Microservice Workshop Position

Len Bass

A microservice architectural style, like any other architectural style, is designed to support particular quality attributes at the possible expense of others. Consequently, to understand a microservice architecture, we look at which quality attributes are supported by the style, which can be supported without major tradeoffs and which are not well supported. Since a microservice architecture is closely connected to the process for using it, we also must examine the related process activities. I now discuss the qualities in three categories – supported, neutral, shoe-horned.

Supported qualities

I identify three qualities supported by a microservice architectural style – deployability, reliability, and testability.

Deployability

The primary motivation for a microservice architectural style is to support continuous deployment which in turn is intended to reduce the time to market. In quality terms, a microservice architectural style supports deployability.

To understand how a microservice architectural style supports the reduction in time to market, it is necessary to understand some reasons for delay in deployment. In particular, release planning introduces the necessity for a great deal of coordination among various development teams. Teams must coordinate to determine items such as technology choices, versions of libraries, and languages used. Furthermore, a new version of a system cannot be released until all of the teams involved in aspects of the system have completed their work. Thus, time to market for a new version of a system is dictated by the slowest team involved in elements of the system.

A microservice architectural style sharply reduces the amount of coordination required among development teams (the process element). Each service is a) produced by a small team and, hence, is small and b) is packaged as a deployable unit communicating with other services only through messages. Thus, technology, version, and language choices can all be made independently by each team and there is no necessity for coordination.

Reliability

Assuming the system being developed is going to be deployed to a cloud environment, handling the inevitable failure of components is an important portion of the style. The microservice architectural style is designed to discover failure quickly. It does this by requiring each instance of a service to register and renew its registration frequently (90 seconds is mentioned as a desirable time frame). A client must add a discovery step periodically prior to invoking a service. Thus, if an instance of the service fails, a client will discover the failure and not attempt to use the failed instance. Caching on the client side and asynchronous verification of the cache will reduce the performance penalty for this repeated discovery.

Other aspects of reliability are treated as with other styles and are neither supported nor inhibited by the choice of a microservice architectural style.

Testability

Testability is supported in two fashions.

1. Small components are easier to test than large one. Coverage tests are easier to achieve with small services. Microservices are developed by small development teams and, consequently, are small and easier to test.
2. Live testing is simplified by microservices. Live testing, such as performed by the Netflix Simian Army, can involve killing services in production, introducing latency into communication, or performing various janitor services. The impact, for example, of killing a small service is going to be less on a production system than the impact of killing a large service. Hence, inducing failure and understanding the consequences of this will be simplified by small services.

Neutral Qualities

A microservice architectural style is neutral with respect to performance and security.

Performance

The use of strictly message passing for communication among the services adds performance overhead and this is a negative. Some chains of services can be 70 deep (e.g. Linkedin) and so there is potentially a performance penalty from using a microservice architectural style. This penalty can be ameliorated by choice of placement of instances within a data center such as requiring some services to reside on the same rack or packaging them into the same virtual machine or container.

On the other hand, for many applications the bulk of latency time is taken up by non message based activities such as algorithms or user display preparation. The performance impact of utilization of amicroservice architectural style will depend on the characteristics of the application being developed.

Security

Perimeter security is not affected by the choice of application architecture. Security of the communications between components can be accomplished by by a combination of tokens (such as Kerberos) and efficient encryption algorithms. In other words, the impact of security is a performance impact. As with our discussion of performance, the choice of when to authenticate and encrypt is driven by the characteristics of the application.

Negative Qualities

Modifiability is both positively and negatively impacted by the utilization of a microservice architectural style.

- Positive. Microservices have by the design of the architecture very low coupling between them. Dependencies on items such as versions of libraries or technology choices are

explicitly removed by having small services where each development team makes their own choices and packages their services into a deployable unit.

- Negative. There are three types of negative impacts.
    - Many small services being independently deployed means that dealing with different simultaneously executing versions can cause problems. Techniques exist to manage multiple simultaneous versions both across instances of a single service and across different services involved in a particular function but they can get complicated.
    - The behaviour of an interface can change from one version to another. The mitigation of this risk is process oriented – do not allow interface behaviour to change. If the behaviour changes, create a new interface and maintain the old one. This leads to a proliferation of interfaces that itself is difficult to manage.
    - Supporting features will involve many services. Keeping track of which services is involved in which feature introduces process overhead.

Summary

The choice of which architectural style to use is driven by the business context and the trade offs involved in each possible style. The case of a microservice architectural style is no different. If time to market and reliability are the highest priority business goals then it is a good choice. If other qualities are more important then other styles may be better.