Designing Web-Scale Workloads with Microservices

Executive Summary

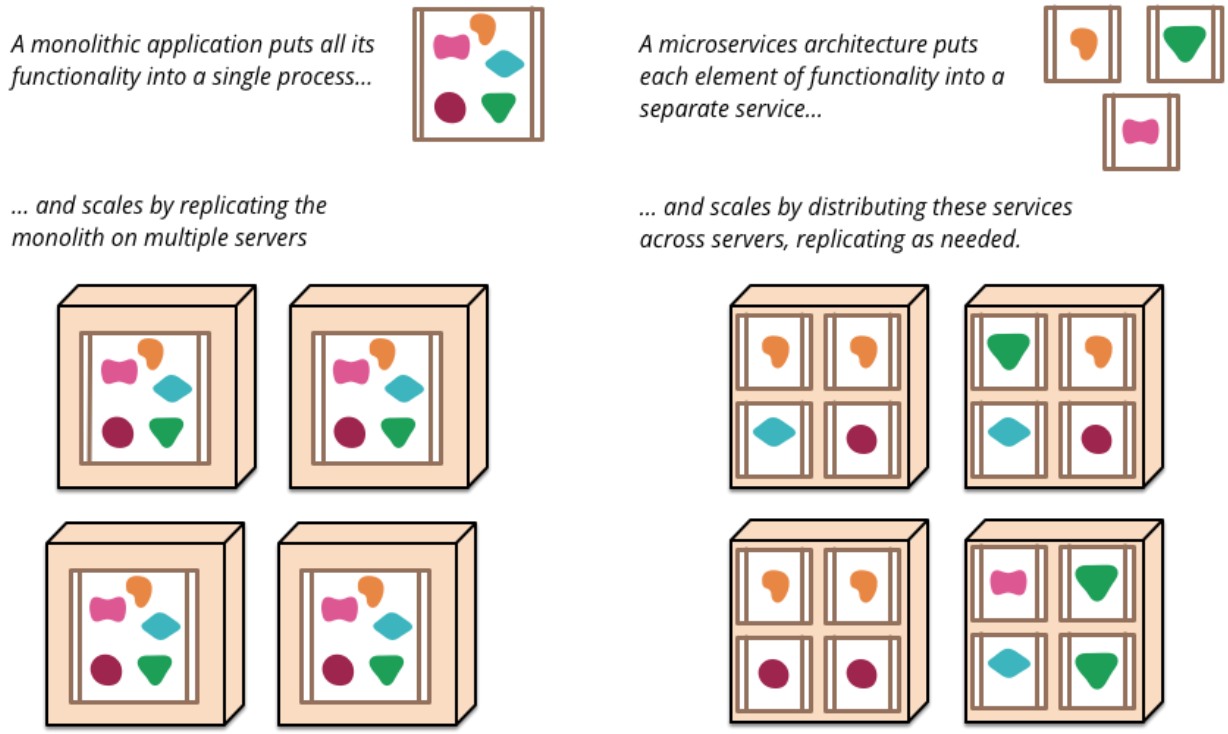
Containers and microservices are redefining the software development lifecycle. Developers are empowered to choose best of the breed languages, frameworks, and runtimes to develop software. DevOps teams are dealing with new packaging and deployment mechanisms. Container orchestration tools such as Docker Swarm, Kubernetes, Apache Mesos are changing the way applications are deployed and managed.

In this new environment, APIs become more important for developers to integrate with internal and external data sources. Through its API-first approach, DreamFactory enables developers in building data-driven, web-scale workloads. Through its support for a variety of relational, NoSQL, and cloud databases, developers can stay focused in building the core business logic instead of dealing with the plumbing required to integrate databases.

This report takes a closer look at the microservices landscape and the role of DreamFactory in empowering developers.

What are Microservices?

Martin Fowler and James Lewis from ThoughtWorks published an article on microservices in March 2014, which gained prominence among web-scale startups and enterprises. According to the authors, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.



Source: Martin Fowler

The microservices architecture promotes developing and deploying applications composed of independent, autonomous, modular, self-contained units. This approach is fundamentally different from the way traditional, monolithic applications are designed, developed, deployed and managed.

The component architecture represented a shift away from how applications were previously developed using dynamic-link libraries, among others.

Distributed computing has been constantly evolving in the past two decades. During the mid-90s, the industry evaluated component technology based on Corba, DCOM and J2EE. A component was regarded as a reusable unit of code with immutable interfaces that could be shared among disparate applications.

