Version: 0.1b
Owner: Stuart Harris
Last updated: 6/11/2019
Status: Draft

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

MHRA - Medicines Information Portal v0.1

High level design

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

Updates

6th November 2019

In the spirit of evolutionary architecture, we've updated this design so that the temporary wordpress site will now only be used for uploading Public Assessment Reports (PARs). Only staff with admin capability will be able to access the site and it will only be visible within the MHRA organisation. The "importer pod" will now import PARs from the wordpress API (as well as the SPCs and PILs from the Sentinel batch export).

This has several benefits:

- The Wordpress site would not be accessible by the public
- Overall security is significantly improved as no external access or login capability required
- A more consistent user experience
- No duplication of the surrounding website content
- A single search index and user experience for all docs (SPC/PIL/PAR)

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

Introduction

The goal of this paper is to describe how MHRA (the Agency) might begin to embrace a microservices approach to digital service delivery, beginning with a low-risk and well isolated **Medicines Information Portal**. It aims to show that by adopting an industry standard approach we can form a foundation for building an ever evolving platform that can support *all* of the Agency's needs well into the future. It will support moving existing functions from legacy hardware and legacy software, together with the evolution of new services to support the agency's ambitions for decades to come.

The agency is undergoing an operational transformation (OT) and has engaged with Red Badger to help align a digital transformation to support this. It is vital that we take a wider strategic view and start off in a direction that will continue to support all future requirements, whilst significantly driving down costs and reducing risk. It is a journey that starts small and continuously evolves by taking lots of tiny steps, gradually building a set of modular microservices, each independently deployable and each with its own roadmap. We will discuss how microservices enable this approach in detail below.

In the meantime, there is an urgent need to provide a service through which the public, health care professionals (HCPs) and industry consumers can find Summary Product Characteristics (SPCs/SmPCs), Patient Information Leaflets (PILs), Public Assessment Reports (PARs) and Safety PARs (SPARs). This is part of the OT portfolio for Customer.

This document recommends that we begin our journey by starting to build a new Medicines Information Portal (MIP) and associated medicines microservice. The medicines microservice will provide an API that can be consumed (at web scale) by any 3rd party as well as users and systems within the agency.

This is an API-first approach, and a wide variety of highly functional customer journeys can be supported by this API and implemented as web (and native) components that sit in a Component Library, which in turn forms part of a Design System. These components carry consistent experience, brand and accessibility requirements from the design system into any user experience in which they take part. The components can be slotted into any web page or mobile app and will talk directly to the API. It may well be that the surrounding content is managed by a Content Management System (e.g. Drupal) in the future, but this is not needed, or desired, for this immediate requirement.

We can use the MIP and medicines microservice to demonstrate how applying the principles that we discuss below can digitally transform the agency and allow it to cheaply and effectively serve its customers.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

In the spirit of Continuous Delivery (CD), we are proposing that the first Minimum Viable Product (MVP) release (of SPC/PIL) to customers would be a static copy of the existing website containing a medicine search component, which talks to a minimal medicines API. The API would be fed passively by tapping into the batch export process from Sentinel to Stellent. It would then allow Stellent to be fully decommissioned. Initially we would continue to use Sentinel to manage the documents and their metadata, but would then also remove Sentinel from the equation by updating the API, and providing UI, to allow the agency to manage these documents, making the service a system of record.

For PARs, which do not come from Sentinel, a temporary internally facing Wordpress site would allow these documents to be managed. This is largely already built so it makes sense to use it to start with. These PARs would be imported into the same search index as the SPCs and PILs.

This document describes the MVP, and how the service can then be evolved into a complete and unified experience that is a pleasure to use, for all the relevant users of the portal. Importantly, it allows from the start, for the service to be fully resilient, highly available, effectively infinitely scalable, and with performance supporting response times of around 100ms. It also paves the way to carve SPC/PIL management out of Sentinel, meaning that there will be one less thing to worry about as Sentinel is deprecated. The medicines microservice would become the system of record for all data and documents (including their metadata) related to the medicines that the agency regulates. The Medicines Information Portal would become the authoritative human interface to this API, whilst allowing other user experiences to be built on it, both within and outside the agency.

Principles

The agency would like to become lean and agile in order to allow it to be effective at digitally serving itself and its customers as quickly and as cheaply as possible. There is no reason why it cannot be as effective as startups in this regard.

There are many principles that we should adopt that will support this vision. Much is written about them on the Internet, which we will link to as we go, so there is no need to describe each of them again in detail.

