ADR 001: Use the microservice architecture style with containerization

Farmacy Food is a start-up company and does not have a sizeable team of experienced developers available. The overarching architecture style for the Farmacy Food system should be simple, easy to create, maintain and **evolve**. Finding developers that can create and evolve the system, as well as tools and frameworks that support the system should not require heaps of money. In other words, Farmacy Food is not in a position to be an *early adopter*, and should hence adopt an established architecture style that supports evolution.

Decision

We will use the microservice architecture style for backend services in the Farmacy Food system. Microservices will be containerized using Docker.

Rationale

These days, backend services for software systems similar to the Farmacy Food system commonly use a microservice architecture. In itself, this is not a reason for adopting the microservice style. However, positive consequences of the wide use of microservices contribute to this decision: - large number of tools and frameworks for microservice development, as well as for monitoring, logging and distributed tracing. - increasing number of engineers familiar with the style and respective dev tools and frameworks. - support for containerized microservices available in public cloud providers.

Nevertheless, there are important tradeoffs involved in using the microservice style.

Rejected alternatives and rationale: - *Serverless*: it would be perfectly feasible to create many of the Farmacy Food services using FaaS (AWS Lambda for example). However, at the end of the day, some of the functionality (e.g., batch processing for data synchronization) requires runtime elements that don't fit the execution constraints of serverless platforms. Therefore, in practice, the Farmacy Food team of developers would end up having to learn, create, maintain, and govern both serverless functions as well as traditional containerized applications and services. This scenario would increase the burden and the tech skill requirements on the team. - *Phased approach*: we have ourselves been part of a handful of startups in the past. It's typically "shut up and type" mode, with strict time and budget constraints. Top priority is to have a running product out of the door. In the Farmacy Food scenario, which is "green field" to the most extent, we don't see the need to invest time in designing and implementing stage architectures (e.g., modular monolith ==> service-based or macroservices ==> microservice). We have good reasons to believe using the microservice style from the getgo is perfectly feasible time-wise, as well as technically and financially.

- *Service-based architecture*: we believe this style, as described in the book by our Kata host, would work just as well for the Farmacy Food system. It has the nice benefit of separating the system in coarse-grained units called domain services, which are simpler to design and reduce the burden of packaging, deploying and overseeing to a reasonably small number of units. We chose microservice over service-based for two main reasons: - Microservices are more suitable to containerization and hence cloud-deployment. - We try to mimic the service-based style maintainability benefit at the development artifacts level via the hexagonal architecture. For example, a single git project that corresponds to a hexagonal architecture can yield a few microservices (i.e., docker images, our deployment units). We then handle the increased burden of managing more fine-grained *runtime* units via **extensive automation**.

Status

Proposed.

Consequences

- A nice benefit of the microservice style is flexibility, which enables evolution. This benefit is two-fold:
  - *At deploy- and run-time*: flexibility to employ service-specific security, data replication, persistence, and monitoring configurations; flexibility to use different scalability mechanisms for each service.
  - *At development-time*: flexibility to use different technologies, patterns, frameworks, libraries, data sources, and other resources in each service.

- The challenge with increased flexibility is the need to establish some design guidelines and design standards for microservice development. Although it's a startup, Farmacy Food should try to define and enforce (via governance mechanisms) a basic reference architecture (with room for evolution). "Governance" is an abominable word, especially for startups. But what we're suggesting can be as simple as:
  - Have a clear representation of the reference architecture (we have a good start in our hex architecture, and deployment architecture).
  - Make sure all devs know about and understand the reference architecture.
  - Create a space (e.g., a Slack channel) for occasionally discussing changes and evolution of the reference architecture. Later on, when Farmacy Food becomes a larger enterprise, it could be done in an "architecture guild".

- Farmacy Food needs a team of developers familiar with microservice development, microservice patterns, as well as with tools and frameworks, such as:

  - Spring Boot
  - Docker
  - Kubernetes
  - Spring Cloud AWS
- The team will also need some basic knowledge around tools for log consolidation, monitoring, and distributed tracing for microservices.