WHITE PAPER

Scope Versus Size: a Pragmatic Approach to Microservice Architecture

By Asanka Abeysinghe
VP, Solutions Architecture, WSO2

1. Introduction

Most enterprises today have created a set of services, incorporated some legacy applications and databases, and on top of that created applications like web apps, mobile apps, and consumer-based applications. While this might be a typical scenario, enterprises often struggle with the resultant complications; therefore, today most organizations are in the process of shifting to a clean service-oriented architecture (SOA).

Architects have a greater challenge now with the concept of microservices becoming increasingly popular within SOA. But it's not a totally new concept. Even a few years ago, some enterprises had started to adopt this concept by spinning up a new environment for each service they wrote instead of sharing existing ones, thus enabling easy maintenance as there were no dependencies between services. Likewise, different development teams were using similar concepts as well.

2. Microservices and the Evolution of Technology

In a very realistic sense, microservices is somewhat of a link between the technologies of two generations. The concept has been explained in different ways. Tight coupling was prevalent in the pre-SOA, or the Platform 1.0 era, which resulted in complications. The progression to Platform 2.0 saw more loose coupling because there were larger services; it included messaging, object orientation, the emergence of traditional SOA, and eventdriven architecture (EDA). The concept of distributing computing moved to the next level in what's referred to as Platform 3.0, which essentially contains the good features that came from Platform 2.0 and the incorporation of next-generation middleware. With the evolution in technology, microservice architecture (MSA) offers complete decoupling thus ensuring agility of delivery and flexibility of deployment.

It's not a completely alien concept though; it essentially pulls all the SOA best practices and then links them with modern application delivery and tooling, such as Docker and Kubernetes, and technology like Puppet or Chef to carry out automation.

That aside, a common misconception among architects about microservices is that it's about the size. However, when they design services they should instead consider how they would scope out their services and eventually

make it a proper set of microservices.

