# Proactima Solutions manages risk on Azure Service Fabric

April 26, 2018

*Authored by Jarle Nygard and Anders Osthus from Proactima Solutions, in conjunction with Ed Price, Paolo Salvatori, and Vaclav Turecek from Microsoft.*

This is part of a series on customers who've participated in the Azure Service Fabric (https://azure.microsoft.com/en-us/services/service-fabric/) preview program over the last year. We look at why they chose Azure Service Fabric and take a closer look at the design of their application, particularly from a microservices perspective. You can read all the profiles in this series here (https://blogs.msdn.microsoft.com/azureservicefabric/tag/customer/) .

In this installment, we profile Proactima, their UXRisk web application running on Azure, and how they designed the architecture using Service Fabric.

Back in 2008, Proactima Solutions set out to create a tool to capture their unique understanding of risk analytics and risk management. The first pilot from this effort was called Webrisk, despite having nothing to do with web. It was based on WPF and SQL Server and featured an on-premises installation and Click-Once

deployment. Fast forward to 2011 and with the hire of a business developer and a solution architect, the formation of a cloud solution started.

The first version of UXRisk—a complete rewrite of Webrisk—launched in spring 2015. The frontend was written as a Windows Store app, and the backend was built on Azure Cloud Services (https://azure.microsoft.com/services/cloud-services/) . However, this solution would not allow Proactima to UXRisk in the direction required. Stateless services with caching layers proved hard to manage, slow to update, and brittle under load.

One of the core principles of line of business (LOB) apps is the ability to customize the input forms and the business rules. In the classic way to deliver LOBs, there are a number of options. Some prefer to alter code and recompile a special version for each client, while most would store the configuration in a database. However, it has traditionally been very hard to replicate configuration between customers, since over time the configuration databases change so much. As a result, it's nearly impossible to just over configuration. For UXRisk, one of the core design goals was to be able to move this type of configuration between clients easily.

As a small player in a large market, UXRisk needed to stand out in some way. The plan was always for the user experience to be above and beyond what others provided (hence the UX part of the name). However, as usage increased, it became evident that the backend could not keep up in terms of latency. It could take as long as 30 seconds to load large data objects, and the CEO of Proactima told the team that this was not acceptable.

And so, over the summer of 2015, the team began the journey to microservices using Azure Service Fabric.

# High level goals for a microservices-oriented architecture

As the team started the journey, goals were defined that would guide the development:

- Use a service resolver for all communication between services, so that the details are hidden from consumers.
- All service calls should use a request-response pattern, not *n*-number of arguments.

- Services should not throw exceptions. Generally, they should be caught and dealt with on the service side, and a response with status 500 (or similar) would be returned, much like how the web works.
- Keep the number of external dependencies as low as .
- Create a multitenant solution, where each tenant has a dedicated set of services.

> " Pushing out new revisions of code in Service Fabric is a core concept. This, combined with all the different upgrade policies, makes code rollout quick, easy and something we have confidence in."
>
> —Anders Østhus:
> Proactima Solutions

