https://doi.org/10.48009/4_iis_2018_65-68

Issues in Information Systems
Volume 19, Issue 4, pp. 65-68, 2018

PARALLEL POLYLITHIC ARCHITECTURE: MICROSERVICES

James J. Lee, Seattle University, leej@seattleu.edu
Ben B. Kim, Seattle University, bkim@seattleu.edu
Tim Mushen, InfoVera, tim.mushen@gmail.com

ABSTRACT

A virtual machine has been adopted to maximize the utilization ratio of computing resources, limiting costs by reducing the need for physical hardware systems. Containers (very light virtual machines) now deliver data locality with processes in the application level whereas conventional application deployment has used data locality in the server which often causes dependency complexity, as applications in the same host share the system's environment. Polylithic architecture using containers in parallel computing opens up new possibilities in the following methods of application foundations: application as resources, application as allocation, application as replication, and application as customization. Parallel polylithic architecture in microservices architecture also empowers us to treat applications as resources where each microservice can be allocated, replicated, or even replaced.

Keywords: ploylithic, monolithic, microservices, docker

INTRODUCTION

Cloud computing is a leading drive of modern IT infrastructure as it gives scalability in which organizations can allocate resources efficiently. In past years, Ubuntu, one of the Linux operating system distributions, has steadily become the choice of cloud's operating system, comprising over 50% of the market share. This is due to the Unix philosophy by its creators (Gancarz, 2003),

> *"The creators of the Unix operating system started with a radical concept: they assumed that the user of their operating system would be computer literate from the start. The entire Unix philosophy revolves around the idea that the user knows what he or she is doing."*

In Eric Raymond's book (2003), the Art of Unix Programming, he provides 17 design rules. Among those, there are some which apply more to clouds today; rule of modularity, rule of clarity, rule of composition, rule of separation, and rule of simplicity.

Despite Unix Philosophy's benefits to modularity and isolation, the Linux environment creates high complexity among installed applications due to its mix of various technologies and dependencies. For example, applications can be developed using different versions of Python such as Python 2.7 or Python 3.4. This makes production servers tuned to run different versions of Python programs. Most developers today are spending time configuring multiple projects' environments on their local computers.

As virtual machines continue to advance and affordability increases from service providers, organizations have managed the large number of servers with no sweat due to the flexibility and scalability from cloud computing. Now with the introduction of containers, the minimal version of virtual machines, managing software engineering has evolved into the newer era today, microservices - the separation of services and applications with polyglot persistence. We will start with the monolithic architecture in the past then discuss how microservices with containers create a new paradigm, parallel polylithic architecture.

MONOLITHIC ARCHITECTURE

Software engineering is rapidly advanced with such a short history since 1960s. One of only few laws in this field that surprisingly explains the personality of application is Conway's law. Conway's law reveals that any products of software engineering are social products which mimic organizational communication structure. This also explains why monolithic systems became too complex to be maintained as those systems aged. Conway's law (Conway, 1968) states,

> *"Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."*

The traditional ***monolithic architecture pattern*** promises 1) a single codebase that is compiled together, 2) all of the code is shipped/deployed at the same time or 3) a single application or process performing the work for the system. Monolithic applications, however, have become monstrous complexities as they accumulate more features with multiple upgrades. The application becomes too large and complex which makes maintaining the system immensely difficult. Additionally, it is often hard to scale when different modules have conflicting system requirements. Furthering the issue, monolithic applications are known not to adopt new technologies easily.

There are historical innovations in software engineering from the old days programming in procedural languages. With the directions in object-oriented programming (OOP), software engineering had advanced rapidly both in modularity and productivity. The OOP foundation is centered on monolithic application design and deployment in mind. To overcome the homogeneous nature of OOP design, the newer systems integration approach has been adopted with the ubiquity of the Internet, service-oriented architecture (SOA). SOA has opened the communication between different applications through the data level such as XML (extensible markup language). The importance of SOA reveals interoperability between multiple different monolithic applications. Nevertheless, monolithic architecture with OOP arsenal is still dominating the software architecture due to clear advantages: simplicity in development, testing, and deployment.

VIRTUAL MACHINES THEN CONTAINERS

With the "Keep it Simple, Stupid (KISS)" Unix philosophy, Linux naturally supports the modular nature of cloud architecture with virtualization technology that maximizes the resource utilization with the concept of multi-tenant systems in one physical server computer (Gupta et. al., 2010). This creates one of major benefits of using cloud computing, scalability, by creating as many as virtual machines as needed. There are various virtual machines provided by multiple software companies, such as Vagrant, VirtualBox and VMWare. In the cloud computing market, especially Infrastructure as a Service (IaaS), there are big players in virtualized computing like Amazon EC2 (Elastic Compute Cloud), Google Cloud Computing, and even Digitalocean, where you can manage virtual hosts (virtual machines as servers).

Because Linux applications have so many dependencies on resource requirements, it is difficult to manage multiple versions of applications with different dependencies in resource environments. With the advancement of Docker containers, this complexity problem is completely resolved, enforcing the polylithic design principles (Richardson and Smith, 2016; Julian et. al., 2016). Docker is a very lightweight virtual machine, called a container virtualization (Anderson, 2015).

