoring to an environment with a different number of partitions.

Since game development usually involves multiple development environments, FunRock also built an extensive library of tools to automate setting up, upgrading, managing, and removing environments. Each environment is a resource group in Azure that contains all resources needed to host a game and is completely isolated from the others.

Service Fabric architecture

The heart of the MMO architecture for Heroes United is a microservices-based internal layer, which is not open for communication from outside. Its services manage the game worlds, assets, and player services.

Services with exposed endpoints are placed behind a DMZ in the communication layer. The back end hosts common services used for backup and logging. The logging service receives and handles all logs except those coming from Service Fabric itself. The backup service performs backups of all stateful actor services on a given interval and manages restores from previous backups.

> " In the gaming industry, a lot can happen overnight, and we really appreciated the flexibility Service Fabric gave us. For example, we can easily scale on a per-service basis or scale our Service Fabric cluster up or out."
>
> —Adam Tibbing: Chief Technology Officer
> FunRock

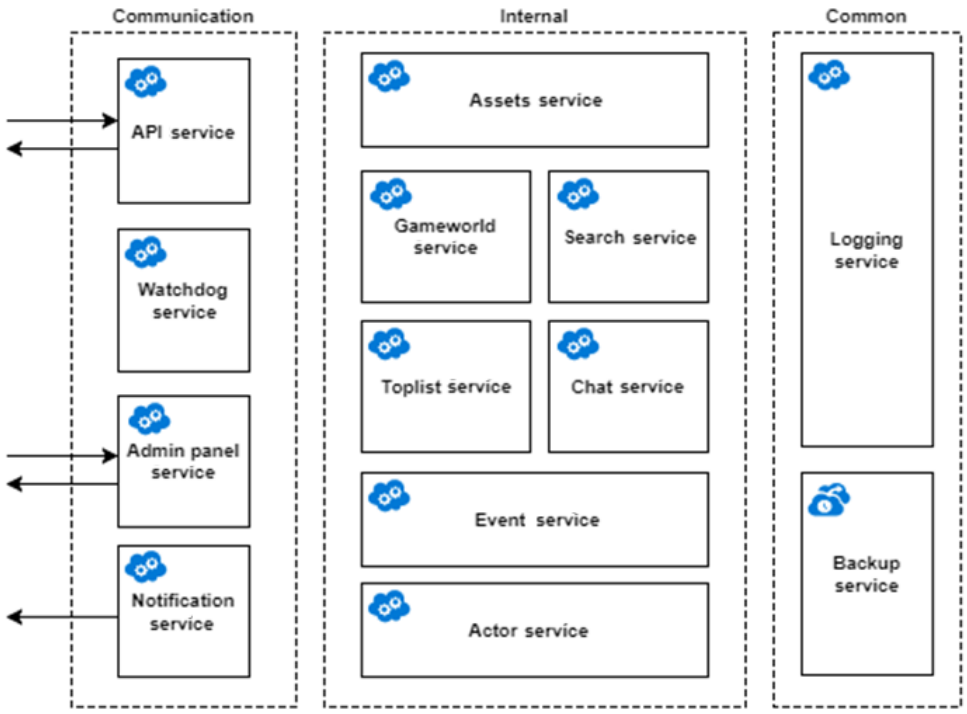

Figure 1: Overview of the different services hosted within Service Fabric.

Communication

All services that need to be exposed are placed in the DMZ layer by specifying that they can only reside on a designated node type. The API used for client communication is served by an API service. Each API service instance is open for communication in two ways: HTTP and SignalR sockets.

HTTP is mainly used as a way for the client to verify that it is compatible with the targeted back end, and to fetch game assets before opening the SignalR socket. The game assets consist of all values, texts, and other content used in the game. The assets can be large, so they are sent over HTTP using HTTP compression.

As soon as a client has completed its handshake and verified that it has the latest game assets, it opens a SignalR websocket secured by Secure Sockets Layer (SSL). At this point the communication is two-way: the server can send information about events in the game that the client might be interested in.