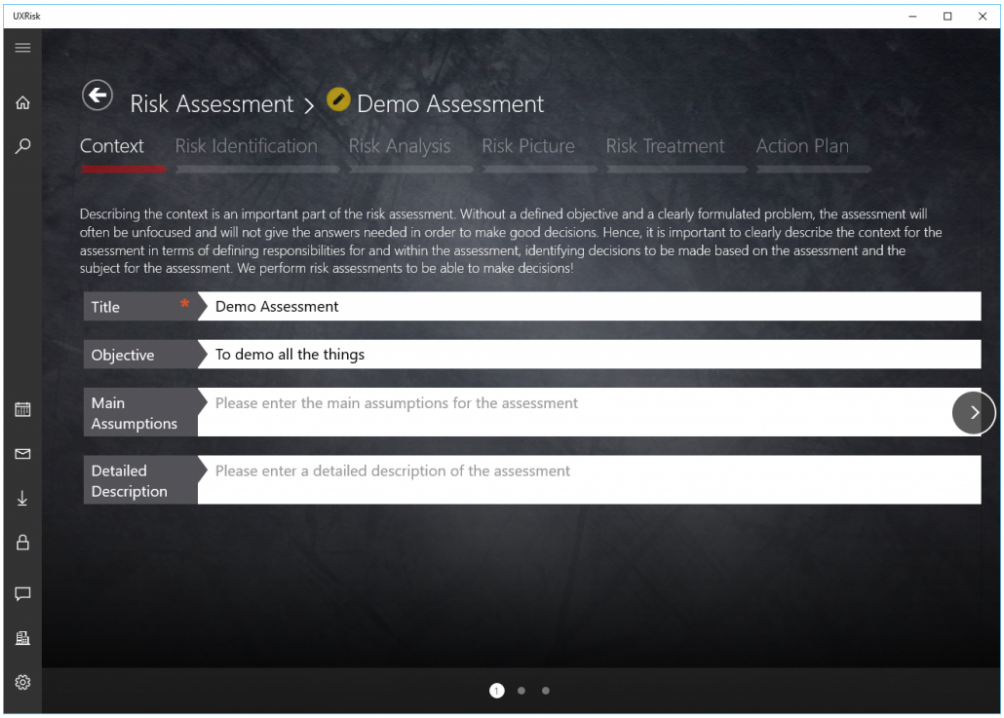


Figure 1: Screenshot from UXRisk

# Azure services

UXRisk makes use of several Azure services to enable rapid development and a secure and scalable solution.

- **Service Fabric** hosts the API endpoint, with tenant-specific services and shared services. Data is stored, projected, and retrieved.
- Service Bus (https://azure.microsoft.com/en-us/services/service-bus/) is used for asynchronous service-to-service communications and to enable a truly scalable architecture. Tenant-specific topics are created and posted to by the data actors and processed by tenant-specific business rules.
- SQL Database (https://azure.microsoft.com/en-us/services/sql-database/) stores all the user and tenant information.
- **Storage** takes care of persisting all user input in a performant and cost-effective manner. By utilizing Azure Storage Tables (https://azure.microsoft.com/en-us/services/storage/tables/) , UXRiskcan
- **Linux VMs** are used for hosting an Elasticsearch UXRisk supports highly dynamic schemas that serve a multitude of roles. Proactima's projected data is stored along with the raw data, to enable fast queries over large datasets and full text index searches over de-normalized data.
- Key Vault (https://azure.microsoft.com/en-us/services/key-vault/) makes sure that their configuration is safe; this includes connection strings and regular application settings.
- App Service (https://azure.microsoft.com/en-us/services/app-service/) for supporting web sites, such as the UXRisk signup site and home page (https://uxrisk.com (https://uxrisk.com/) ).
- Azure Active Directory (https://azure.microsoft.com/en-us/services/active-directory/) (Azure AD) enables Proactima to provide their tenants with SSO, and it ensures that users' credentials are stored in a secure and controlled manner.

> " The local onebox Service Fabric experience is great. It's not an emulation, it's the actual bits running locally, and it allows us to debug 99% of our issues locally."
>
> —Anders Østhus:
> Proactima Solutions

