Cyber Security - Information Security & Testing

Strategic delivery:	☐ Setting standards	☐ Increasing and informing choice	☒ Demonstrating efficiency economy and value
Details:			
Meeting	AGC		
Agenda item	11		
Paper number	[AGC (07/12/2016) 519 DM]		
Meeting date	07 December 2016		
Author	David Moysen, Head of IT		
Output:			
For information or decision?	For information		
Recommendation	The Committee is asked to note this report.		
Resource implications	As outlined		
Implementation date	Ongoing		
Communication(s)	Ongoing		
Organisational risk	☐ Low	☐ Medium	☒ High
Annexes			
Annex A - Application Security assessment Annex B – IfQ security model			

1. Introduction and summary

1.1. The purpose of this report is to provide the Committee with our approach to cyber security, further to its request following an oral presentation at its last meeting.

2. Background

2.1. The National Cyber Security Strategy 2016-2021 was published in November 2016 and noted the UK is critically dependent on the Internet. 'However, it is inherently insecure and there will always be attempts to exploit weaknesses to launch cyber attacks. This threat cannot be eliminated completely, but the risk can be greatly reduced to a level that allows society to continue to prosper, and benefit from the huge opportunities that digital technology brings.

2.2. Our systems have grown and developed over the years alongside the growth of cyber threat. We have put in place a range of mechanisms, and ways of providing assurance, that those mechanisms are effective, to guard against threat.

2.3. At the last meeting of the Audit and Governance Committee an oral presentation set out the steps the HFEA is taking, as the IFQ programme moves from development to implementation, to ensure cyber security. This paper builds on that presentation and provides documentation supporting our assessment that, in developing our new systems within the Information for Quality Programme, our arrangements are as secure as possible.

3. Standards

3.1. There is a plethora of standards, assurance frameworks and expectations in place. The '10 Steps to Cyber Security' are widely known and are recognised as an effective means of raising awareness of cyber threats within the leadership of organisations, and to enable a greater capability to safeguard their most important information assets, such as personal data, online services and intellectual property. The 10 Steps to Cyber Security features controls to reduce risks in the following areas:

- Information Risk Management Regime;
- Secure Configuration;
- Network Security;
- Managing User Privileges;
- User Education and Awareness;
- Incident Management;
- Malware Prevention;
- Monitoring;
- Removable Media Controls;
- Home and Mobile Working.

3.2. The HFEA has a successful track record in ensuring its systems, over time, meet these important expectations. We have policies in place relating to information governance and security. Periodically, we have sought assurance by a range of means including review by internal audit and penetration testing (carried out by independent third party experts) and by the application of regular vulnerability assessments.

3.3. Equally the HFEA has devoted two years to a fundamental redesign of its information architecture. Principles relating to security considerations have been built in from inception.

3.4. The IFQ team has adopted the principle of "Secure by Design". This is an approach, developed in conjunction with our retained security consultant working alongside us since the inception of the Programme, which has as its paradigm that software and systems are designed and implemented with security in mind from the ground up.

3.5. The HFEA is developing an Assurance Plan leading to a full Risk Management and Accreditation Document Set for approval prior to the EDI replacement going live. This RMADS is being developed by an independent security consultant. In effect, this document provides details about the system being developed and a full risk assessment. The document will then go on to provide details of how risks are to be mitigated by the application of a Baseline Control Set and will need to be signed off by the Siro prior to the application going live. Current CESG guidance suggests that the RMADS approach is often disproportionate in terms of the effort that is required and that the business should decide what level of risk management is suitable to its needs. However, given the sensitive nature of the data the HFEA feels that the creation of an RMADS is proportionate.

4. Security progress to date

4.1. The high level aims of the security objectives are set out here, and ensure the:

i. Confidentiality, integrity and availability of the sensitive data held in the solution

ii. Confidentiality, integrity and availability of all data and systems in all environments hosting the systems. (This includes stages of development, testing, pre-production and production).

iii. Solution adheres to relevant legislation and regulatory standards

iv. Solution (and any infrastructure changes required for it) do not have any effect on the operations of the core corporate systems.

v. Reputation of the organisation is not damaged by any activities surrounding the implementation and operation of the new systems.

4.2. A set of technical security model documents have been produced as part of the Programme – principally for use by the various internal and external development teams to ensure integrity with the model and to provide background briefing information to independent assessors, contracted to provide external assurance.

4.3. Appendix A contains the IfQ security model and high level security and architecture solutions for the HFEA Clinic Portal and the Release 2 data submission (EDI) replacement systems. Whilst these are dense and technical in nature given the audience they are intended for they are a demonstration that the architecture is being developed with security at its core, and are annexed as information.

4.4. Nevertheless, what is more important is that there is a programme of independent assessment for vulnerabilities in place, providing assurance to the SIRO, Authority and the Audit and Governance Committee.

4.5. Members are aware there are three main components of the Programme – the HFEA website; the HFEA Clinic Portal; and 'Release 2' of the Clinic Portal – the ability for clinics to submit treatment information to the HFEA. (A separate agenda item updates the Committee on progress with the IfQ Programme more generally). In terms of security, the website is lowest risk; increasing with the Portal (as there is more two-way interaction) and reaches its peak with Release 2. Currently, the system threat is limited as there is a direct link between clinics and the HFEA. The new system is browser based and therefore the 'attack surface' is greatly increased.

4.6. Our testing programme is established in two phases – firstly at Beta (broadly) and then prior to live release. The HFEA website and Clinic Portal have been independently assessed for vulnerabilities at the Beta stage, with the recommendations made in the report addressed. It reported:

4.7. "In general, the security of the application components reviewed was high, as the applications are employing some of the latest technologies from Microsoft they are following good security practices in the main when it comes to the application code with no apparent weaknesses that are covered by the OWASP Top 10, such as Cross Site-Scripting and Injection attacks being handled by the .NET platform security features. The application is employing security features of the platform to provide protection as a

4.8. result when testing many types of attack are being defended by these features as a result it is not possible to fully assess the underlying code for weaknesses should the platform protect fail or be removed."

4.9. The full report produced by Reaper Technologies is at annex B.

4.10. We have now engaged a CESG Check approved consultancy who will be performing end to end vulnerability assessment of the HFEA website, Clinic Portal and Release 2 in addition to penetration testing of the HFEA's perimeter network as each aspect of the Programme goes live. The IfQ Programme Board receives these reports, and further updates will be provided to the Audit and Governance Committee as part of the update reports by the Director of Compliance and Information.

5. Recommendation:

5.1. The Audit and Governance Committee is asked to:

- Note this report

6. Annexes:

- Annex A - Application Security assessment
- Annex B – IfQ security model

Annex B
Audit and Governance Committee 7 December 2016
Agenda item 11 - Cyber Security

APPLICATION SECURITY ASSESSMENT

FOR

HUMAN FERTILISATION AND EMBRYOLOGY AUTHORITY

Security Assessment Summary
28 May 2016

Client Information	
Company Name:	Human Fertilisation and Embryology Authority
City:	10 Spring Gardens London SW1A 2BU
URL:	http://www.hfea.gov.uk

Client Contact Information	
Contact Name:	David Moysen
Title:	Head of IT
E-mail:	David.Moysen@hfea.gov.uk

Consultant Information	
Company Name:	Reaper Technologies Limited
Contact Name:	Stephen Kapp
Telephone:	+447770566687
E-mail:	skapp@reapertech.com

1.0 Business Risk Summary

Overview

A security review was conducted of the new HFEA website, portal and supporting API. The review looked at assessing the services against the OWASP Top 10 to determine if any security issues were present. The following is a summary of the findings and conclusions and recommendations based on these findings.

The security review was conducted between the 10th and 24th May (5 days) and looked at the following services:

- ifq-website.azurenetworks.net
- ifq-portal.azurenetworks.net
- ifqclouddevwebapi.azurewebsites.net

General

In general, the security of the reviewed applications was good, the website and portal leverage the Umbraco CMS platform to provide the frontend elements, with extensions implemented to provide specific features for the HFEA. The Umbraco CMS platform is a web supported .NET based CMS and HFEA appear to be running the latest release. The entire frontend is hosted on Microsoft Azure and uses various elements of the Azure platform to provide services for the application, for example authentication is provided through integration with the authentication services provided by the Azure platform.

During the course of the assessment each of the areas was tested for common security vulnerabilities including those outlined in the OWASP Top 10 the review did not identify any significant issues, however there are some areas of concern as detailed as follows.

Communications Security

The first area of concern was the security of the communications with the HFEA systems, neither the website or the portal required the use of TLS to provide transport security to the application. As a result authentication information for the portal and authentication information for the management of the Umbraco CMS is not protected as it is transmitted over the Internet.

As a result, the lack of transport security means that features such as Strict Transport Security and protection mechanisms for preventing eavesdropping of session cookies are not present.