Each client is piped to its own gateway actor instance within Service Fabric. These actors are hosted within the communication layer, which is evenly distributed across all API nodes and doesn't contain any persisted state.

The communication layer and its API also provide session handling and the rules that define allowed and expected requests. The communication layer then in turn communicates with the internal application services as necessary.

Internal communication

Service Fabric supports multiple internal communication methods. The most common way is by using service remoting, but HTTP or other protocols can also be used.

Since services might move between different nodes, the addresses and ports constantly change. Service Fabric provides a Naming service that handles discovery and resolution of all services.

Some services are stateless, meaning that they can't and shouldn't keep any persisted state. When communicating with a stateless singleton service, the game developers won't know which instance they get, unlike coding in traditional models. Instead, it's up to the load balancer to decide.

Even though state is not kept in these services, at times all instances of a stateless service need to be notified of some new information to hold in memory. For example, when the game developers want to temporarily put the communication layer into a maintenance mode, normal requests are not allowed.

To handle these cases, the developers added an Azure Service Bus (https://azure.microsoft.com/en-us/services/service-bus/) communication listener to these services. Using Service Bus topics and subscriptions ensures that all subscribing instances receive the message and act if needed. Service Bus is ideal for asynchronous communication, while HTTP and other mechanisms are better for synchronous and RPC-style communication.

Stateless services doing most of the heavy lifting

Managing the game and its different actors is performed by a couple of stateless services. In the communication layer, a notification service sends push notifications via Notification Hubs, a game administration panel built in ASP.NET Core, and of course the API service for client communication. In addition, a watchdog service simulates a client and periodically reports information about cluster health, response times, and other vital diagnostics.

The internal layer hosts services that are not open for communication from outside. The Gameworld service manages all available game worlds used by the players. This service keeps track of many variables, such as how large a world is and when to expand. It also handles the placement of new and relocated bases.

The Assets service provides game assets for all other services, including all actor services. The Assets service tracks the buildings, troops, and achievements that are available as well as all text used in the client. An Event service handles all game events sent to the clients--that is, the happenings in the game that the client cannot predict such as incoming messages or attacks.

The internal layer is also home to services handling top lists, search, and chat in the game. These are designed using the Eventual Consistency pattern to achieve high performance and availability, and the data is kept in memory only to be persisted periodically.

Using the actor model

The architecture relies heavily on actors. Most entities in the game are modeled as actors, each handling nothing but its own state. This model is in most cases a huge advantage since actors are lightweight, reliable, independent of each other, and scale extremely well. They are also single-threaded, which eliminates the risk of race conditions.

For example, an actor is used for each player base in a game world and contains information about the owner, buildings, and troops it contains. If the owner sends a movement of troops to a different base, a new movement actor is created, containing only the information about the troops, the starting location, and the intended destination. An actor reminder is set to the intended time of arrival, and as soon as it fires, it calls the target base and inserts those troops before it destroys itself.

However, the FunRock developers discovered some anomalies. Actors frequently communicate among one other, and occasionally a deadlock can appear. For example, if actor A is waiting to communicate with actor B, and actor B is waiting for actor C, or a chain of calls is waiting to communicate with actor A, a deadlock

might occur. The FunRock developers learned to be careful to avoid these situations while designing the game logic.

Logging and analytics

All services in the Service Fabric cluster forward their application logs, errors, statistics, and analytics as Event Tracing for Windows (ETW) events. Service Fabric comes with support for ETW logging, but the FunRock team needed a way to aggregate and search these logs.

After researching possible solutions, they found that the best-suited tool was the ELK stack (ElasticSearch, Logstash, and Kibana) combined with X-Pack. This setup is currently handling and indexing almost four million logs per day in the production environment without any hiccups.

ElasticSearch is also hosted in a cluster of virtual machines running Linux. This cluster allows it to scale as needed depending on the number of requests (insert and read). Currently the logging platform uses two client nodes to handle all incoming requests, three master nodes, and three data nodes.