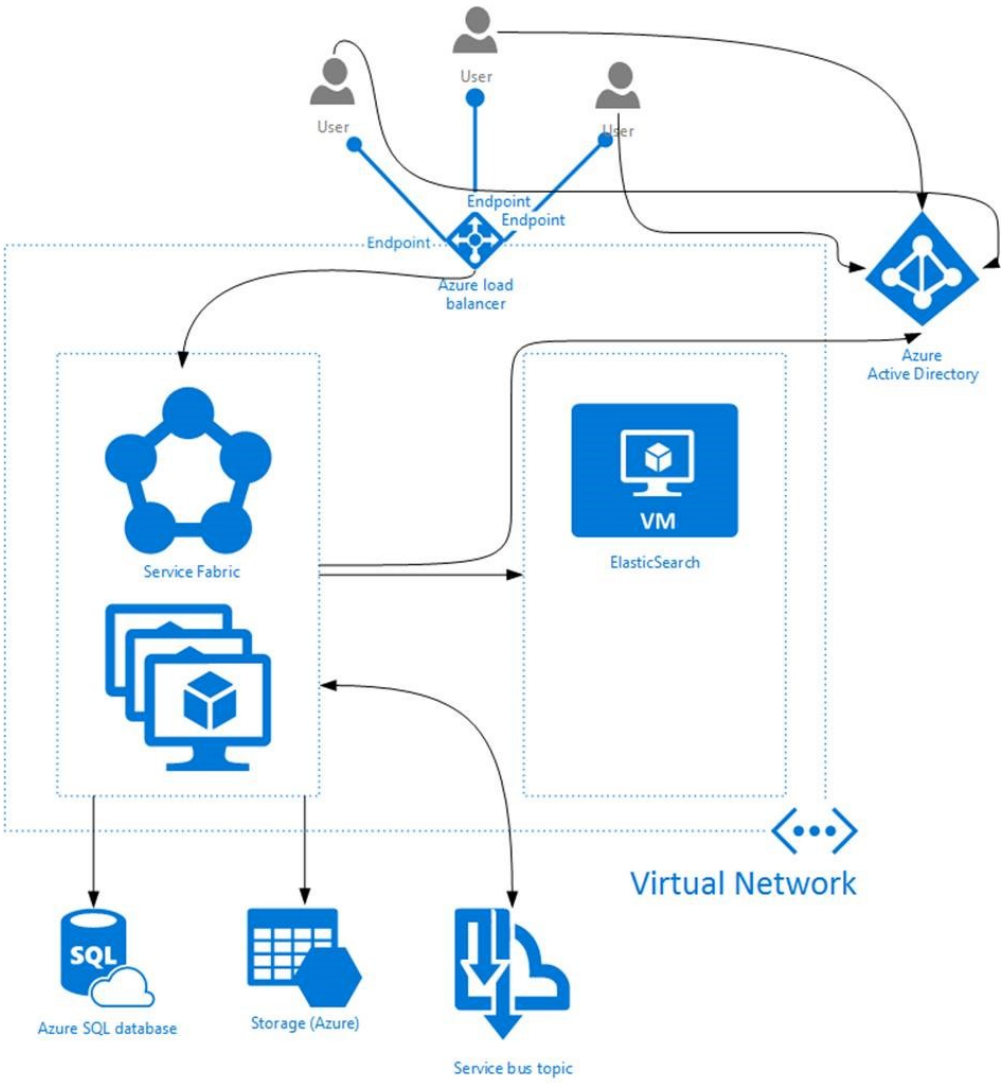


Figure 2: The Azure services leveraged by UXRisk

# Slice and dice

From the start of the development of UXRisk, the team had a vision of microservices in mind. So for the transition to Azure Service Fabric, the challenge was in how to slice and where to dice the code to achieve this goal.

Figure 3 shows a quick summary of all the services and their logical placement in the Service Fabric applications. Proactima created a separate instance of the tenant application with services for each customer.

With the aid of Microsoft, the development team used Service Fabric's ability to create services through C# code. With this in mind, the development team started slicing all the existing code into either a set of tenant-specific services could be created as tenants were signed up. This removed the partitioning challenge for shared services altogether and ensures that Proactima can scale their backend to almost an infinite number of tenants.

Any change to data by a user will trigger a number of operations within UXRisk:

- The incoming request is validated and potentially transformed, via tenant-specific rules:
  - Check for schema validation.
  - Add and remove properties on the data.
- Once the request is validated, it gets committed to :
  - This is where optimistic locking is enforced, using Table storage mechanisms.
  - If the store operation fails, the status code from storage is returned to the user.
- The successful operation is then broadcasted to a post-processing Service Bus topic specific to each tenant. The topic has 4 distinct subscriptions (one each consumer service):
  - SearchWriter service will store the raw object in
  - ObjectViewProcessors service will project the data and store it in
  - PostProcessRule service will execute any valid custom business rule for the data.
  - Signaling service will inform any other user subscribing to notifications on the current object of the change, enabling them to load the changed object for an improved user experience.

Figure 4 shows you the high-level overview of the pipeline that is executed when a customer signs up. Proactima pre-resolves all the steps that should be done in the Provisioning Manager and then pass on the steps to the Provisioning Processor. This is done so that they can track progress and easily resume in case the process gets interrupted (for example, by a service upgrade).