The backend API however was protected by HTTPS; this was using the default SSL termination for the azurewebsites.net domain.

It is recommended that all externally visible components of the application are secured by using HTTPS to ensure that all information is protected while in transit, with requests to the HTTP version of the applications redirected to the secured versions. Additionally, once this has been implemented implement Strict Transport Security and options on session cookies to further secure the information transmitted.

Information Leakage through Error Messages

During the course of the security assessment a number of error messages where recorded that leaked potentially useful information regarding the environment the application runs within. The error messages in themselves did not leak anything sensitive in terms of the data handled by the applications, however they did leak system information for example one error message leaked the path information for the application on the server and detailed call stack information. This type of information is useful for an attacker, it can provide them useful insight into the application, providing the attacker with a location for files on the system as well as being able to deduce the version of the Umbraco CMS in use.

Information like this can be used to improve the success of other attacks or provide enticement information for areas to exploit. As a result it is highly recommended that custom error handling is implemented to capture errors, log the specifics of the errors to a log file for investigation and return a basic minimal error response to the application user.

Umbraco CMS Configuration

There is an concern within the Umbraco CMS environment, some default content appeared exist without having been removed or default configuration changed. This doesn't follow best practice as recommended by the Umbraco maintainers. It could be

indicative of other elements of the Umbraco code that may not have been properly configured to remove default settings or features.

It is recommended that the Umbraco installation 'hardening' be completed. Ensure that default configurations have been customised and unused features are disabled or removed from the environment.

Conclusions and Recommendations

In general, the security of the application components reviewed was high, as the applications are employing some of the latest technologies from Microsoft they are following good security practices in the main when it comes to the application code with no apparent weaknesses that are covered by the OWASP Top 10, such as Cross Site-Scripting and Injection attacks being handled by the .NET platform security features. The application is employing security features of the platform to provide protection as a result when testing many types of attack are being defended by these features as a result it is not possible to fully assess the underlying code for weaknesses should the platform protect fail or be removed.

It is recommended that the application code undergo a security review to provide insight into the security of the application code at a deeper level. This would be in the form of a source code review and it is also recommended that as part of the build and deployment process a static code analysis step be introduced to provide insight into code issues as the application is being developed with any identified problems being fed back into the development teams to be addressed earlier in the development process.

As the application development progresses with the next phases where patient data being handled it is more important that the security of the application code is reviewed in more depth. Any source code review would look for the common security weaknesses, static analysis tools will help identify these, however another area that would be reviewed by a manual review would be the logic behind the scenes to assess if there are any issues being introduced.

Additionally, to help going forward with the identification of potential areas of concern and to guide remediation and development of security controls and features I would recommend that the next phase of development have a Threat Modelling exercise performed. This threat model would then be used to guide the future design and implementation to ensure that later phases of development address security risks.

2.0 Technical Summary

2.1 Test Area Summary

The following is a summary of the posture of the applications reviewed as part of this assessment.

Vulnerability Area	Brief Description	N/A	Good	Fair	Weak	Poor
Configuration	How secure the configuration of the application is.					
Authentication	Are there any specific issues regarding the authentication of users.					
Session Management and Authorisation	How well the application handles the authorisation and keeps the session secure.					
Encryption	Is there any encryption in place in the application and how well it is configured.					
Data Validation	How well the application handles sensitive user input.					
Error Handling	How the application reacts when an error occurs					

2.2 Test Findings

The assessment findings are included in the accompanying spreadsheet.

3.0 Risk Rating

This report harnesses the power of CVSS v3, the latest industry standard for vulnerability scoring, it combines this with the simplicity of colour coding. This enables access to this report by all levels of management.

CVSS v3 Explanation

CVSS (currently version 3) is the Common Vulnerability Scoring System. This is a vendor independent way of scoring vulnerabilities in a more granular way than just being assigned as a critical, high, medium, low or no (informational) risk.

This system takes a variety of factors (known as metrics) into account such as the level of complexity required to reach the affected system, whether or not exploit code exists, the impact successful exploitation of the issue would have on the business and the type of area of concern (availability, confidentiality and integrity).

By applying these factors to each unique vulnerability, a score from 0 to 10 calculated and assigned.

Reaper Technologies assigns high, medium or low to each vulnerability based on the following criteria as defined by the CVSS v3 standard:

Critical:	Any issue with a CVSS score of 9.0 or higher
High:	Any issue with a CVSS score of 7.0 or higher but lower than 9.0
Medium:	Any issue with a CVSS score of 4.0 or higher but lower than 7.0
Low:	Any issue with a CVSS score of 0.01 or higher but lower than 4.0
Informational:	Any issue with a CVSS score of 0.0

This assures that each vulnerability has been tailored to the client, as each vulnerability affects each client in different ways.

For example, an SQL injection issue affecting a public facing website would be an high risk. That same issue on an internal host with adequate firewall configurations could be classed as a medium risk. A high risk issue on a low impact server may carry a lower CVSS score than a medium risk issue on a critical server.

For more information on CVSS please refer to the First.org website link below: http://www.first.org/cvss/

IFQ Security Model

Identity Solution

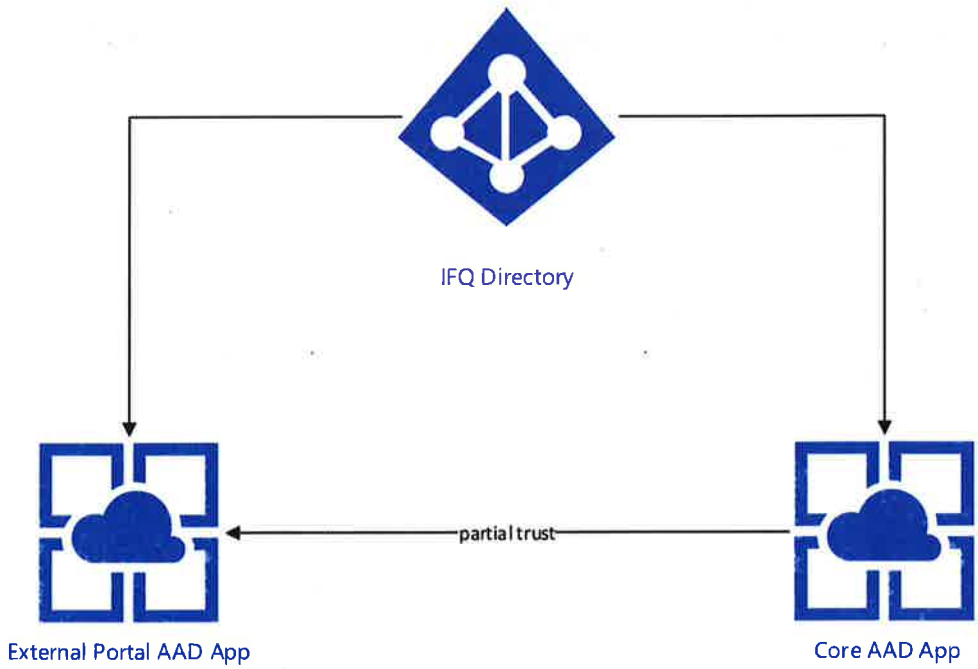
We have chosen Azure Active Directory (AAD) to store user identities. AAD replicates well both with Microsoft Active Directory and third party providers. In addition to that AAD supports modern identity standards such as OpenID Connect and OAuth 2.0, as well as multi factor authentication when required in future.

Azure Active Directory Partitioning

IFQ Application owns a separate directory in HFEA Azure Tenant. That means it's totally separate from any other directories in the Azure Tenant, has its own users, groups and roles. In addition to that a designated people can be assigned to manage this directory with very granular permissions depending on what they need to do.

In general, any system that needs to access AAD has to be registered as an AAD application. Applications is another abstraction which allows to set up more granular access to the directory data. A single AAD can have many applications registered, each with it's own permissions, access keys, roles, and permissions.

IFQ registers at least two applications.



Core App

The first "Core AAD App" is intended for backend use. It has been granted enough permissions to both authenticate users and fully manage AAD. Core App can:

- Create/Update/Delete Users

- Create/Update/Delete Groups
- Create/Update/Delete Application Roles
- Authenticate users
- Change user membership in Roles and Groups

IFQ internal services such as API and Orchestration Layer are a part of the Core AAD app so they can perform these operations on request.

External App

External application is created on demand for any application outside of IFQ network, typically this is an API user, including Website or Clinic Portal. They can be two extra applications or the same one.