We must adopt a continuous mindset, which means that we continuously discover, test, build and deploy. We want to get value to customers quickly by building a Minimum Viable Product. Then we use Continuous Delivery to quickly evolve this product into a delightful experience for the users (often called a Minimum Desirable Product). This evolution happens in tiny iterations and continues way beyond any “minimum products”, and potentially forever.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

It allows us to respond quickly and cheaply to changes in our business and in our customers' expectations. It also avoids the need to forklift upgrades or major product replacement activities. This project has often been called the "Stellent Replacement project". We want to ensure that in 3-5 years time we are not embarking on a "Drupal Replacement project".

Being continuous means that if we were building a car, we wouldn't build the wheels, then the chassis, then the body. Instead we would build a skateboard and use that for a while, before evolving it into a scooter, then a bike, then a motorbike and then a car. We would get value early on from using each product, and ensure that each is better than the previous, and more suited to our needs.

In order to be continuous we must have full automation throughout. This means that we need to treat everything as code. When everything is code, it can all evolve together. Everything means everything – there is nothing about the overall system that is not code. It includes Infrastructure as Code, code as code, pipelines as code, tests as code (automated unit tests, integration tests, functional tests), build as code (e.g. Dockerfile), configuration as code (e.g. 12-factor), orchestration as code (e.g. kubernetes manifests), deployments as code (e.g. Gitops), security as code (e.g. Istio policies).

When everything is code we have full traceability (everything is versioned in Git repositories), repeatability (environments are stamped out and thus absolutely identical), and we gain full automation, all the way to production. We call this Continuous Deployment. It's really Continuous Delivery, but with human gates removed. It might take us a while to get there, because the whole organisation has to be aligned and very comfortable that we can move safely at speed. This means that we have to demonstrate that this is a safer way of working, with less risk than we have today.

This confidence comes from everyone seeing higher quality product (fewer defects) being delivered faster at less cost, a fully automated CI/CD process with an audit trail and short mean time to recovery.

It's all to do with the size of each change. In a traditional waterfall project (where there is no distinction between deployment and release), releases are infrequent and contain a lot of changes. There is a lot of risk associated with those changes, so process and documentation is employed to help protect us. The volume of changes mean that problems are hard to diagnose, especially as they are not all current.

Compare this with continuously deploying many small changes (maybe even 20 per day). Each change is small, meaning the risk of deploying it is small. It's fresh in everyone's heads. If there is a problem it's easy to find as the locality of the change is small. Rolling back is easy, but more importantly, fixing forward is also easy and quick, and therefore preferable. The

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

ability to roll a system back to a known state remains important in the case that a fix forward does not resolve an outage, or the resolution/diagnosis requires considerable effort and therefore prioritisation. Overall, we can reduce the Mean Time to Recovery (MTTR) to just a few minutes.

Tiny changes delivered continuously means we also get faster feedback and the opportunity to fix bugs early when they are easier (and much cheaper) to fix.

Alongside this, we separate deployments from releases. Deployments happen all the time, are fully automated, and are a technical decision. Releases happen less frequently, are a business decision and just consist of turning a feature on. These features are built gradually behind feature flags. We can release features incrementally, to individual users, groups or percentages of a cohort. This reduces risk and allows us to quickly turn the feature off if there are problems. Ultimately this allows us to validate features quickly and safely, in production, with controlled audiences.

We are describing a north star here and wouldn't expect that we can get there quickly or all at once. The Agency is going on a journey towards this, but we want to make sure we are pointing in the right direction from the outset. We want to use the Medicines Information Portal as a low-risk example of how this can work for the

 of the agency and its customers.

Microservices

So how do microservices help us with this journey?

Traditionally, monoliths were built using waterfall methodologies. They were designed up front. Built in their entirety. Tested and deployed as a whole. All in sequential phases.

By breaking the monolith into much smaller microservices, we can continuously deliver each of them independently of the others. Each microservice is much more focused and easier to understand and evolve. Each can be completely replaced, if necessary, much more easily, and with significantly less disruption. Importantly, each can be built using technologies that are much more suited to the job they are doing (for example one microservice might be dealing with data that is better suited to a document database than a SQL database).

With microservices, applications are decomposed into easily upgradeable, distributed components that are simpler to reason about and easier to scale independently. Messages

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

are passed around the system over standard protocols, making the system more observable.

Cloud

The proposed design fully embraces the cloud strategy and is almost completely agnostic to which cloud provider is used. It does not mandate any proprietary software or platforms.