> " For a multitenant solution, scale is vital and with Service Fabric we can scale out as more tenants sign up. In fact, by creating services for each tenant we did not have to spend a lot of time thinking about the partitioning and sharding requirements let Service Fabric handle it for us."

—Jarle Nygård: Solution Architect
Proactima Solutions

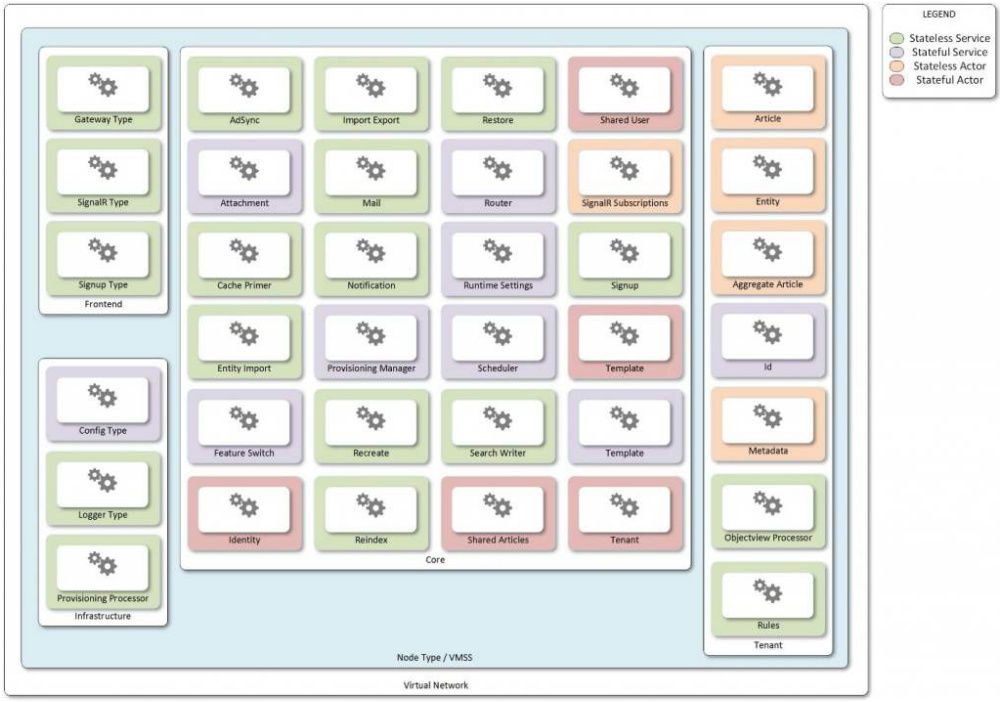


Figure 3: Overview of the slice-and-dice system and tenant services in the UXRisk backend