External application is registered by HFEA in IFQ directory with minimum permission set possible, it can:

- Sign in a user and read profile
- Access Core App (be able to communicate with it)

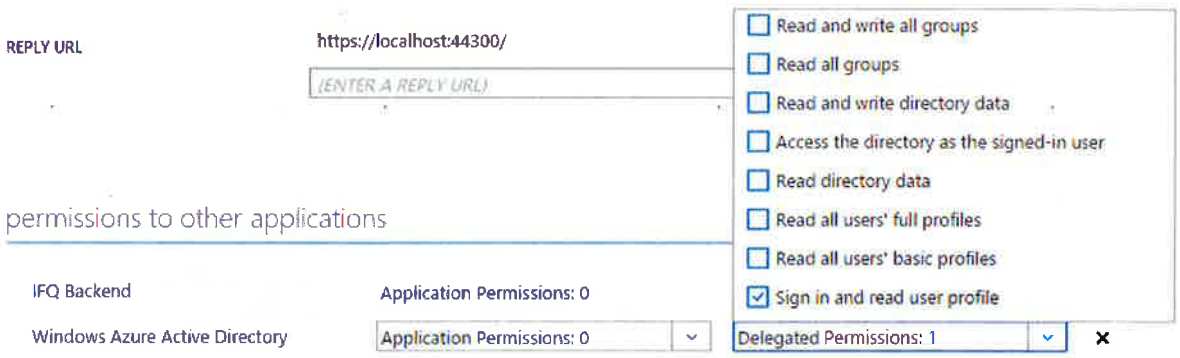
This gives HFEA flexibility in terms of outsourcing the development of external apps and services as AAD app has its own access keys and not enough permission to do anything dangerous.

Trust

HFEA administrator needs to configure minimum trust so that external application can sign-in users and call core application in Azure Portal on the external application configuration page:

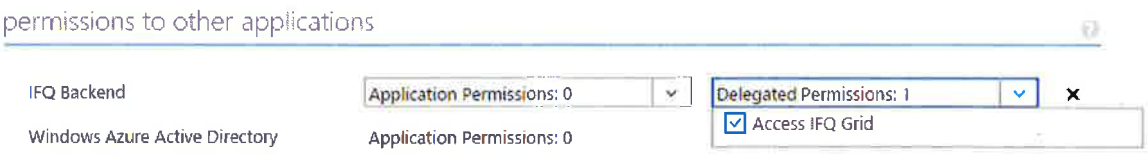
Sign-in

One permission in Microsoft's "Windows Azure Active Directory" application for users to be able to sign-in at all:



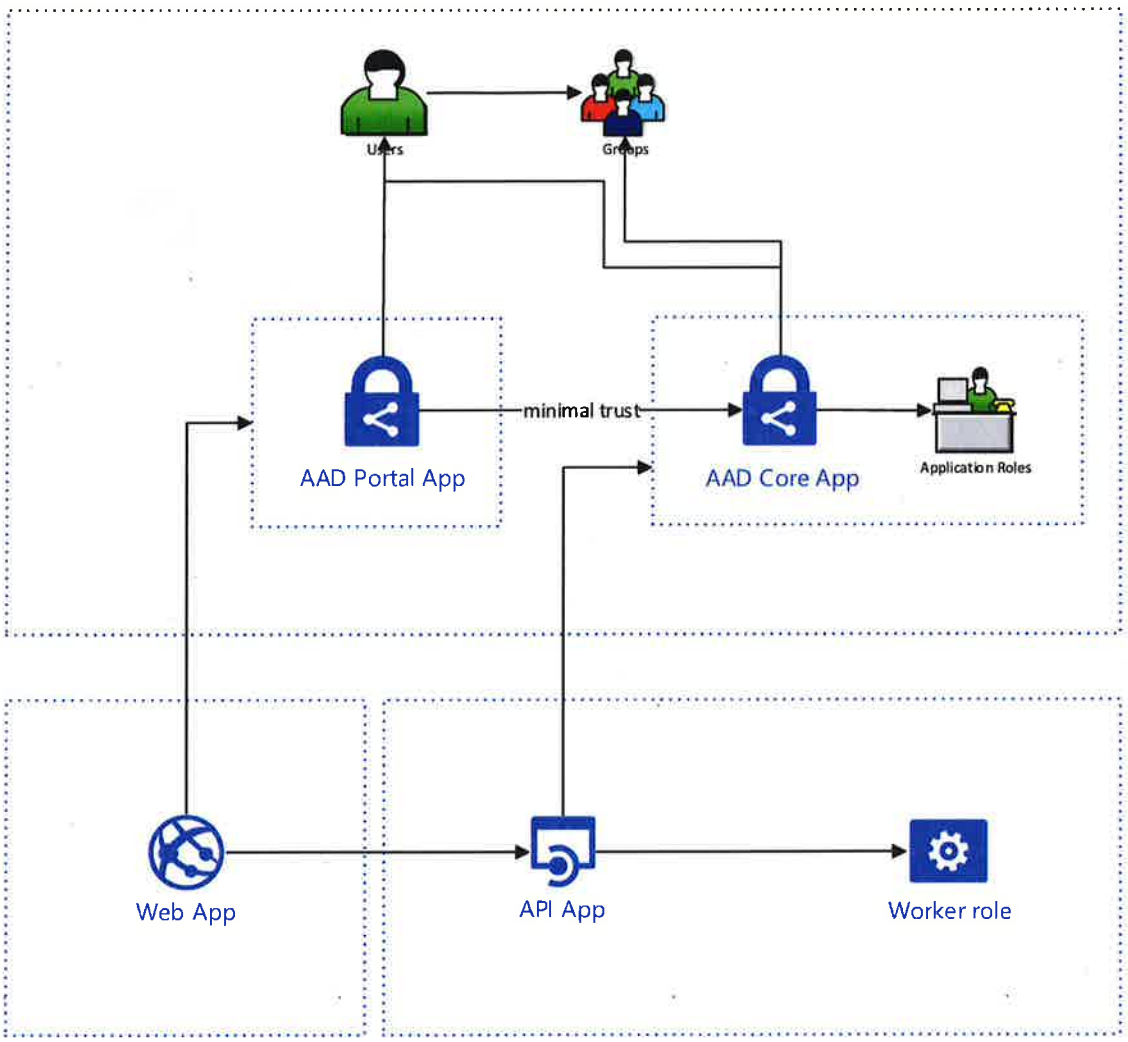
Access IFQ Core App

One permission to our own Core App which is named "IFQ Backend" in this screenshot:



Authentication Sequence

In the diagram below dotted rectangles represent authentication boundaries. In order to cross them a component needs to perform a security operation to have a specific permission.

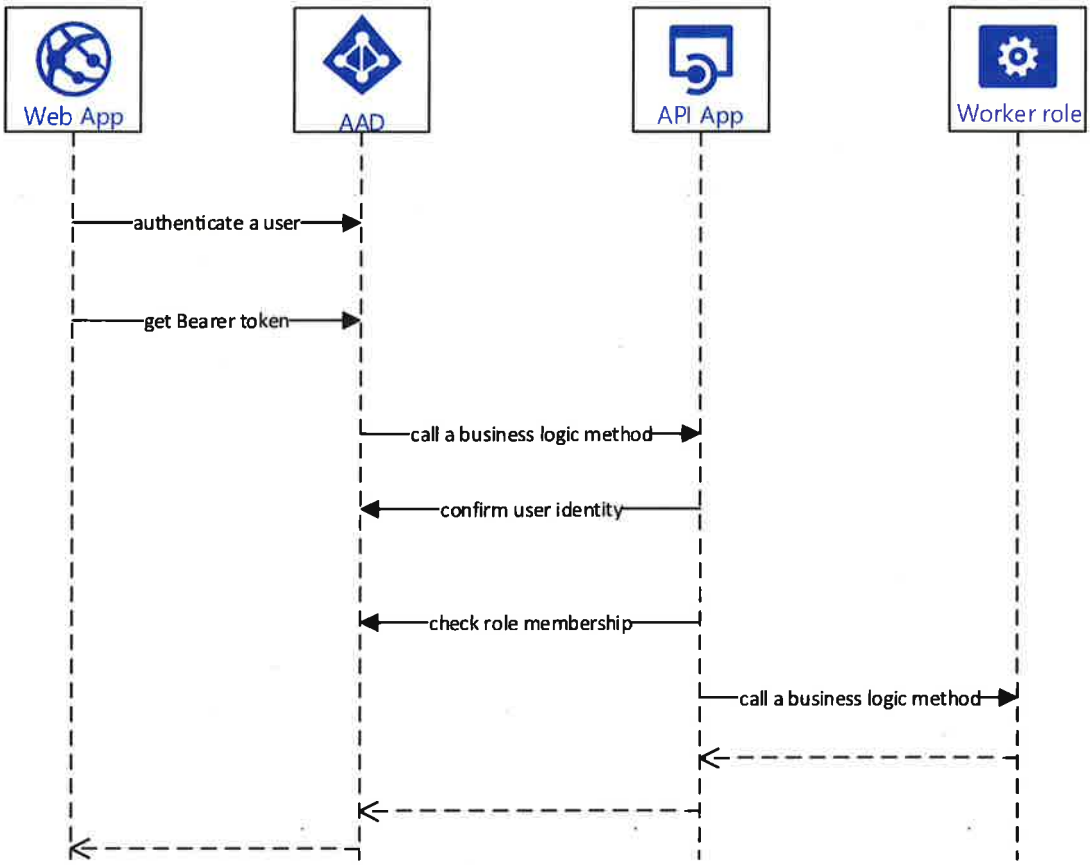


In order for a web site to make a generic call the following authentication parts will be involved:

1. Web App (Website or Clinic Portal) which already has application details for the "AAD Portal App" authenticates a user across IFQ directory. Technically this is done by an OWIN component for ASP.NET MVC supplied by a Microsoft and there is a minimal developer effort.
2. Web App obtains an authentication token and stores session data in browser cookies.
3. Before making the call to API App, Web App calls Azure Active Directory Graph API to obtain a JWT authentication token. Due to the fact HFEA administrator has set up the trust permission AAD returns such a token.
4. Web App uses a simple **Bearer Authentication** to call the Web API. Bearer authentication only requires you to include one extra HTTP header in the call and is supported by most of the web frameworks such as RestSharp.