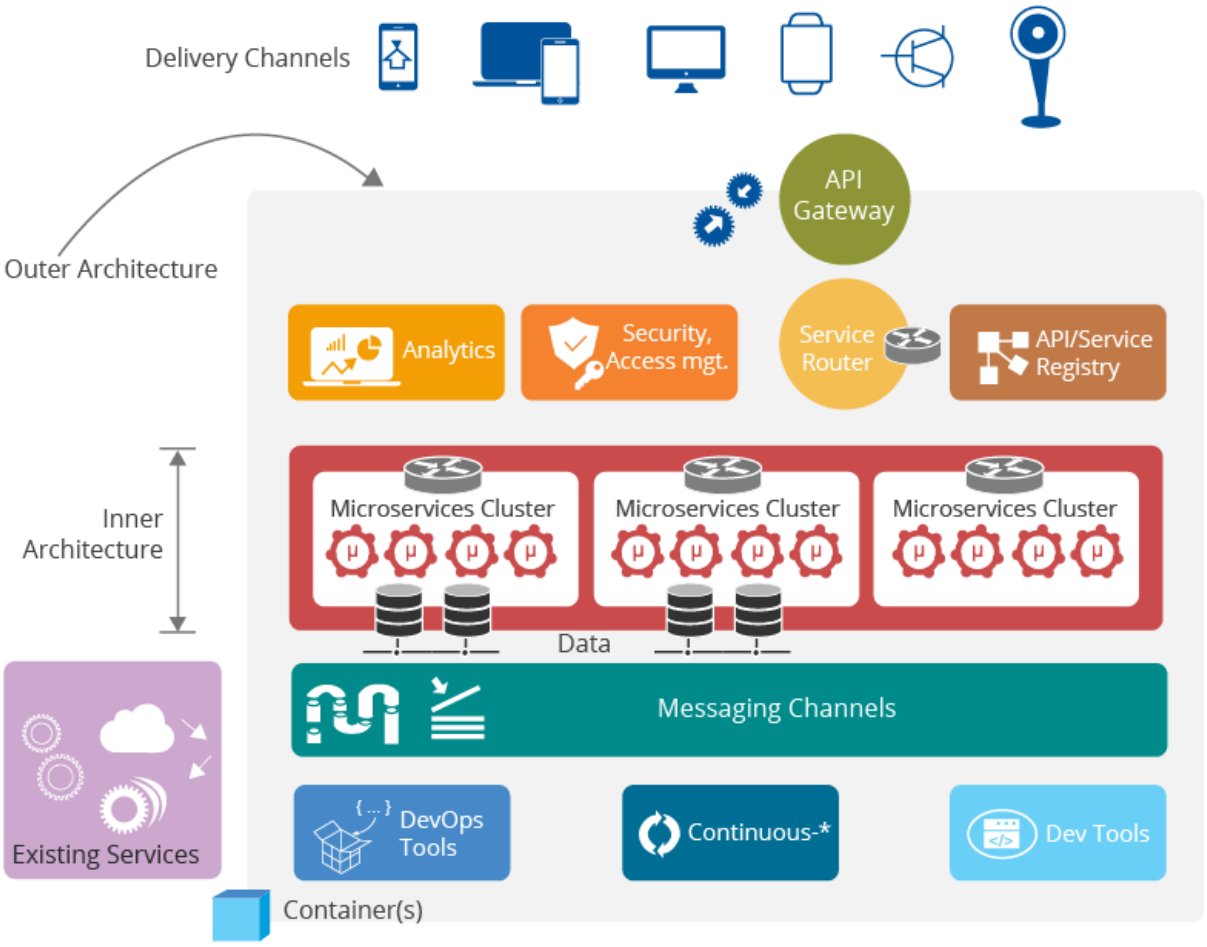

Figure 01

Figure 1 depicts a typical MSA reference architecture. While most architects are keen on the inner architecture where you would write the microservices and deploy them, there's an outer architecture too that would need to be considered. This involves the way in which services get invoked, how the services are governed, how the services communicate with each other, and the rest of the deployment would come beneath these.

The inner architecture refers to microservices and the runtime for hosting microservices, i.e. the services component; the outer architecture is about additional middleware runtime(s) that extend the microservices into an enterprise architecture pattern.

3. Microservice Architecture - Key Fundamentals

MSA has many features that focus on various aspects; however, the following (as explained in a blog by James Lewis and Martin Fowler)[1] could be identified as a key set of features that MSA should contain.

Componentization as services - this basically refers to dividing large services into small sets of services and theoretically they should communicate with each other independently without going through a common messaging layer.

Organize around business capabilities - over time the concept of building systems across business units has changed to meet today's requirements; now organizations operate as pods with each business unit using their own technical capabilities to build applications to provide functionality to their own consumers.

Smart endpoints and dumb pipes - this concept somewhat leans towards having a pointto-point connection; however, given the potential complications that could arise with connecting services to each other, a messaging layer can be used to address the same requirements.

Decentralized governance - this refers to the need to breakaway from having a common set of practices and policies to solve varied and rapidly changing business requirements, and to meet specific business stakeholder demands.

Decentralized data management - this concept will use different types of data stores to store different data instead of having a centralized one. However, a challenge from the business side is the need to have common data sets as well as models even if they individually store different data stores.

Infrastructure automation - this is a key feature in microservice architecture and refers to how you build solutions from dev, then take it to the testing phase and eventually into production, and then how you could quickly spin up new instances based on runtime features.

Design for failure - this is not a new concept and you can write code using this crash only concept. Yet, from a platform perspective, you would need some key features such as devops-friendliness to spin up new instances in case one instance goes down. Moreover, data and analytics is important as well to be able to monitor, predict, and take necessary actions.

Evolutionary design - this concept focuses on avoidance of building a heavy solution and rather using a concept like an MVP (most viable product) and improving this in an iterative manner.

4. The Role of Enterprise Middleware

With the above key MSA features in place, the next step would be to ascertain what's required in terms of middleware.

With regard to **componentization as a service**, the middleware should basically provide a high performance functionality as well as support various types of service standards that comply with RESTful service implementation standards like JAX-RS. It also needs to be lean and use minimum resources in your infrastructure thus providing the necessary lightweight and high-performance runtime to dynamically deploy the services.

For **organizing around business capabilities**, the middleware should include a platform for business units to build services and expose those capabilities. While these small groups would have the freedom to build applications, the middleware should also provide some sort of governance as well as some standards to write applications on top of the platform.