Since Service Fabric emits ETW events, the FunRock developers were able to easily implement additional diagnostics using EventFlow (https://github.com/Azure/diagnostics-eventflow) , an open-source tool from Microsoft. EventFlow listens to the ETW stream, then bulk-posts the events to ElasticSearch.

Built-in Service Fabric telemetry is forwarded to Application Insights.

Open-source tools Kibana and Grafana are used to visualize and quickly query the logs in ElasticSearch. Kibana provides a data visualization plugin for ElasticSearch, and Grafana aggregates data from ElasticSearch or other data sources, and displays it as beautiful graphs and diagrams in real time.

Security

The game doesn't handle any sensitive data, but the FunRock developers still take security very seriously. Public-facing servers are hosted within the DMZ. Service Fabric cluster communication is secured using certificates stored in Azure Key Vault (https://azure.microsoft.com/en-us/services/key-vault/) . External communications are encrypted using SSL, and access control for all administrative user management is handled by Azure Active Directory (https://azure.microsoft.com/en-us/services/active-directory/) .

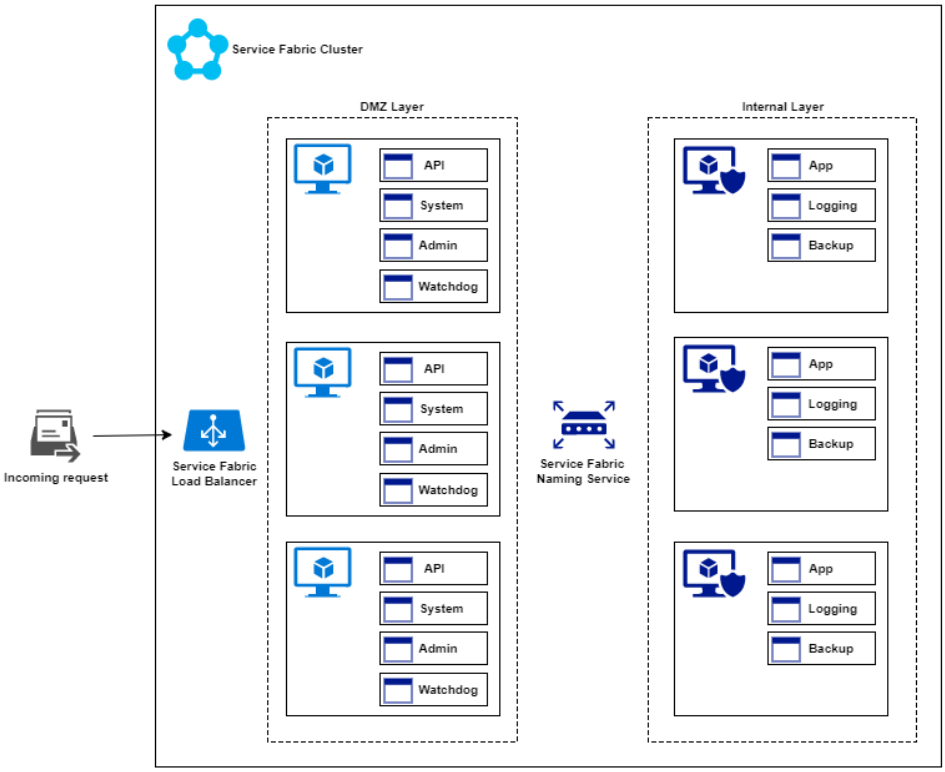

Figure 2: Different node types are used for layered security. The type and number of node instances vary between environments.

Disaster recovery

Service Fabric comes with built-in disaster recovery functionality that was used to back up and restore service data. The backups are stored in Azure Blob Storage as binary dumps. FunRock created a backup service to take backups regularly. In case of a total data loss or a specific, on-demand restore, service states are restored from there.

However, to improve consistency and reduce load, the team is currently working on an improved backup solution based on Azure Cosmos DB (https://docs.microsoft.com/en-us/azure/cosmos-db/) , which is specifically designed for global distribution across multiple Azure regions.