5. API App validates the JWT bearer token to make sure the user is genuine and authenticates as a user in the Core App.
6. AAD Core App also contains a list of application roles registered for this application. API App performs a check against AAD that the user belongs to an appropriate role and either allows to denies the call. Note that application roles are private to the AAD Core App and external applications don't have access to read them even if they had full permissions in their own app space.

The following diagram illustrates the flow:



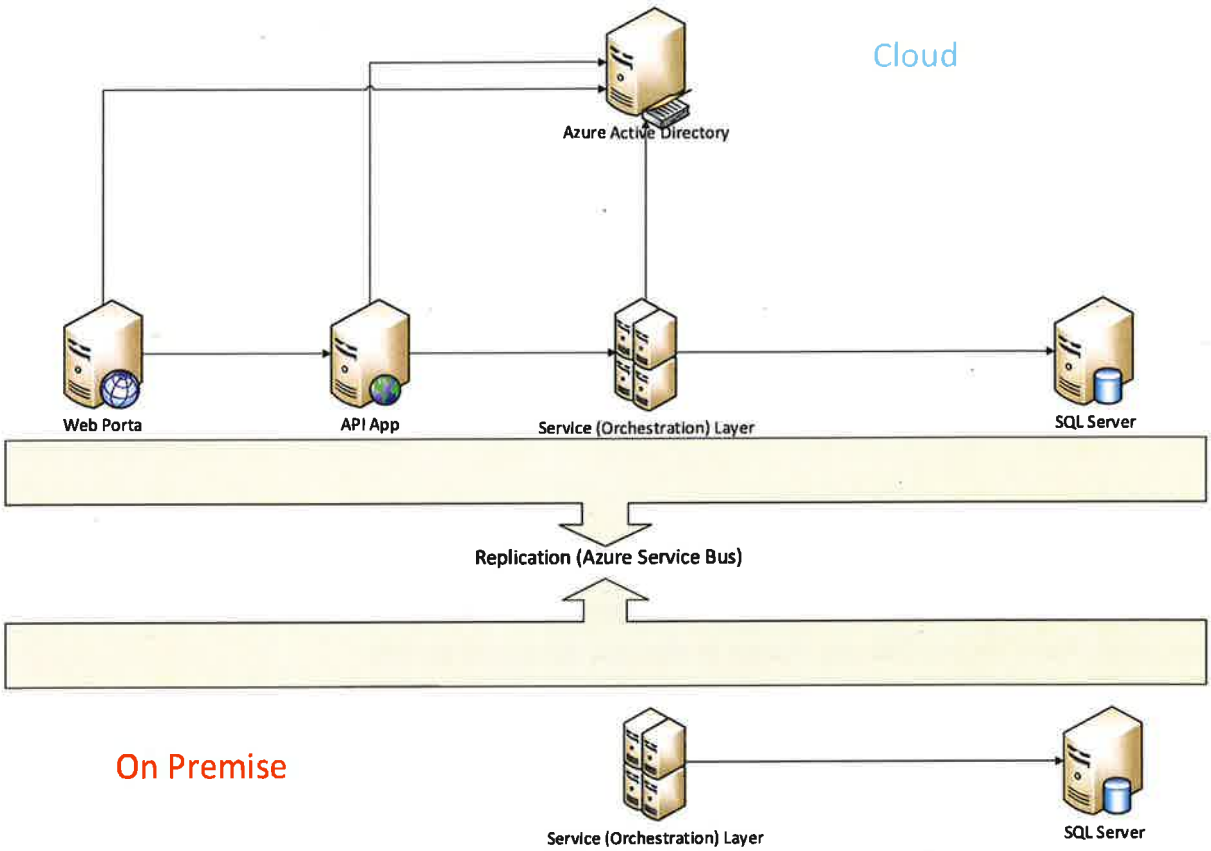
We will provide a full blown easy-to-use authentication library both for internal use and for external developers building applications with ASP.NET MVC.

We will also provide a .NET SDK for calling IFQ REST API, including Bearer authentication support.

Design proposal

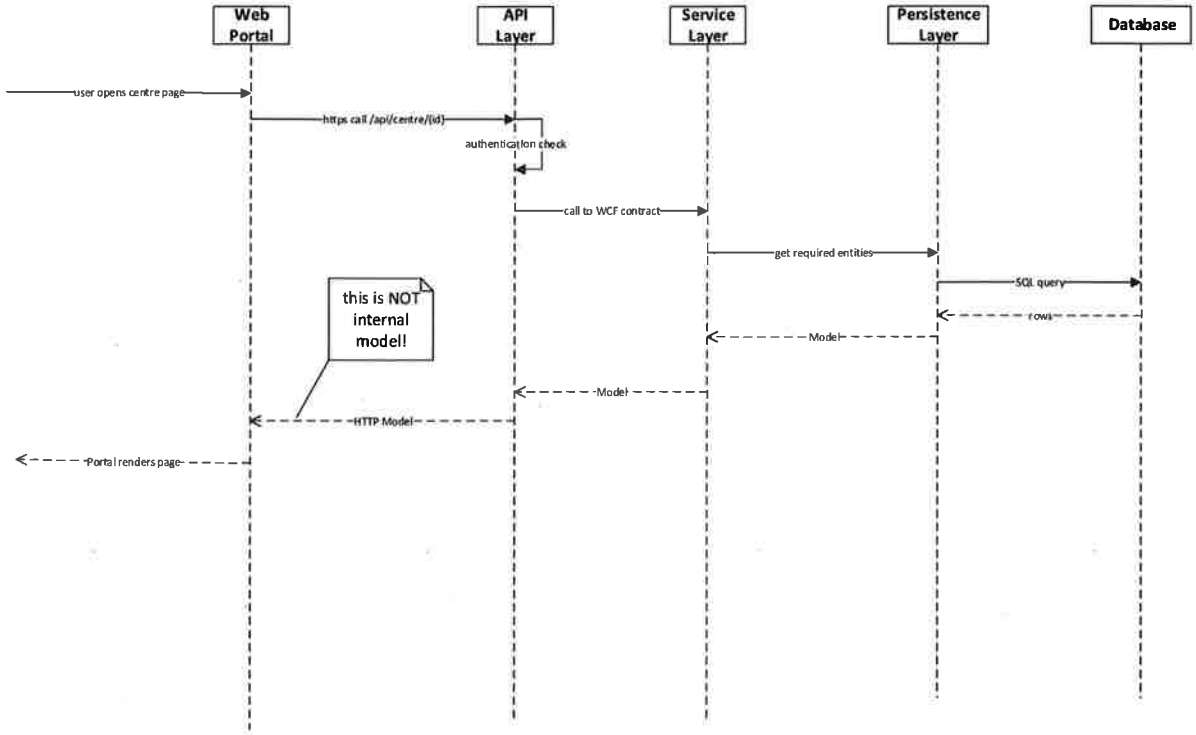
Overview

See the simplified logical structure here:



- **Web Portal.** Developed outside of the scope of this project, is an external caller (such as Umbraco CMS) calling our API App.
- **API App.** The only public facing service accessible outside of the network. This is an ASP.NET Web API project defining and standardising the public and private API functions available. Potentially this app can be wrapped behind the API Management portal which should be discussed separately.
- **Service (Orchestration) Layer.** Contains all the business logic of the IFQ application. Any API apps or any additional public services must call this layer to perform any business operation. This layer exposes some high level functionality via it's WCF contracts.
- **SQL Server.** Existing sql database with existing tables.

To display a typical web page, the system will involve all the parts in this order:



Due to the fact that parts of the system stays on premise we do replicate some data between the two sides. Azure Service Bus was chosen as the best options to do that.

