KEY ENABLING TECHNOLOGY OVERVIEW

Microservices & Containerization

Description

Microservices Design Pattern

Microservices represent an architectural and development approach that is much different from the practice used to build large, monolithic applications. The more functionality encompassed by these large codesets the more complex the design, development and testing. Although monolithic architectures involving narrowly scoped applications can often be deployed much faster than microservices architectures, at least initially, changes to monolithic systems become increasingly more difficult and time consuming as each change requires extensive regression testing.

On the other hand, the microservices approach consists of a set of loosely coupled, collaborating set of services supporting a narrowly defined scope of business functionality. They are characterized by their independence, where each of the services is developed, tested and deployed separately from each other and run as separate processes. Given their independence, microservices can be developed using different development languages (i.e., polyglot) and persistence components. They interact with other services through well-defined interfaces. Being simpler, and more modular microservices become a catalyst for enhanced productivity, innovation, polyglot coding and data persistence. Microservices typically execute within a container.

Major Vendors:

This is an open source design pattern, not a product. However, Amazon Web Services, Microsoft Azure, Google Cloud Platform, Dockers and other vendors provide enabling platforms and technology that can be used to support a microservices/container architecture.

Case Study:

Migrating E-commerce Architecture from Monolithic COTS to Microservices Platform

Developing a Continuous Integration and Deployment Pipeline

Containerization

Containers are self-sufficient bundles that comprise all the software needed (with the exception of the operating system kernel) to run services in an isolated process and are immutable (unchanging over time). Being self-sufficient means that a container commonly has the core application components along with its dependencies, such as libraries, configuration variables, persistence and release artifacts. Containers came as a solution to the challenge of running applications and system reliably when moved across different computing environments, hence becoming a key tool to enable a DevOps pipeline approach in a sustainable manner.

Application

As the microservices design pattern leverages the philosophy of business domain-driven design (where the granularity of the service itself focuses on a specific business function), the idea of matching software components with business processes and related functions gets closer to realization. "Microservices allow for building software solutions incrementally instead of deploying large, complex projects". Development can be done in small teams, more or less independent of each other. This results in software development teams that are more easily organized around business capabilities and not technologies. Services grow as adaptable components to be used in multiple business contexts and leveraged by more than one business process or function across the enterprise.

As such microservices become an excellent tool to complement or extend the capabilities of existing enterprise software platforms or even support the wholesale replacement of systems within and across business domains. Their fine granularity and inherently bounded context allow for surgical, progressive, rapid, low risk implementation of change (new computing, data augmentation, integration capabilities, etc.) with fast return on investment. They also enable a degree of experimentation by supporting the rapid deployment and redesign of solutions based on responses from employees and/or customers.

How to Get Started

A microservices/containers approach is best suited where the business environment is rapidly changing and developing large, complex, monolithic solutions over months and even years is not an option. With that said, it is quite normal to utilize both approaches, monolithic and microservices, within the same company based on the degree of business change over time, the distributed nature of the software development teams (e.g., on-premise, remote, offshore), the maturity/fit of legacy systems and several other factors.

Without sounding self-serving, the best way to get started is to utilize an outside firm to "kick start" your microservices initiative. Cimphoni's consultants have extensive experience in the design and implementation of distributed platforms along with progressive migrations of monolithic legacy software solutions towards modern, agile architectures (i.e., wholesale system replacement). Cimphoni's consultants have also used microservices to improve the capabilities of legacy systems (i.e., "bolt on" solutions). Developing a small, narrowly scoped system to serve as a proof-of-concept also works to demonstrate the value of microservices as well as educate the development team on the tools, methodologies, design patterns, best practices, etc.

About Cimphoni

Cimphoni is built on the premise that technology, when properly applied and led, can deliver innovative solutions that transform businesses. The Cimphoni team is comprised of technology, operations and business consultants with a thirst for innovation and a passion for leveraging emerging technologies to deliver exceptional, measurable results for our clients. Founded in 2012, Cimphoni serves customers throughout the United States from its headquarters in suburban Milwaukee. More information can be found at cimphoni.com