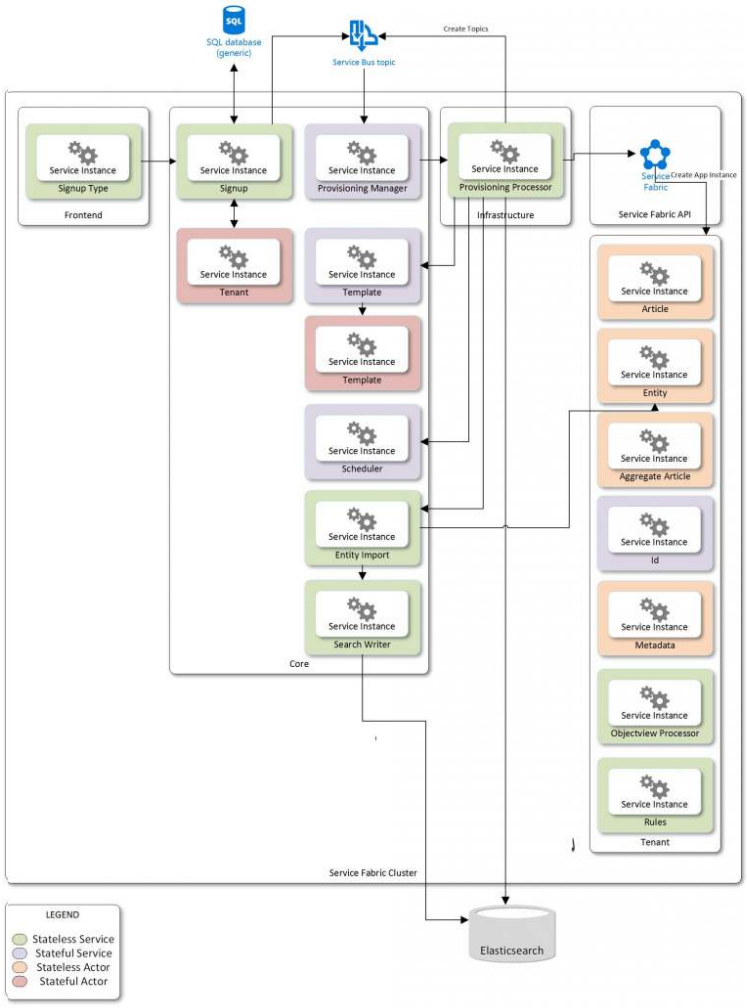

Figure 4: High-level overview of the internal process when provisioning a new customer in UXRisk

# UXRisk templates

A key component of UXRisk are templates. As an example, Proactima has templates for doing risk assessment, creating minutes of meetings, and handling audits. The template defines the UI (field ordering, and so on), schema (fields, required/min/max values, and so on), notifications, projected values, and much more. The templates are 100% self-contained, and can be moved between environments (development, staging, and production) and tenants. And since UXRisk is multi-language, the template even contains all the strings a template requires, for all the languages that the template supports.

UXRisk stores all the data as single objects from a graph. So for an audit, that would look something like this:

- Audit
  - Findings
    - Measures
  - Workshops
    - Invitations

And then each of these objects can link to *n*-number of entities (such as a responsible user, business process, and more).

Typically, a graph of objects consists of 20 or more objects, with some as large as 300 objects. Each object is an for the entire graph. Traversing the graph potentially takes a very long time, say if the actors have not been . To solve this performance challenge, Proactima created a service that will load the top *x* number of root objects for each tenant. The triggering of these calls is handled by their service, and the schedule is created upon provisioning. So every 30 minutes the top *x* root objects are loaded into memory for all the tenants. This is a nice tradeoff between performance and memory usage.

All the configuration is stored in Azure Key Vault, but it's exposed to Proactima's services via a service. It takes care of caching the values (to avoid going to Key Vault on every request) and returns the data in the correct format (string, datetime, and so on). This service is stateless and runs on all nodes, so configuration data is really fast to load.

Identity is handled via an identity actor where there is one actor for each user in UXRisk. Upon first , the actor is created by loading information from Proactima's SQL Database and then persisted in a stateful actor as an identity model. Every single API call then uses the JSON Web Token (JWT) from Azure

AD to determine if the user can sign in, and then it loads the identity model from the actor. The identity model is then passed along the call chain, enabling them to always know which user started the call chain.

UXRisk has more than 20 shared system services, such as , and 4 actor types. Each tenant gets 3 services and 4 actor types.

# Major benefits of Azure Service Fabric

- Tenant-specific services enable scalability and isolation between tenants.
- Flexible programming model enables a multitude of scenarios, and allows Proactima to granularly add extra functionality.
- It's safe, easy, and quick to deploy.
- Performance and uptime has been extraordinarily valuable for UXRisk.

# Summary

Proactima wanted a backend that could keep up with latency and load large data objects. They wanted a service resolver (with a request-response pattern) to communicate between services behind the scenes. They wanted to ensure that the services don't throw exceptions and to keep the external dependencies as low as possible. To achieve this, Proactima Solutions leveraged Azure Service Fabric to build the structure for UXRisk, so that it is flexible, scalable, integrated, high-performing, easy to deploy, and easy to update. Service Fabric enabled Proactima to create a platform that could scale to millions of users.

> " Knowing that Service Fabric is a distributed platform, I don't have to worry about losing a node. I know that Service Fabric will rebalance itself, and continue running, so that I can fix it when I'm back at work. "
>
> —Anders Østhus:
> Proactima Solutions