Solution / Project structure

All the code is written in C#, with the exception of occasional T-SQL stored procedures or queries depending on the application needs. Some code may be written in PowerShell or any other scripting language required for Microsoft Azure deployment scripts.

The solution should have the following projects (approximately):

- **HFEA.Model** – contains all the model classes following the DDD strategy (see below).
- **HFEA.WebApi** – the API app (ASP.NET MVC WebApi with Swagger support). This application uses its own model specific to the REST/SOAP calls it exposes to the public.
- **HFEA.DataLayer** – the data access layer project. Interfaces for accessing the data must be defined in the HFEA.Model project and use the Model in method parameters and return results exclusively. Internally we will use the latest version of the Entity Framework (v6 at the time of this writing). We will utilize code first approach as much as possible here.
- **HFEA.SDK** – contains all the business logic.
- **HFEA.WebPortal** – a test portal which demonstates some of the functionality of the application, and contains some administration functionality (for example adding users, assigning roles etc.)
- **HFEA.Tests.Unit** – unit tests only.
- **HFEA.Tests.Integration** – integration tests only.

- In addition to this there will be an extra project or two which hosts business logic. Depending on where we host the logic (on premise or in Azure) these can be either Azure Worker Role or a Windows Service application. To test this locally we can use Compute Emulator coming with Azure SDK.

Code Structure

HFEA.Model

Collection of model classes describing IFQ domain and according to DDD:

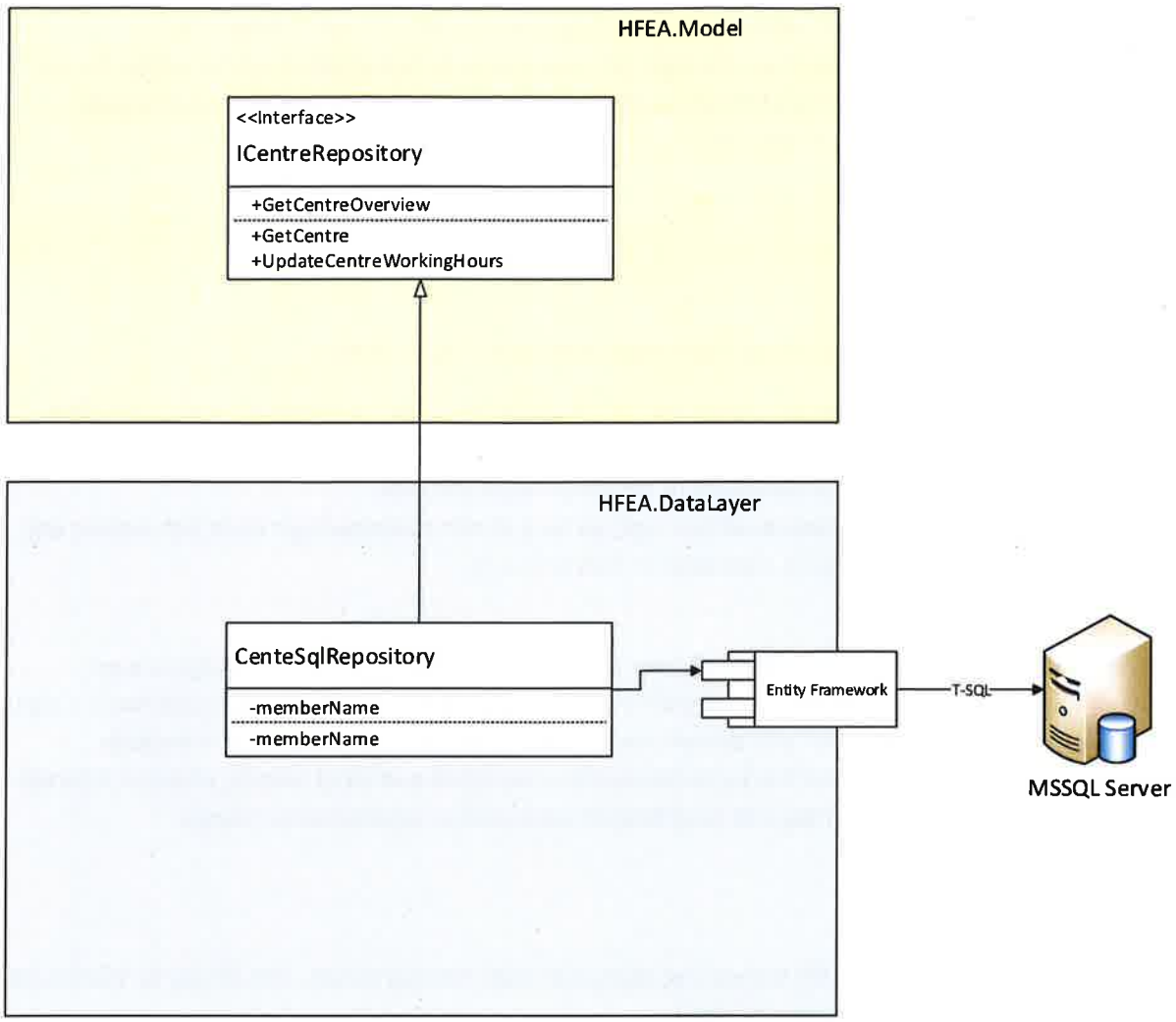
- Not used or exposed in any sense outside of the IFQ solution to external world, including Portal and public website.
- Not compromised by serialisation or database requirements.
- They can and should have business logic, as long as this business logic does not require any external dependencies i.e. database or network calls.

HFEA.WebApi

ASP.NET WebApi application i.e. it *doesn't have a frontend* other than Swagger. Swagger is an industry standard for API discovery, cross platform and widely adopted. This application has it's own model exposed via REST or SOAP and doesn't expose HFEA.Model in any way. This is because external model can't change and has to be backward compatible and dead simple, whereas internal model is rich and is subject to frequent modification as business requirements change.

HFEA.DataLayer

This is a C# library responsible for translating repository calls into sql server. The library is referenced later by a process running the business logic.

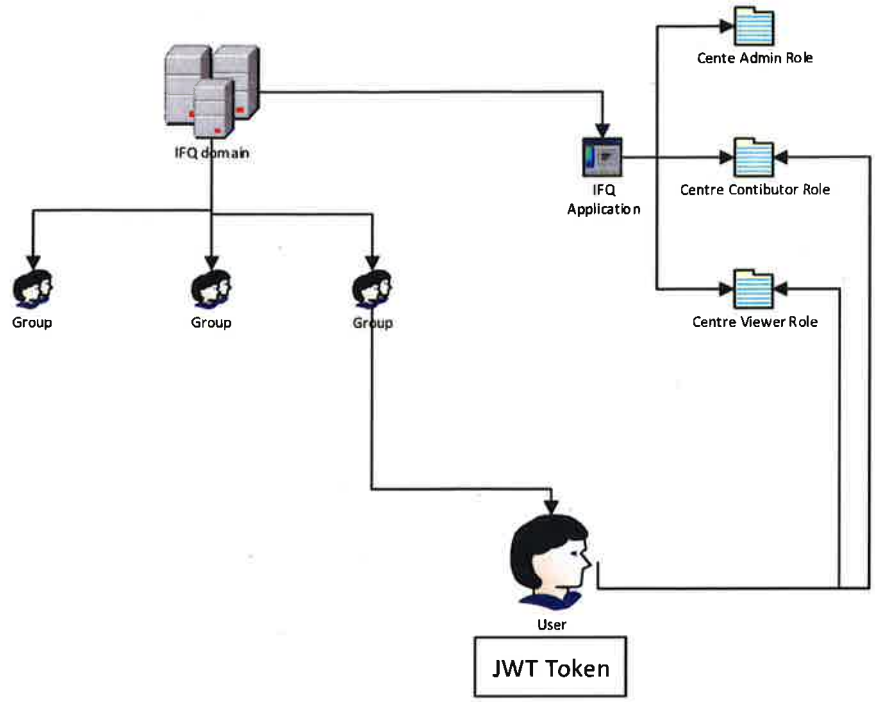


Actual repository interfaces are declared in HFEA.Model and DataLayer only implements them following these practices:

- A repository is a DDD repository with all the attributes implied:
  - Repository does not have any business logic and is only designed to store/retrieve data
  - Repository always operates with Model classes, i.e. repository methods accept model classes as parameters and return model classes as result – it's the DataLayer's responsibility to translate Model into underlying database technology.
- We will use EF6 and code-first to map database to model.

Identity Model

We have chosen Azure Active Directory (AAD) to store user identities. AAD replicates well both with Microsoft Active Directory and third party providers. In addition to that AAD supports modern identity standards such as OpenID Connect and OAuth 2.0, as well as multi factor authentication when required in future.