Figure 2 illustrates a sample reference architecture that can be referred to as a platform for digital transformation. The diagram shows different middleware capabilities, such as API, integration, analytics, security, governance, etc. that have been provided as a platform and how the various business units utilize these. There's also the concept of multi-tenancy that's heavily utilized; each business unit will get a tenant and utilize it or some organizations may opt for a federated deployment where the same platform can be duplicated and run with different business units as well. However, what's important is that the platform will contain a set of standards that will be inherited by all and used in their respective deployments.

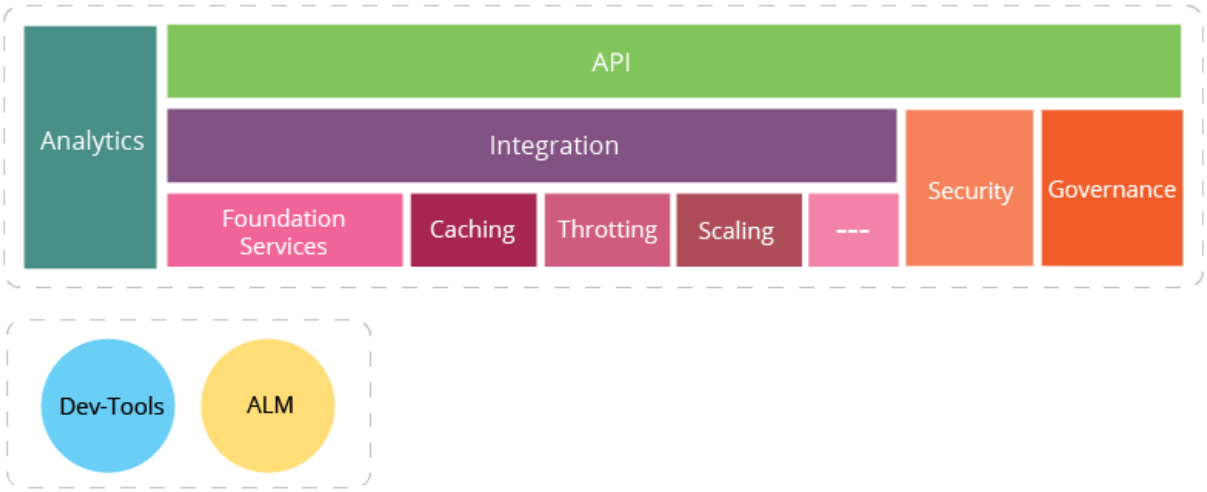

Figure 02

In order to provide a **product** to the end-user, an enterprise should use different types of technical capabilities, hence the middleware stack should offer a complete stack with endto-end capabilities as explained in Figure 3.