However, the communication protocol used by each component technology was proprietary – RMI for Java, IIOB for Corba and RPC for DCOM. This made interoperability and integration of applications built on different platforms using different languages a complex task.

Evolution of Microservices

With the acceptance of XML and HTTP as standard protocols for cross-platform communication, service-oriented architecture (SOA) attempted to define a set of standards for interoperability.

Based on XML and Simple Object Access Protocol (SOAP), the initial standards for web services interoperability were handed over to a committee called Oasis.

Suppliers like IBM, Tibco, Microsoft and Oracle started to ship enterprise application integration products based on SOA principles.

While these were gaining traction among the enterprises, young Web 2.0 companies started to adopt representational state transfer (REST) as their preferred protocol for distributed computing.

With JavaScript gaining ground, JavaScript Object Notation (JSON) and REST quickly became the de facto standards for the web.

Key Attributes of Microservices

Microservices are fine-grained units of execution. They are designed to do one thing very well. Each microservice has exactly one well-known entry point. While this may sound like an attribute of a component, the difference is in the way they are packaged.

Microservices are not just code modules or libraries – they contain everything from the operating system, platform, framework, runtime and dependencies, packaged as one unit of execution.

Each microservice is an independent, autonomous process with no dependency on other microservices. It doesn't even know or acknowledge the existence of other microservices.

Microservices communicate with each other through language and platform-agnostic application programming interfaces (APIs). These APIs are typically exposed as REST endpoints, WebHooks, or can be invoked via lightweight messaging protocols such as RabbitMQ. They are loosely coupled with each other to avoid synchronous and blocking-calls whenever possible.

Why Should Developers Embrace Microservices?

With microservices, developers and operators can develop and deploy self-healing applications. Since each microservice is autonomous and independent, it is easy to monitor and replace a faulty service without impacting any other.

By moving to microservices, organisations can invest in reusable building blocks that are composable. Unlike monolithic applications, microservice-based applications can be selectively scaled out.

Instead of launching multiple instances of the application server, it is possible to scale-out a specific microservice on-demand. When the load shifts to other parts of the application, an earlier microservice will be scaled-in while scaling-out a different service. This delivers better value from the underlying infrastructure as the need to provision new virtual machines shifts to provisioning new microservice instances on existing virtual machines.

Developers and administrators will be able to opt for best-of-breed technologies that work best with a specific microservice. They will be able to mix and match a variety of operating systems, languages, frameworks, runtimes, databases and monitoring tools.

Finally, by moving to microservices, organisations can invest in reusable building blocks that are composable. Each microservice acts like a Lego block that can be plugged into an application stack. By investing in a set of core microservices, organisations can assemble them to build applications catering to a variety of use cases.

What’s in it for Businesses?