IFQ Application (see diagram below) is registered in IFQ domain. In terms of AAD application is a separate entity which can hold application specific roles and control user access separate to the primary domain. Application also allow to control directory access in a finer grained way comparing to using the global directory.

IFQ application holds application specific roles which can be added or removed dynamically. Once a user is authenticated against AAD his identity token (JWT – JSON Web Token) carries his identity and roles/groups he belongs to which allows us to do proper authorisation on API calls and other resources.

Authentication scenarios

- Web Portal application developed externally authenticates to AAD itself, then passes the auth token to our API App call which performs authorization checks.
- An external developer authenticates to AAD and passes the token.

Data Replication

It's worth mentioning that data must be replicated only due to the fact that system can't be hosted 100% in Azure Cloud. When this is not the issue we don't need two separate databases on each side and the whole system can use just one single primary replica.

Not all of the data will be replicated, but only a subset required for the API calls. For comparison, there is around 20Gb of SQL table data on premise and only around 300Mb in the cloud.

Database schema (at least for the subset of replicated tables) is identical both on premise and in the cloud, however we don't have to keep SQL server in the cloud and free to choose any technology.

Design decision

- Due to the fact that cloud database may disappear in future as the whole solution potentially can be hosted in Azure Cloud we've decided to keep SQL server and not to use anything more modern. When the physical boundary is removed queries for Azure side should just work with the big on premise database meaning no code changes.

- Due to the fact that the database on premise can be changed by anyone directly avoiding service layer, we don't have a reliable way to intercept the events to send the replication event to the cloud. There are a few options here:
    - Add trigger functions to SQL server which will notify SL on changes so we can push it further.
    - Intoduce a call in SL which external callers have to call in order to notify they have changed particular data.
- We still need to see the database in action to make the best architectural decision here.

Logging

All the layers must log as much as possible. Any sort of problem or failure must be able to be replicated in the dev environment by analysing logs.

When there is no preference on any particular logging frameworks I would recommend choosing one of two: log4net or NLog.

Scope for Release 1

- Build a deployable solution which can be pushed to Windows Azure with Continuous Integration.
- Data Replication between cloud and on premise systems. Depending on the complexity and amount of work we may not finish all of it but will build a good framework with best practices.
- Some or all the API calls implemented in all layers. Due to the fact we have no idea how many calls we need to implement or how complex they may be we will do the most.

Methodologies

DDD (Domain Driven Design). DDD is one of the most successful design approaches to tackle easy to most complex problems in building software. The most important principle is that our business domain is described in terms of model classes. DDD is getting even more popular in the Micro services world.

TDD (Test Driven Design). Each new functionality should have a minimal test written before it's implemented. Most suitable frameworks of choice:

- NUnit – general unit/integration test framework
- Moq – mocking framework

HFEA

Release 2 Solution Architecture

Version 1

Ivan Gavryliuk
9-15-2016

Contents

Problem

R2 requires a certain security guarantees which would impact the architecture for the next part of the system. Some of them are:

- Data cannot leave UK premises
- All the systems need to be closed down, except for those which need to be accessed by the public
- All the public system must have a fine grained security rules implemented

Most of the public cloud providers such as Microsoft Azure or AWS do not have physical data centres in the UK which prevents us from hosting data there. However, Microsoft Azure is building one at the moment where we have a private preview access. If that happens the overall architecture and costs are considerably lower than creating a hybrid system.

Generally, HFEA system consists of several basic layers:

- Persistence. Microsoft SQL server (relational data) and large files (binary data).
- Business Logic (Orchestration Layer).
- Public API for EPRS access.
- Clinic Portal Website.
- QA Website.

We use both IaaS and PaaS Azure offerings, balancing between security, support costs, ease of use, and developer productivity.

Also we assume that all of the services are built with .NET Framework and T-SQL.

Security

Authentication

Release 1 (R1) is already using Azure Active directory B2C for authentication (see R1 architecture document for choice decisions). Due to the fact that same users will be able to access R1 and R2 systems it is a natural choice to reuse the same directory. This allows to reuse technology, experience and frameworks built around B2C.

However, QA Website app requires authentication with HFEA Active Directory by internal staff, this is the same directory used to login to Windows or access Office 365 accounts. It is also replicated to Microsoft Azure Active Directory, making it simple to use the Cloud version for authentication from Azure data centre. We assume replication is already configured and is outside of the scope of this document.

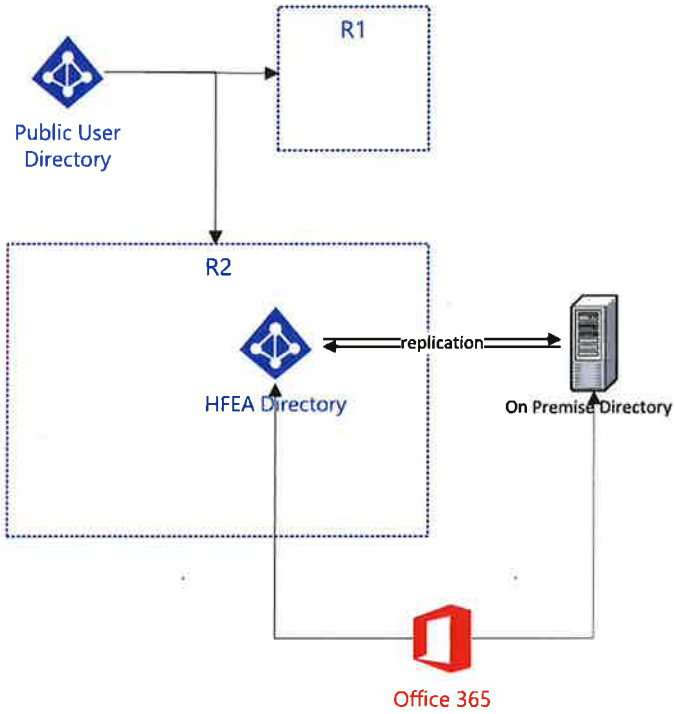


Figure 1 – Active Directory Relationships

Network Security

Considering we are using Azure UK data centre the whole system is hosted physically in the UK. To achieve low latency, high performance and restrict the network from the outside attacks R2 will utilise Azure Virtual Networking technology. It allows customers to replicate physical network in Azure data centres as they would on premise.

Azure Virtual Network is a dedicated workspace for services which is never visible from outside world, unless specifically told by configuring integration with outside world, just like on premise network.

R2 virtual network never talks to HFEA on premise network, and the only piece of information they share is the Active Directory information replicated.

A subnet in a virtual network is a range of IP addresses, with its own security rules and routing tables.

The Virtual Network is divided into 4 subnets (see Figure 2 - Network Diagram):

- **Core.** Contains all the internal (core) services of HFEA application. This subnet does not have outside access from anywhere except for *Orchestration Layer* from other subnets. None of the services in this network have outside internet access. Contained services:
    - Microsoft SQL Server Machine.
    - Orchestration Layer Service.
    - Image Storage.
- **EPRS.** Hosts Public REST API used by EPRS providers.
- **Internal.** Contains QA Web Application used exclusively by internal HFEA staff. Theoretically it could live in the Core subnet, however for easier access management it is recommended to create a separate subnet.
- **Gateway Subnet.** This subnet is used only for administrative access to the parts of the system and can be accessed by individuals using Azure VPN Gateway. Required mostly for troubleshooting and access by software developers. Note that this subnet doesn't have NSG attached which is a requirement for VPN connection to work properly in Azure.