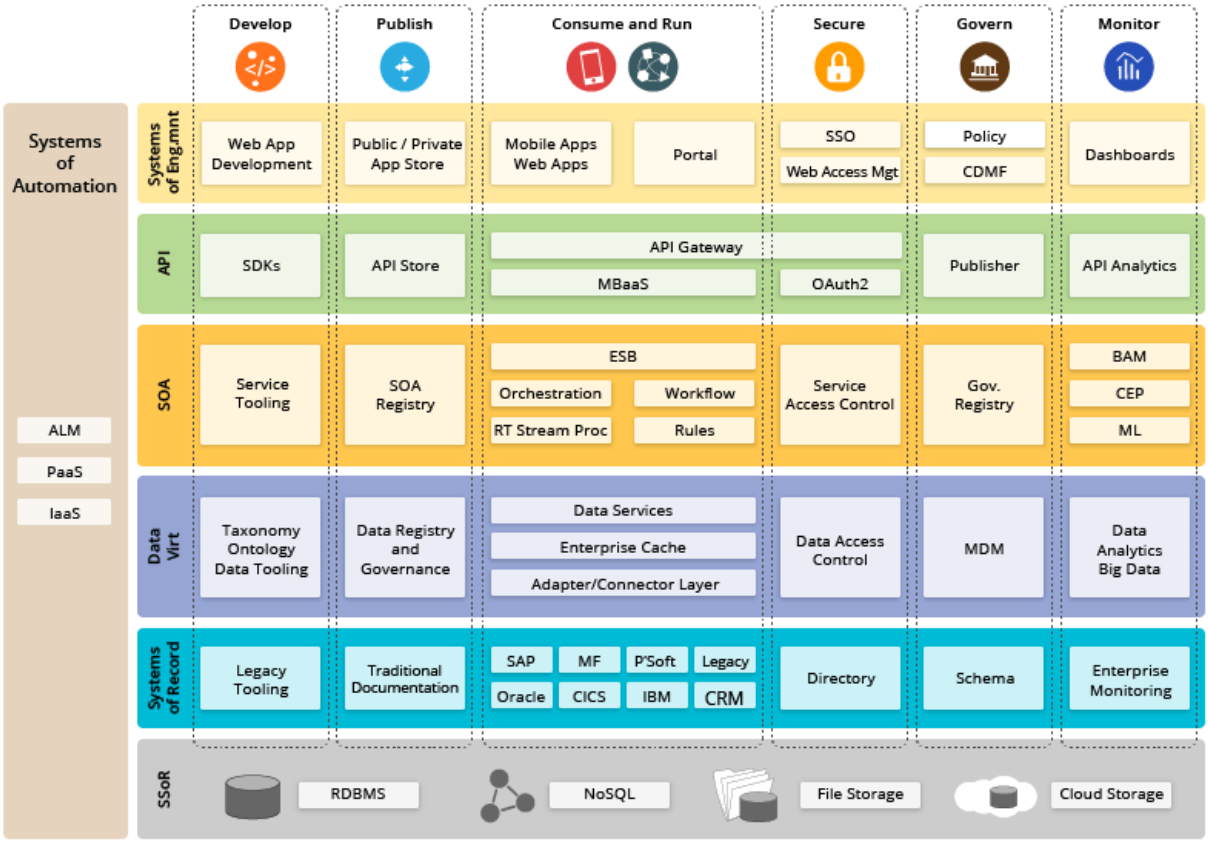

Figure 03

The enterprise would require different types of capabilities to carry out various tasks. From the source system of record to system of records, you would see all the storages that you store and data access capabilities built on top of that; you may also need data virtualization as there might be some non-compliances with the standards that you need as a data model from your APIs; lastly, you would have all the SOA-related components and the API would be the consumable unit of the service. On top of the services, you would have all the APIs and all the delivery channels and applications you write for your consumers. Thus, a business unit can create an application and then provide it if they have most of these capabilities in their middleware platform as it will enable them to easily write a service, build applications, and provide these to consumers. The devops and infrastructure-related components too would come across the different architecture layers that are shown in the diagram.

In the case of implementing **smart endpoints and dumb pipes**, the middleware should support the bus and broker architecture. Enterprise integration patterns would be useful too as apart from the microservices, you would need to connect your legacy systems, i.e. connecting the old world with the new world. Middleware capabilities such as an enterprise service bus (ESB) and a message broker still play a vital role to accommodate connectivity in MSA.

The concepts of API-driven and polyglot programming in Platform 3.0 will help to resolve issues related with **decentralizing governance**. Platform 3 is a revolutionary way to do development that many companies are trying to adjust to; in other words, it's a platform for IT growth and innovation built on mobile devices, cloud services, social technologies, big data analytics, and the internet of things. When an enterprise exposes everything as an API they also provide freedom to app developers to write applications. Even though these developers will use business functionalities via the API there is no risk as the APIs are properly secured and the usage of the APIs are governed as well.

On the other hand, to **decentralize data management**, the middleware should provide or support EDA; for instance, in case there are updates to data or a different set of data needs to be refreshed, an event-driven nature enables eventing that will push these events and make sure all sources that subscribed to the event will be updated and propagated to their system. That way data-related functional capabilities, such as transaction management and data security, can be implemented with the support of EDA.

In terms of **infrastructure automation**, the middleware platform would need to be devopsfriendly and support functionalities like automation testing, continuous integration, and containerization. It would also need to support devops automation like how devops scripts are carried out and how middleware supports scripting and distributed deployment. In terms of containerization and virtualization, you would spin up an instance and then communicate with each and every node, such as hashing and throttling, to ensure proper distributed deployment. The deployment should be able to support higher availability and scalability as well. Lean, independant runtime is key in case dependencies will not allow you to quickly spin up a new instance; what's required is the ability to quickly boot up the runtime and make it a part of the existing cluster that's running already.