Cloud providers are much better at managing infrastructure, services and datastores than organisations such as the agency. We want to leverage managed services as much as possible, especially for storing state. We can provision quickly, significantly reduce costs and reliably store data with effectively infinite scale. For example, CosmosDB has an SLA of 99.999% uptime (5-nines), which is the equivalent of about 5 minutes downtime per year. Data is replicated across zones and/or regions. This is much more reliable (and much cheaper) than we could ever hope to achieve if we were managing our own databases.

The architecture described below fully resides in Microsoft Azure. However, it would be identical if it were to be built in AWS or GCP (we would just replace AKS with EKS or GKE, CosmosDB with DynamoDB or Firestore, Blob Storage with S3 or Cloud Storage). This allows us potentially to move to another provider with little additional effort if needed. It also allows us to adopt a multi-cloud strategy, meaning we don't place all our eggs in one basket.

Docker, Kubernetes and Cloud Native

True portability, though, comes from using industry standard Docker containers for packaging and deploying our workloads (web apps, microservices, legacy apps etc). They can be deployed to any orchestrator (Kubernetes, DC/OS, Mesos, Docker Swarm), but 2 years ago Kubernetes won the battle and is now used almost universally at organisations across the world. All the major cloud providers have managed Kubernetes offerings (AWS EKS, GCP GKE and Azure AKS), which are free to use (you only pay for the VMs that form the nodes in the cluster).

All the cloud providers also have serverless offerings (AWS Lambda, GCP Cloud Functions and Azure functions). This is an even higher level of abstraction where you deploy just functions and pay per second of runtime. Serverless is becoming a viable alternative, but whilst it is cheap for intermittent workloads, it can be expensive for continuous workloads and we want to have our microservices always running. Serverless cold start times are still too long, adversely affecting user experience and possibly making Serverless an unsuitable option for this design.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

Inside the Docker containers are Cloud Native workloads. There is an excellent article on the New Stack which describes what it means to be Cloud Native, but basically cloud native services are designed to be lightweight, run in containers, are 12-factor, and can be built, managed and deployed using DevOps practices.

This design incorporates stateless cloud native microservices, running in Docker containers, orchestrated by Kubernetes provided by Azure AKS, consuming data stored in managed services such as Azure CosmosDB, Azure Blob Storage, and Azure Search, monitored by Azure Monitor, and reporting access activity to Azure Activity Log.

Service Mesh

In order to operate a secure, fast and reliable microservices ecosystem, we need a service mesh to route and manage traffic between services, and secure that traffic with authentication (both end-user and service-to-service) and authorization (role-based access control). Additionally, service meshes allow us to observe how services are behaving and can help us test how our services behave when faults are injected.

We are choosing to use Istio as this is the most mature and feature complete implementation available. One to watch is Linkerd 2.0 which is extremely well implemented, but some important features are not yet available.

All this may seem like a lot of overhead to manage a simple service. However, this is a platform that can be used to run all of the Agency's services (including those yet to be built) and will become increasingly important as functions are carved out of Sentinel prior to its decommissioning.

By starting out with a service mesh, we get to leverage the mesh's implementations of all the cross-cutting concerns that services have, allowing the services to purely concentrate on business functionality, making them simpler and much easier and cheaper to build.