Administration

Management of the Heroes United MMO platform involves both game and cluster administration. To do this, FunRock built a separate application in ASP.NET Core and hosted it within Service Fabric. Administrators can now communicate directly with all other services without the need for a separate administration API.

The foundation here is a generic way to edit the state of any given actor, based on interface name and actor ID. FunRock can easily apply attributes directly in the state data contracts to define read/write rules for different administrator user levels.

Advantages of Service Fabric

To build the game platform back end, the FunRock team used the following Service Fabric capabilities:

- **Scaling:** In the gaming industry, changes happen quickly. With Service Fabric, FunRock can easily scale processing power on a per-service level as needed. The load is automatically balanced across the nodes in the Service Fabric cluster. If necessary, they can scale the architecture both horizontally and vertically by adding nodes or changing the node size.
- **Reliability and service discovery:** Service Fabric guarantees that services are always available and reliable. The Service Fabric Naming service directs any request to an available instance. If a node goes down or is unhealthy, a new instance is started on a healthy node. Persistent data is automatically replicated to the new node.
- **Flexibility:** FunRock needed the flexibility to respond to changes in the fast-paced gaming world and add requirements quickly. Although they considered other PaaS services such as those provided by Azure App Service (https://azure.microsoft.com/en-us/services/app-service/) for their MMO platform, Service Fabric offered the level of flexibility and control they needed. It can be hosted on-premises if desired and can contain many different types of services at the same time, including queues, worker processes, guest executables, and websites.
- **Co-locating code and compute:** Instead of having a separate data store, the game state is stored within the services and kept in memory while active. This means faster access times, lower costs, and great scalability.
- **Layered security:** A cluster can contain multiple types of nodes, so FunRock can physically separate different kinds of workloads. For example, one node type handles all public communication for the API layer. The internal services all reside on other node types that allow only encrypted communication and are not exposed to the Internet. Using placement constraints, the FunRock team can decide at a service level which node types are allowed as hosts.
- **Actors and actor reminders:** With the actor model, the game developers need never worry about race conditions. Actors are single-threaded and will only process one thing at a time. The developers use actor

reminders to queue an action for a given point in the future or at a regular interval. By using actor reminders, FunRock doesn't need any other type of scheduling or polling, which is common in traditional back ends.

- **Pub-Sub pattern:** Service Fabric offers a pub-sub model for sending events from actors. FunRock uses this feature to publish in-game events to connected clients that might be interested in them. An event could be an incoming message, attack, or anything else that the client is not able to predict.

Integrating Azure Services

Each game environment consists of many Azure services organized in resource groups. Setting up or taking down an environment is easily done using Azure Resource Manager (https://azure.microsoft.com/en-us/features/resource-manager/) templates and PowerShell scripts. All environments are fully cloud-hosted but could theoretically be hosted on-premises if ever needed.

Service Fabric comes with several storage accounts, scale sets, load balancers, and virtual networks. An Application Insights account is created with each environment, where all fabric events are collected. To store backups, each environment is set up with an Azure Blob Storage (https://azure.microsoft.com/en-us/services/storage/blobs/) account. These can be accessed by other environments in case a backup needs to be restored from a different blob.

Additionally, FunRock uses Azure Service Bus to communicate with stateless singleton services, since it is not normally possible to specify the instance to call. By definition, they do not store any data, but information can still be kept in memory and modified to suit certain gaming situations.

Some Azure services are common for all environments, such as Notification Hubs, Key Vault, Azure Active Directory, and Azure Functions (https://azure.microsoft.com/en-us/services/functions/) . These services are kept together in a special resource group and are not deployed with a new environment.