As discussed earlier, big data analytics is important to ensure **design for failure**. To meet this requirement, the middleware platform should have a comprehensive data analytics solution and should be devops-friendly. Moreover, it should support an iterative architecture and implementation to maintain **evolutionary design**. To do this, the middleware should be pluggable, i.e. you should be able to plug component without affecting the runtime, and it has to be extensible where you can change the middleware based on your needs as well as those of your domain.

5. The WSO2 Advantage

To meet all of these requirements, an enterprise would need a complete middleware stack that can support different types of platforms (Figure 4).

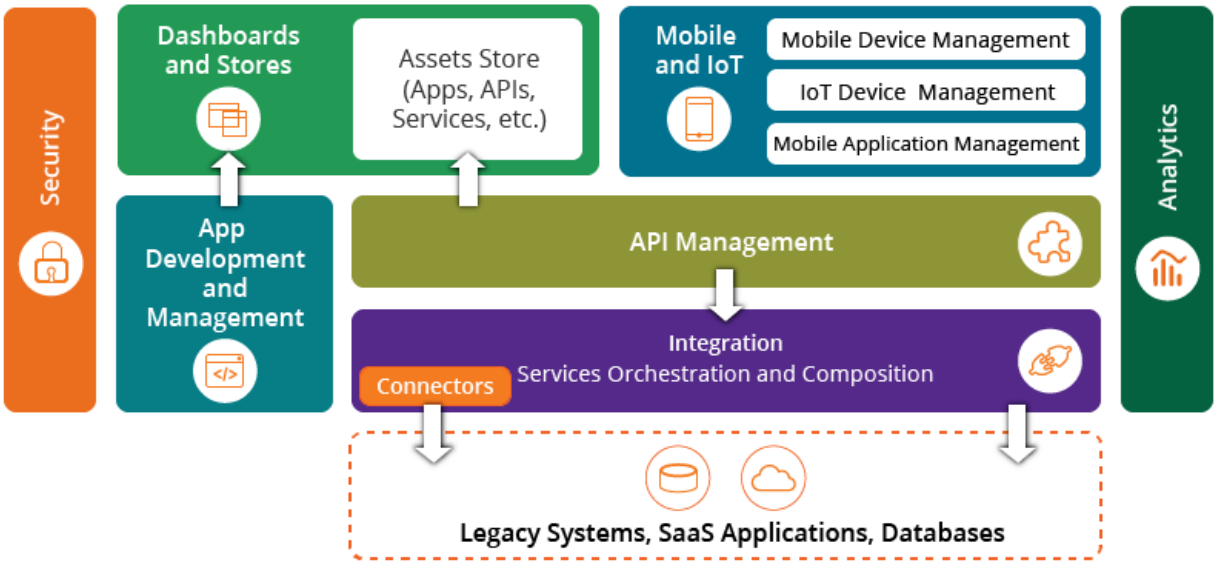

Figure 04

The middleware platform should be able to carry out all integration requirements and then service writing, API management, analytics, and security. It should also be able to support mobile and IoT needs as well as dashboard stores, and app development and management. Lastly, it should have the capability to integrate with various systems like legacy systems and Cloud-based systems as well.