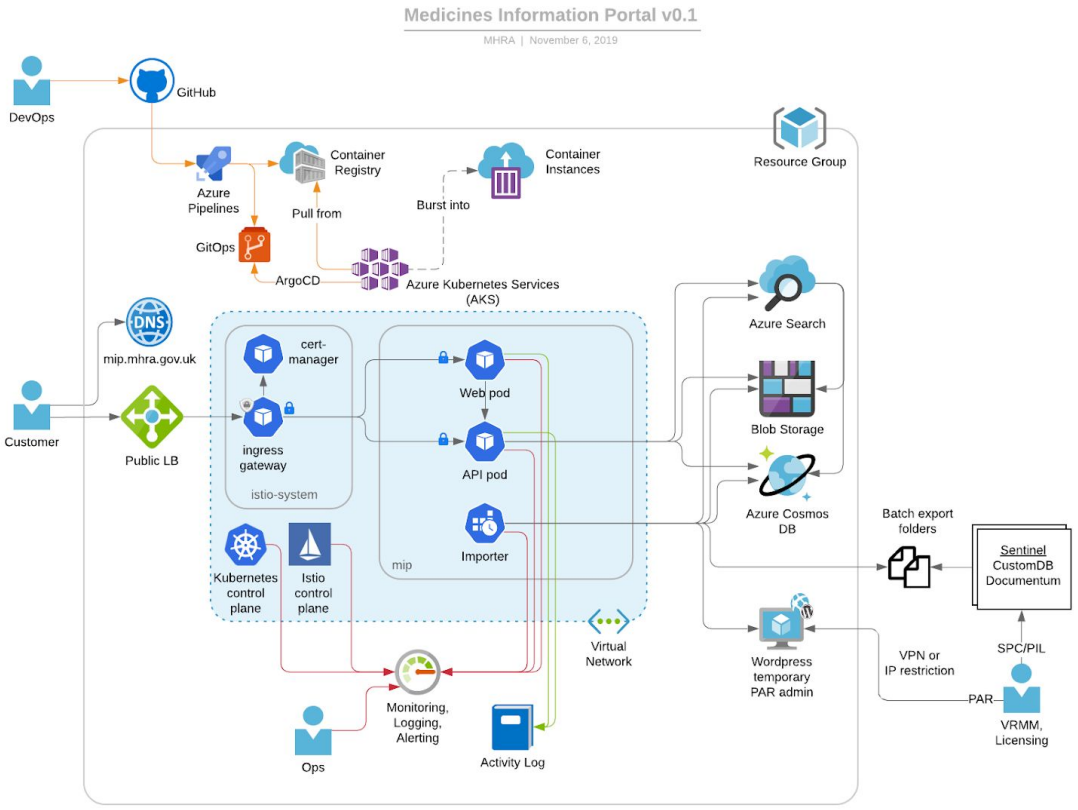
DevOps

DevOps is a set of practices, a mindset, a culture. Not a set of tools or a distinct team. Modern cross-functional teams have engineers that are DevOps capable allowing the team to be fully responsible for deploying and running their products all the way to production.

The design described below also shows how a DevOps focused cross-functional team can use a pipeline to fully automate the path to production.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

High level design



The diagram above shows how we can leverage Azure's Kubernetes Service (AKS) to host an API pod for the medicines microservice, a Web pod for the (initially static) website, and an importer "cronjob" pod which periodically imports data from Sentinel's batch export process and from the internal Wordpress instance. Although each pod is only shown once, in reality there are multiple instances of each, for reliability (across zones) and scalability (new instances can be created and destroyed automatically and in just a few seconds).

The shaded blue area is the Kubernetes (K8s) cluster, which also hosts an Istio ingress gateway. This is the entrypoint for all requests incoming from the Internet. It terminates SSL using certificates automatically renewed (from Let's Encrypt) by the certificate manager pod. The gateway allows us to do host- and path-based routing to services within the cluster, and we can apply Istio policies, e.g. rate limiting, here.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

Documents would be stored in Azure Blob Storage, named after a digest of their contents. This is called content-based addressing and means that two identical documents resolve to the same name giving us deduplication for free. An updated document would resolve to a new name, giving us version history for free. We can keep historical documents for ever as storage is very cheap and effectively infinite.

Metadata for the medicines (including lists of associated documents) would be stored as a JSON object in Azure CosmosDB. Azure Search would index both the documents (pdf and Blob Storage are both supported) and the metadata (CosmosDB is supported). Azure search also has AI plugins for features such as phrase detection. This would give us a rich, relevant search experience similar to that provided by commercially available search engines such as Google or Bing).

The API pod would contain a lightweight custom HTTP server, written in a language like Go or Rust. It would be stateless so it can scale out with ease. It aggregates data from all three sources and presents a documented API that conforms to the Open API 3.0 specification. It would be read-only to start with, and would be trivial to build.

The Web pod would start life just serving static HTML, CSS and JS files for client-side rendering. Later it can be evolved for server-side rendering to give users a faster time to interactivity and better SEO. It could be written in JavaScript on NodeJS and would be trivial to build.

The importer pod would be scheduled to run weekly by the K8s scheduler. It would also be written in a language like Go or Rust and would read the files exported from Sentinel, validate and process them before storing data and PDFs to Cosmos and Blob Storage respectively, triggering Azure Search to update its indexes. This would be a temporary component that would be deprecated as the management function is migrated here from Sentinel.

We may well be able to carve off an MHRA subdomain and manage this in Azure too. The root “A” record would point to the load balancer. We would need to ask GDS to set up “301” redirects for existing URLs.

Security considerations

Our aim is to apply the architecture described above in a way that is secure by default. This means that workloads that are deployed to the platform will automatically operate with the most secure options enabled.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

TLS

This includes ingress traffic using TLS only (with version negotiation up to and including TLS 1.3) and service-to-service communication within the mesh using mutual TLS (mTLS, client and server certificates, with rotation). HTTP ingress on port 80 will be permanently redirected to HTTPS on port 443.

All egress traffic will be denied unless a relevant Istio ServiceEntry (and associated Istio VirtualService) has been defined. When allowed, egress traffic will be HTTPS only.