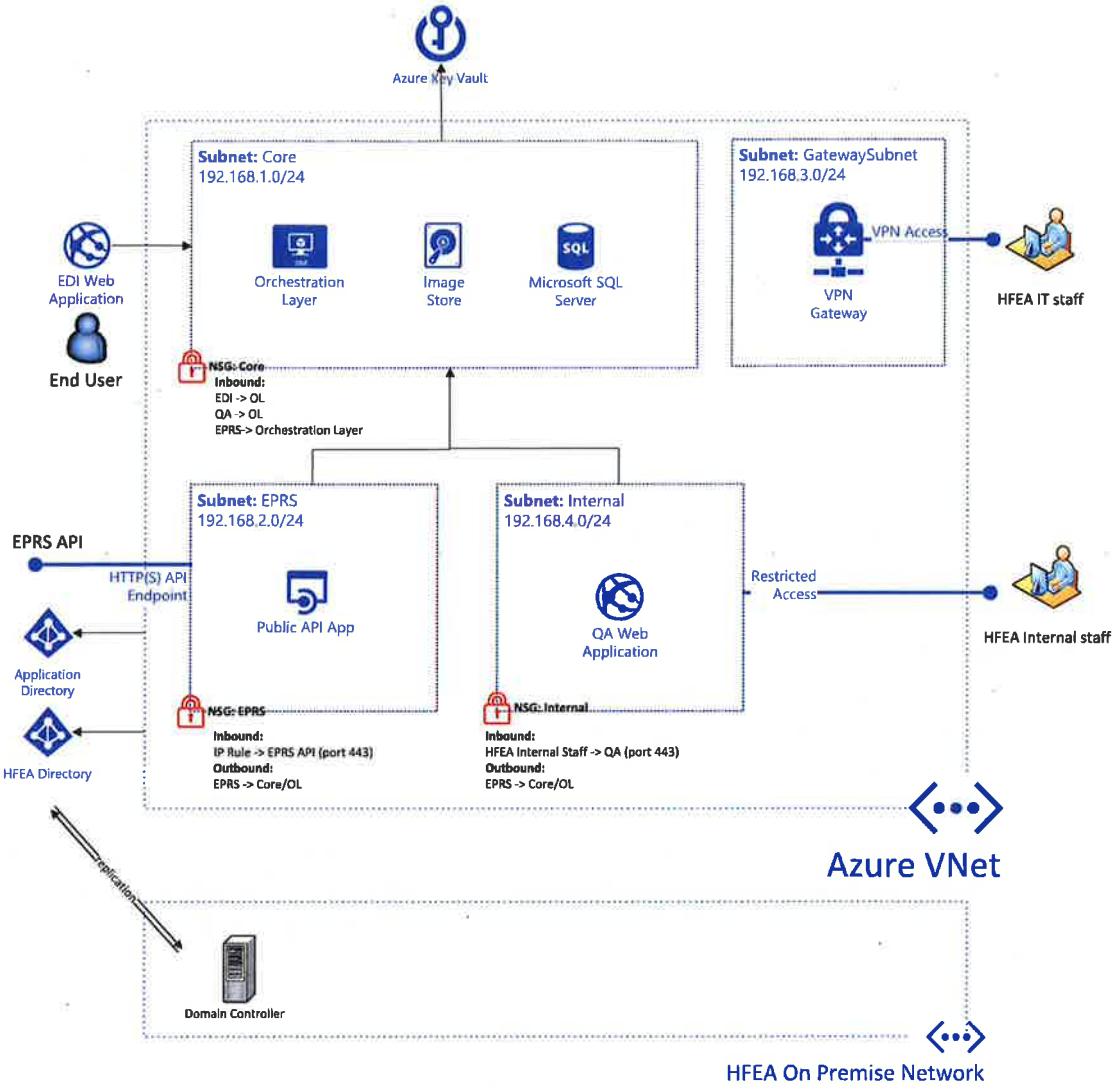


Figure 2 - Network Diagram

Core Subnet

Due to the geographical limitations that data must not leave the UK and be encrypted at rest we cannot use traditional Azure Storage or Azure SQL Server.

SQL Server

We will rent a Virtual Machine with SQL Server 2014 SP1 from Azure Marketplace on PAYG license which is a usual VM with all capabilities a normal VM has. This can be joined to our Virtual Network in Storage subnet with restricted access only by Orchestration Service from the Business Logic subnet. SQL Server database is encrypted[1] using Transparent Data Encryption[2] (TDE) for IaaS instances. Encryption has to be configured manually after installing SQL Server instance[3].

Image Storage

Image Storage is a set of files falling under the same security restrictions. There are two major options in Azure to protect file data:

1. **Azure Storage Service Encryption for Data at Rest** enables encryption on blob storage on the Azure Cloud side. Encryption is handled
2. **Azure Disk Encryption**. Encrypts OS and data disks on a VM level using BitLocker on Windows or DM-Crypt on Linux. Raw data never leaves VM boundaries and customer secrets are stored in Azure Key Vault service. It is the safest choice for maximum security.
3. **Custom Encryption.** Involves writing a software component in the Orchestration Layer for handling encryption from the software.

Table below outlines the pros and cons of every approach.

	Azure Storage	Azure Disk Encryption	Custom Encryption
Encryption at Rest	Yes	Yes	Implementation
Transport Encryption	HTTPS only	Not applicable	Implementation
Implementation Effort	Trivial	Hard (infrastructure configuration)	Hard (developer resources)
Deployment Effort	Trivial	Hard	Trivial
Encryption Method	AES 256	AES 256 (BitLocker)	Implementation
Overall Security	Medium	High	Medium-High
Scalability	Up and Out	Up Only	Up and Out

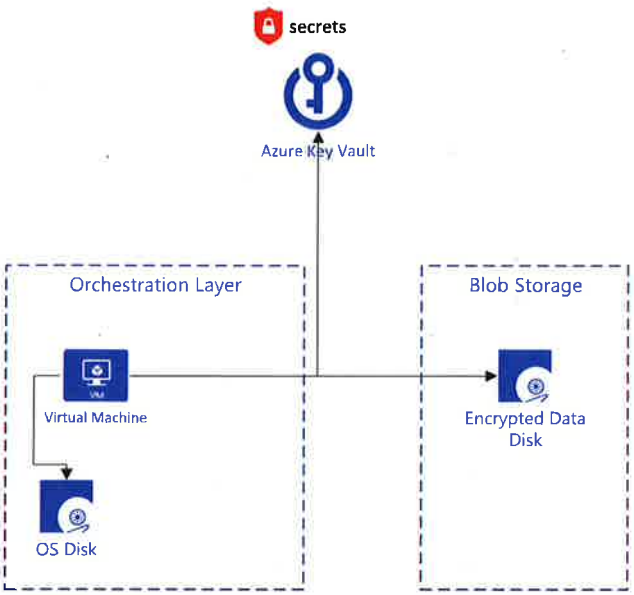
Based on the comparison and requirements I would strongly recommend implementing storage with Azure Disk Encryption. Read more: https://azure.microsoft.com/en-gb/documentation/articles/azure-security-disk-encryption/.

This is not a scripted process and has to be configured manually.

---

[1] SQL Server Encryption, https://msdn.microsoft.com/en-us/library/bb510663.aspx
[2] Transparent Data Encryption, https://msdn.microsoft.com/en-us/library/bb934049.aspx
[3] Configuring SQL Server Encryption, https://blogs.technet.microsoft.com/kv/2015/01/12/using-the-key-vault-for-sql-server-encryption/

In essence when configuring a disk encryption a new data disk is attached to Orchestration Layer machine with a new drive letter. Software running on this machine can access the drive as it would normally do and configured Windows OS takes care encrypting and decrypting data on the go.



One of the limitations of this approach is it's not scalable for the reason that a disk cannot be shared between two or more instances of virtual machines. For that reason, Orchestration Layer machine can only be scaled up.

When workload will reach a certain limit in future OL can be moved to a separate VM instance and existing instance can act as a file server freeing up resources for the actual business logic processes.

In order to scale even more you can implement custom encryption in the Orchestration Layer endpoint which will allow to scale out infinitely, but none of the scaling options are included in this release implementation.

Orchestration Layer

Unlike R1 where Business Logic was hosted in an Azure Cloud Service we can't do this anymore, because Cloud Services have a public access and require creating a public IP address for communication purposes. There are many options available to solve this problem, however considering a low load and security restrictions the most appropriate one is using Azure Virtual Machines. There are a few pros and cons to using VMs.

Pros:

- Cloud provider agnostic, every single cloud provider supports Virtual Machines and running executables which is what a windows service is.
- Flexibility when configuring security and installing custom software.
- Lower cost comparing to other alternatives.

Cons:

- Management overhead comparing to Cloud Services or Service Fabric.
- No auto deployment built-in (but you can utilise existing *Octopus Deploy*[4] instance)

In the first release we will use one virtual machine, however you must consider scaling options (see the section) before going to production or when performance is going bad.

Orchestration Layer is a Microsoft Windows Service application running in a background and hosting all the business logic and model. Other layers provide only integration or user interface services.

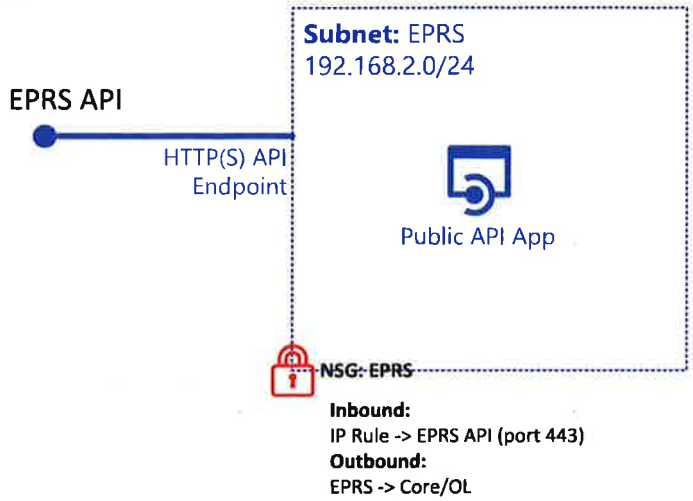
Orchestration Layer both hosts services for API or internal applications via Windows Communication Foundation (WCF) and performs long-running background tasks when needed.