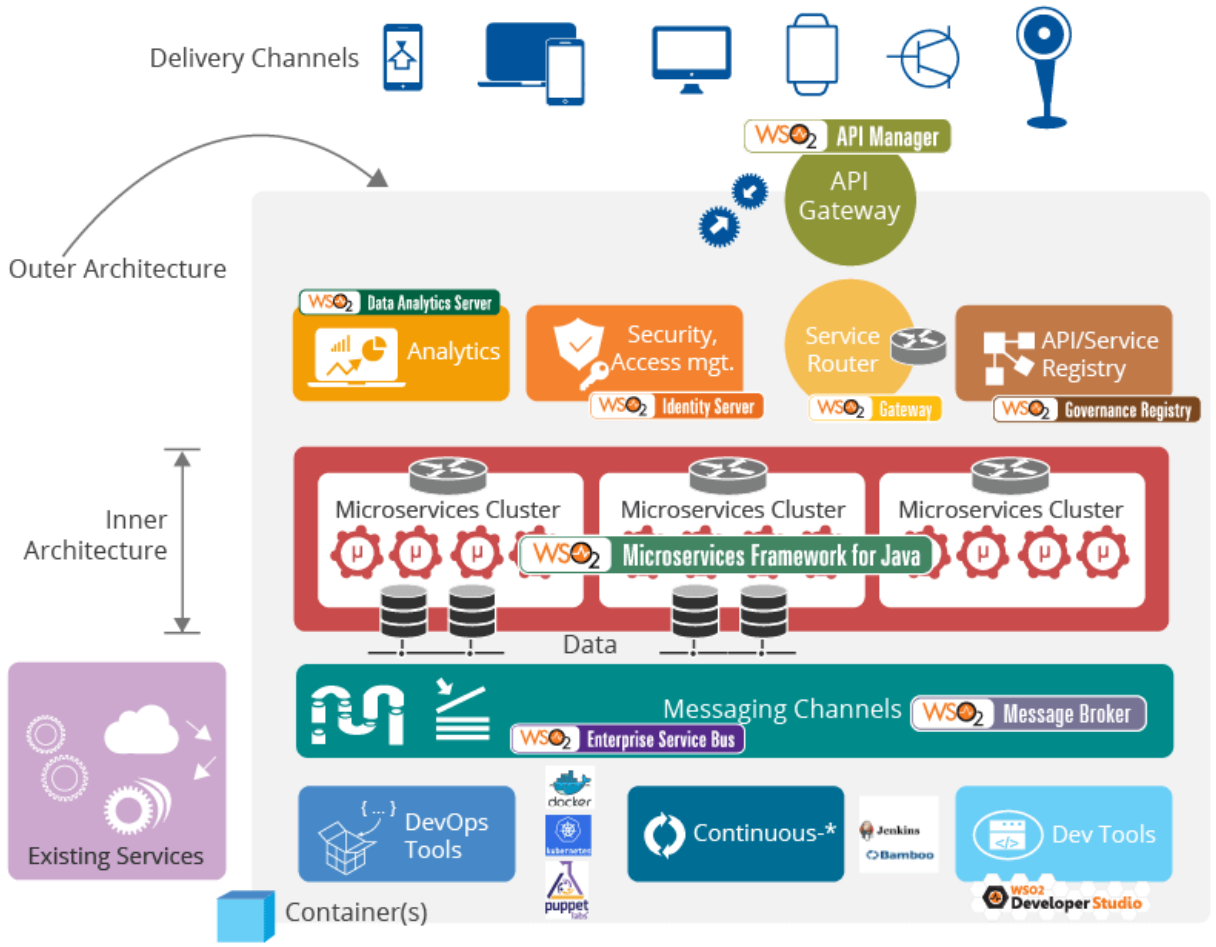

Figure 05

Figure 5 maps WSO2's middleware capabilities with the MSA reference architecture explained earlier (refer to Figure 1). The above diagram illustrates how WSO2 has created some microservices as well as microservice-like architectures. For instance, WSO2 has a lightweight, fast runtime and annotation-based programming model, WSO2

Microservices Framework for Java, offering the best option to create microservices in Java with container-based deployment in mind. WSO2 Message Broker is used to do messaging and WSO2 Governance Registry is used for governance services of your APIs. To expose all these services as a service gateway, you can use WSO2 API Manager to take care of the API management component. For security and identity requirements you can use WSO2 Identity Server and capturing of data analytics can be carried out with WSO2 Data Analytics Server. In order to connect the old world with the new world, WSO2 also has a traditional enterprise service bus, WSO2 Enterprise Service Bus, which also supports technologies that come from Docker and Kubernetes as a container as a service and then Puppet for automation as well. It's further supported by WSO2 Gateway, an ultra high performance, lightweight and configuration driven message gateway based on standard gateway patterns.

6. Summary

To create a platform for innovation as well as rapid application development, an enterprise should build an API-driven architecture that's more consumer-driven. It can then utilize the infrastructure and dynamically add to the platform based on runtime events; these are not statically configured and everything works based on the runtime where the events will flow and then based on those events you can make decisions.

While an MSA is the way forward, enterprises would need to strike a balance by incorporating the good things in an existing architecture. It's important to not lose existing applications as well as some key SOA principles. Middleware capabilities like integration engines as well as tools that are being used, and distributed deployment with functional containers should be included in the architecture built with microservices.

For more details watch our on-demand webinar on A Pragmatic Approach to Microservice Architecture: The Role of Middleware.

---

[1] http://martinfowler.com/articles/microservices.html