Secrets

A Bitnami Sealed Secrets controller running in the cluster will decrypt Kubernetes secrets that are stored encrypted in the Github repository alongside the other application manifests. The private key for the controller will be stored in Azure Key Vault. There will be one key for non-production instances and separate keys for each production instance.

Using encrypted secrets works well with the "Everything as code" principle that we discussed above. It means that passwords and keys for upstream services can be versioned and controlled safely alongside everything else, knowing that they can only be read on the target cluster once they have been decrypted by the controller.

Authentication

This MVP phase does not require origin authentication (e.g. a JWT bearer token that the user has obtained from an identity provider).

However, service-to-service traffic (including from the ingress controller) will have transport authentication provided by enforcing mTLS and leveraging Istio Secure Naming.

Authorisation

Kubernetes Role-based Access Control (RBAC) will be used to ensure services have the relevant access to the Kubernetes API.

There is no need in this phase to use Istio RBAC as authentication is not currently required in order to search for and view the SPC/PIL/PAR data.

Data security

There is no Personally Identifiable Information (PII) of any kind involved in this phase of the solution design. All data is freely available to anyone and therefore does not need encryption

in transit or at rest (although in-transit protection will be provided by enforcing TLS throughout).

Vulnerabilities and attack vectors

Software versions will be kept up to date in order to ensure that security fixes for Common Vulnerabilities and Exposures (CVE) are consumed as soon as possible after becoming available. This includes Istio, Kubernetes and Linux versions (both within containers and on nodes). Nodes in the cluster will be refreshed regularly, and new nodes will be created from up to date images. Containers will be built from up to date base images as they pass through the CI/CD pipeline. We will need to evolve policies for ensuring that this happens and is auditable (again the "everything as code" principle helps us with audit and traceability).

As we build web applications, services and APIs that run on the cluster, we will need to ensure that (at least) the OWASP top ten security risks are considered. Many do not apply to this design (e.g. SQL injection attacks because there is no SQL or direct database access, broken authentication as there is no authentication, broken access control as authorisation is not required, XML external entities as there is no XML), whilst others do (e.g. security misconfigurations, cross-site scripting, insufficient logging etc). Where relevant, we will use appropriate techniques (e.g. by using React with additional mitigations for XSS, and CSRF tokens in forms, exhaustive access logging, etc) to mitigate these attack vectors.

Perceived risks, and their mitigation

This section is a work in progress. Please feel free to add/amend as needed.

Upskilling/Hiring

There are quite a lot of skills encapsulated in this design: DevOps, Kubernetes, Istio, Go/Rust, Azure/Cloud architectures, distributed applications, continuous delivery etc. A lot is expected of modern cross-functional teams.

There will probably be a perceived risk that because we don't have those skills in house, it would be difficult to create and own such an architecture. One of Red Badger's core value propositions is "Empower and Embed" and we will help the agency build their own capability as quickly as we can. This starts with leading by example, and we will help build teams and scale capability.

All the skills are mainstream in the industry and so the pool of talent is wider. That said it is hard to recruit great people and it is currently an employee's market. We need to adopt these technologies in-house so that we can attract great talent. Employees today have a

choice of employer and will choose organisations that are already using these principles to build microservices on top of industry standard open source technologies.

Cost

It may be perceived that this approach is too expensive. However, embracing the cloud (if done properly) can significantly reduce the total cost of ownership of a solution. For example, the above architecture could cost less than a few hundred pounds a month (probably not even that much because enterprise agreements are in place). By using cloud and open-source software no licensing costs are incurred.

There may be a perception that the amount of work involved is greater than with a “low code” platform. However, our feeling is that while these appear to be attractive they end up costing more in licensing alone. They also get you 80% of the way there very cheaply, but the other 20% is much harder and very expensive. There is also a significant cost associated with not being in the mainstream. People are harder to recruit and cost more.

Enterprise software also “locks” the consumer in and thus the cost of change as the consumer behavior changes or there are other external regulatory or market changes becomes very high. Thus building microservices that are quick and cheap to build and easy to evolve or discard is ultimately a more cost efficient strategy.

Draft
Please feel free to add comments, or ask questions:
stuart.harris@red-badger.com/stuart.harris@mhra.gov.uk

Changelog

Date	Version	Status	Author	Change
25/10/2019	0.1	Draft	Stuart Harris	First draft written
5/11/2019	0.1a	Draft	Stuart Harris	Added section on security considerations
6/11/2019	0.1b	Draft	Stuart Harris	Update to show temporary Wordpress site as being used only by admin staff for uploading PAR documents