There are no special security requirements for OL VM, it doesn't have outbound Internet access and only needs to access SQL server database.

[4] Octopus Deploy, https://octopus.com/

EPRS Subnet

Figure 3 - EPRS API



EPRS API lives in a separate subnet only for security reasons. EPRS provide API access for external clients to their dedicated centres and have a key-based authentication. In addition to that EPRS service has only port 443 (HTTPS) open for internal calls.

HFEA is responsible for giving out keys to the EPRS customers. Every key has access to only one single centre. For increased simplicity and cross platform support (we suppose that EPRS clients can use a multitude of different client platforms) we recommend using Hash-Based Message Authentication Code (HMAC) with SHA-1 algorithm which is still considered to be secure and performing well.

Key Authentication

Generating HMAC-SHA1 key

In Windows Server the key can be generated by calling to Cryptography API, see MSDN Documentation. The key then must be assigned to the client by putting in SQL Server database on HFEA side and transferring to EPRS user.

Calling EPRS API with security key

In order to call EPRS client have to sign their request before issuing. The signature is generated in a few steps:

1. Get the current clock time in UTC – **T**.
2. Get your centre ID – **C**.
3. Format a signature string as "C:T".
4. Compute HMACSHA1 hash of the signature string using your key and represent as a BASE64 encoded string.
5. Add HTTP header "x-hfea-date" to request. Note that you have to agree on time format on both client and server side, the most common is (C# example): T.ToUniversalTime().ToString("ddd, dd MMM yyyy HH:mm:ss UTC");

6. Add HTTP header "x-hfea-signature" to request by setting the value of the string computed in step 4.

Validating EPRS API call on the server

The validation can be performed in a few simple steps too:

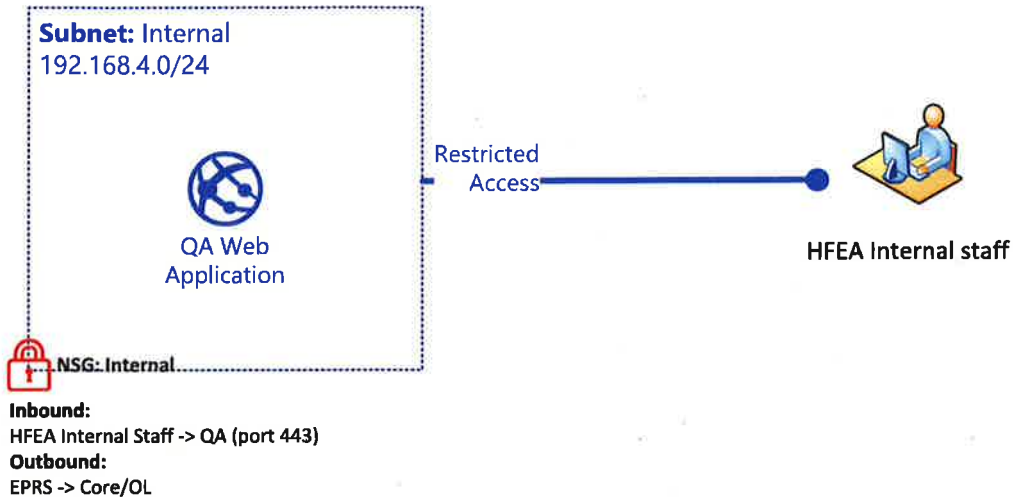
1. Check that both *"x-hfea-date"* and *"x-hfea-signature"* is present, otherwise reply with 401 (Unauthorized) code.
2. Take the current clock time and compare to *"x-hfea-date"* to check if they differ a lot. If the difference is more than 1-minute reply with 401.
3. Compute HMACSHA1 hash of the request following steps 1, 2, 3, 4 in "Calling EPRS API with secure key" section. Remember to take "x-hfea-date" for date value as current clock will be different from client's clock.
4. Validate if your hash is matching client's hash. Deny authentication if they aren't and allow otherwise.

IP Address Restrictions

In order to restrict access from certain IP addresses or ranges a built-in functionality of Network Security Group EPRS can be used to create Inbound Access Rules[5].

[5] Managing Network Security Groups in Azure, https://azure.microsoft.com/en-gb/documentation/articles/virtual-networks-create-nsg-arm-pportal/

Internal Subnet



This subnet hosts internal QA Application accessed exclusively by HFEA Staff.

For authentication it uses Azure Active Directory replicated from the on-premises directory. It is not to be confused with Application Directory used to access the system from the Internet.

QA Web Application is also an IIS hosted solution on Windows Azure VM.

It has a direct access to Orchestration Layer from the *Core* subnet to interact with data. The access is controlled by Network Security Groups.

EDI Web Application

EDI Web Application

End User

Is another part of IFQ visible from outside (the other one is EPRS API). It's hosted as a PaaS solution on Azure Platform[6]. PaaS is possible in this case because the target audience is internet facing and there are no special security requirement except for a strong authentication.

Due to the fact we are using Azure B2C to offshore authentication process it's satisfied too.

Having this hosted as PaaS also gives us better flexibility in terms of scaling, deployment and monitoring.

EDI connects directly to the Orchestration Layer by connecting to our Virtual Network[7].

[6] Azure Web Apps, https://azure.microsoft.com/en-gb/documentation/articles/app-service-web-overview/
[7] Integrate your app with Azure Virtual Network, https://azure.microsoft.com/en-gb/documentation/articles/web-sites-integrate-with-vnet/

Scaling Options

Virtual Machines

Most of the services are hosted inside virtual machines, which have two major scaling approaches.

Scaling Up

Scaling up involves adding more virtual resources depending on where the bottleneck is in current performance scenario (CPU, Disk, Memory etc.). You can rescale the machine instance in Azure Portal.

Scaling Out

Scaling up is not a perfect option and always comes with it's drawbacks:

- During scaling up virtual machine goes offline for a short period of time, and needs warming up again to start your hosted application
- Scaled up instances are expensive, as they always billed for provisioned resources regardless whether they are in use
- Scaled up instances can't provide enough continuity – when the service fails a manual intervention is required.

Scaling out comes to the rescue and covers all the bad points. Essentially scaling out means cloning a virtual machine to run two or more instances. Every instance has its own virtual internal IP address, regardless whether it's public or private facing.

Client of scaled out instances are not aware of the change; they are calling a virtual endpoint called a Load Balancer. Load Balancers can be internal or external, which only indicates which sort of IP address they are assigned to (public or private). For instance, Orchestration Layer will use internal load balancer as it's not publicly visible from the Internet, whereas all of the remaining services are using External Load Balancer.

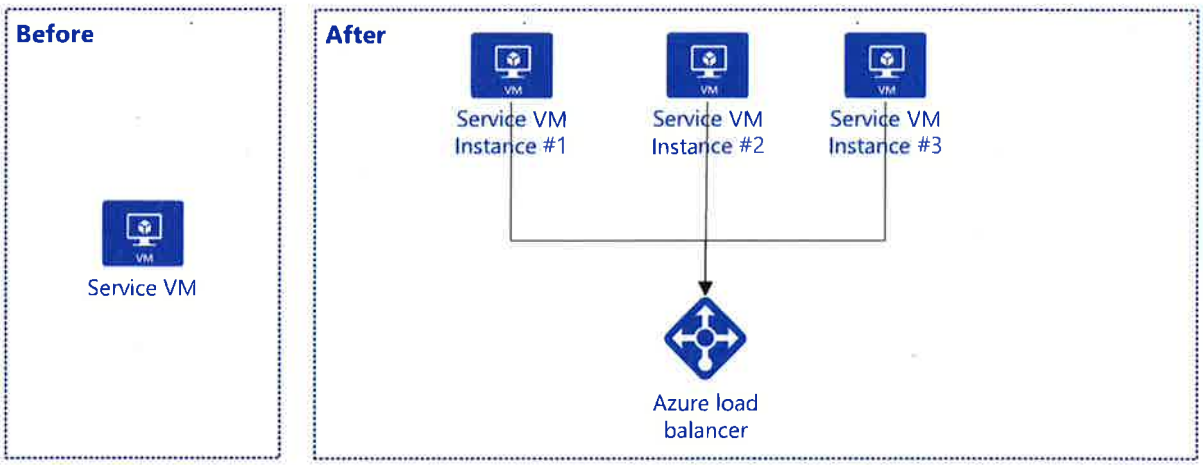


Figure 4 - Without and With Scaling out

EDI Web Application

This application is hosted inside Azure App Service container, therefore scaling is trivial as it's baked into App Service package and works out of the box[8].

[8] Scaling Out Application Plans, https://azure.microsoft.com/en-us/documentation/articles/insights-how-to-scale/