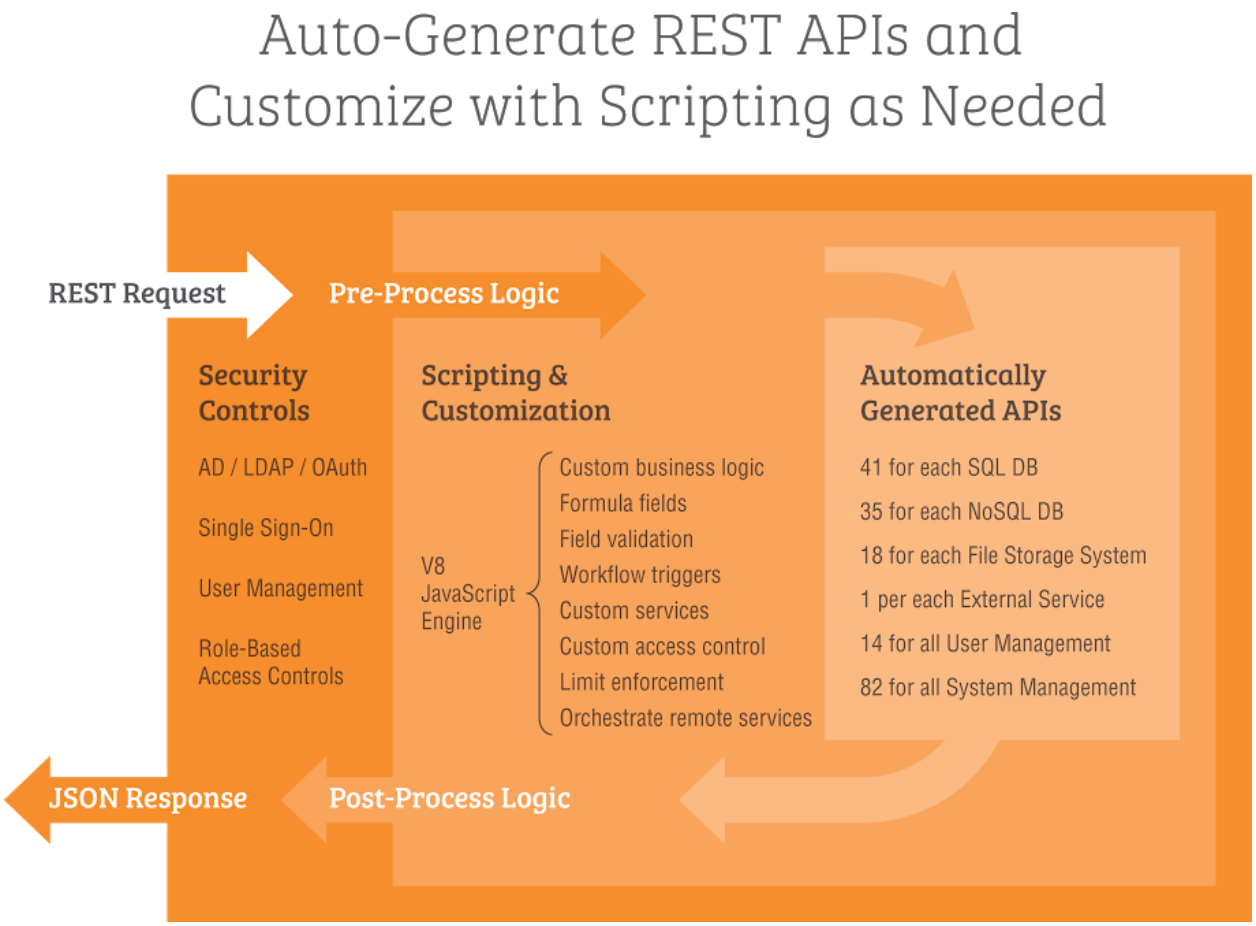
While microservices make developers focus on one component or a module of the software, it also benefits the businesses by delivering agility. Organizations embracing microservices will build highly reusable software that can be utilized across a variety of projects. The modularity of microservices enables them in engaging the best teams to build the components.

Faster “go to market” strategy, high reusability, and modularity are some of the advantages of adopting microservices.

Building Microservices with DreamFactory

DreamFactory is an open source REST API middleware platform that provides RESTful services for building mobile, web, and IoT applications.

It enables developers to connect to any data source and instantly get a full palette of secure, reliable, and customizable REST APIs for their projects. It is a runtime environment that can be easily integrated with web-scale applications. Developers can declare database endpoints that are invoked at runtime.



DreamFactory is based on a modular architecture with clean separation of stateful and stateless layers. The core building blocks of DreamFactory are based on popular open source languages, frameworks, and databases. Each of these layers can be packaged into independent deployment units based on virtual machines or containers. The core runtime built on PHP can be easily scaled out to handle additional load. Redis and MongoDB layers are stateful that can be configured for redundancy and higher availability.

Features such as server-side scripting and API-driven administration makes DreamFactory ideal for microservices. Developers can dynamically update the endpoints without changing the code. Server-side scripting makes it possible to define business logic specific to data sources. Basic validation and error handling can be embedded within DreamFactory without hardwiring it in the application logic.

The DreamFactory stack can be composed of a set of containers that can be deployed on container management platforms such as Swarm, Kubernetes, and Apache Mesos. These container definitions can be integrated with contemporary microservices based on Docker.

Next Steps

DreamFactory is available as a single Docker container image that can be pulled from Docker Hub. Microservices developers can take advantage of the modular architecture of DreamFactory by deploying and scaling individual containers. The blog post on scaling DreamFactory with Docker explains how to achieve scalability in production environments. The Docker Compose file available on Github can be used as a reference architecture to get started. It can be easily modified for creating the YAML artifacts supported by Kubernetes.

Summary

While system and application requirements continue to evolve, the methodology behind how we solve these problems is often based on older models and patterns. As mentioned before, microservices architecture has its roots in models like COM, CORBA, EJB and SOA, but there are still some rules to live by when creating microservices utilizing current technologies.

Developers can leverage microservices to achieve modularity and scale required by contemporary applications. API-first platforms such as DreamFactory enable developers to separate stateless and stateful service through a declarative mechanism.