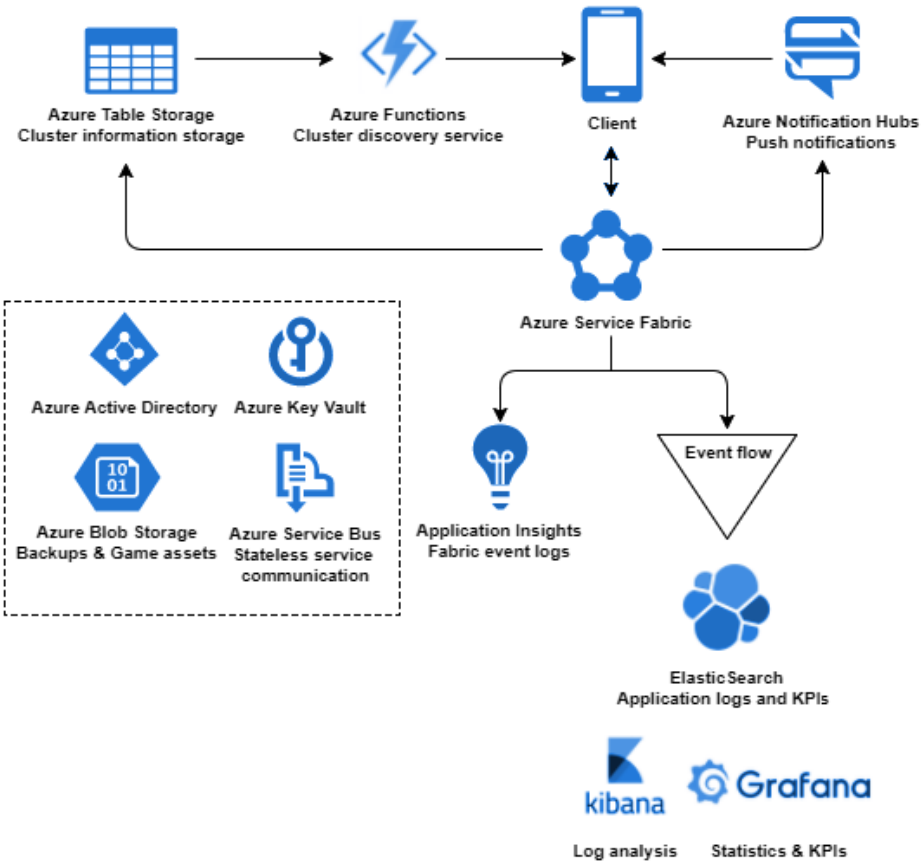

Figure 3: Many Azure Services are used in the FunRock MMO platform for Heroes United.

Summary of the services used:

- **Azure Service Bus:** Used for communication with all instances in stateless singleton services.
- **Azure Blob Storage:** Stores backups for disaster recovery and game assets.
- **Azure Functions:** Supports the client discovery service used by the client to determine which server to connect to.
- **Azure Table Storage:** (https://azure.microsoft.com/en-us/services/storage/tables/) Stores a list of all deployed clusters (in Azure and local) and their properties and health, and is used by the cluster discovery service.
- **Azure Key Vault:** Manages certificates and secret keys.
- **Azure Active Directory:** Manages administrative user accounts for developers and support team.
- **Azure DevOps (formerly Visual Studio Team Services):** Used for load testing applications and simulated clients.

- **Azure Notification Hubs:** (https://azure.microsoft.com/en-us/services/notification-hubs/) Sends multi-platform push notifications to the players.
- **Application Insights:** Stores and aggregates Service Fabric telemetry.

Summary

When FunRock moved to Service Fabric, they made the most of its actor and service programming models and management capabilities to create a highly scalable back end for the MMO platform. The solution scales to accommodate a very high number of concurrent users when needed, while providing an affordable service during off-peak hours.

The microservice architecture that Service Fabric enabled is also flexible enough to allow major future design changes if necessary. Migrating an existing architecture to Service Fabric initially seemed challenging, but the FunRock team found the process incredibly simple. As a result, FunRock is well positioned for future growth and changes.

> " Our gaming architecture relies heavily on Service Fabric actors, which was a huge advantage since they are reliable, thread-safe, and scale extremely well. "
>
> —Adam Tibbing: Chief Technology Officer
> FunRock