One of the original usages of containers is helping developers' workflows by providing container images and working on projects in the derived containers for the provided images, which of course can be shared among other developers. As it is very light-weighted, containers get high scalability - containers launch in a sub second. One of the initial promises of a container virtualization was enabling a portable resource where code and applications reside in their own settings. With agile service creation for one application from a container image, Docker (one of major container service provider) containers can create production-like environments for each application managed as a service instantly, isolated, but collective ways. There has been a hope that all applications would be deployed in containers which makes the painful deployment with aligning with host's complex dependencies part of the old days

(Julian et. al., 2016). And this hype is not wrong with the emerging microservices architecture, discussed in the following section.
One of initial promises of a container virtualization is enabling a portable resource environment for the specific projects where code and applications reside in its own settings (encapsulation). As a typical development process includes multiple computers; developers' computers (development environments), staging server, and a production server, keeping the unified environments across all the computers involved requires huge effort. A container virtualization resolves these time-consuming issues at once as it provides portable and isolated virtualization from an image which does not cause any conflicts between projects in multiple machines.

This benefit is so valuable that now container virtualization is taking consideration into production servers. Because of containerization, modularity can be achievable in the polylithic way with multiple frameworks in different programming languages. This approach is backed by microservices architecture where each module (a service) can be an application and the modules of a full scale application (aka., monolithic software) can be developed in different framework/programming languages.

DESIGN PATTERNS IN MICROSERVICES

The philosophy of microservices is to split an application into set of smaller, interconnected services (Thones, 2015). A microservice can be a loosely coupled functionality, such as registration management, download management, order management, production management, etc. Therefore, each microservice is a mini-application with its own database which complements polyglot persistence - different kinds of data are best dealt with different databases.

Microservices can be implemented either in a cloud virtual machine or a container. This paper mainly uses the example of the Docker container today as it provides various tools with ease of maintenance in many departments, such as scheduling, scaling, upgrades, health checking, and service discovery.

The container approach like Docker often implements multiple containers at the same time. There are two ways to design communications between containers: a single-host container networking pattern and a multi-host container networking pattern. In a single-host container networking pattern, one host typically has several containers running on it whereas a multiple-host container networking pattern utilizes multiple hosts with several running containers in each host. The important point is how polyglot persistence contributes in this architecture. To maximize the benefits from parallel processing with a distributed system such as Hadoop File System (HDFS), an application should stay where the data is (data locality), which naturally premises polyglot persistence by containers implementation to achieve microservices architecture.

Like parallel computing in data analytics field today, orchestration in container is getting attention as microservice architecture is a double-edged sword: decomposing applications in multiple microservices while adding complexity in managing the overwhelming number of microservices with application growth (Richardson and Smith, 2016). Orchestration comes into this position to manage resources systematically. Managing multitude containers can be a daunting job without orchestration tools today; Kubernetes, Mesos, ECS, Swarm, and Nomad (at this writing).

Instead of discussing all five alternatives, this paper is using Kubernetes as the choice of orchestration to manage microservices because of its popularity. Kubernetes (Hausenblas, 2018) is an open source container orchestration and it is on top of DC/OS (Data Center Operating System, simplified version of Mesos). Kubernetes uses a pod, the basic unit of scheduling. A pod is defined as a tightly coupled set of one or more containers that are always collocated, and they cannot be spread over nodes where a node is worker machine. Kubernetes, especially, provides various ways of communications; intra-pod networking, inter-pod networking, ingress and egress. Through this networking, Kubernetes can support communications between containers in a pod, other pods, routing traffic from external apps, and calling external APIs from pods.

PARRALLEL POLYLITHIC ARCHITECTURE

When OOP first came out, it was a great breakthrough to give data locality with processes in procedural level. Containers (very light virtual machines), now, deliver data locality with processes in application level whereas conventional application deployment has used data locality in server which often causes dependency complexity as applications in the same host share the system's environment.

As we discussed in this paper, polylithic architecture using containers in parallel computing opens up the new possibilities in the following ways of application foundations: application as resources, application as allocation, application as replication, and application as customization. What is more, parallel polylithic architecture in microservices empowers us to treat applications as resources where each microservice can be allocated, replicated, even replaced. The rich libraries of microservices will bring the application architecture into the next level with orchestrating microservices.

REFERENCES

Anderson, C. (2015). Docker, IEEE Software, May/June 2015.

Conway, M. (1968). How Do Committees Invent, Datamation, April, 1968 in Newman, S. (2015). Building Microservices, O'Reilly Media.

Gancarz, M. (2003). Linux and the Unix Philosophy, Digital Press.

Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A., Varghese, G., Voelker, G., and Vahdat, A. (2010). Difference Engine: Harnessing Memory Redundancy in Virtual Machines, Vol. 53, No. 10, Communications of the ACM.

Hausenblas, M. (2018). Container Networking, O'Reilly Media

Julian, S., Shuey, M., and Cook, S. (2016). Containers in Research: Initial Experiences with Lightweight Infrastructure, XSEDE16 Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale, Article No. 25, Miami, USA.

Raymond, E. (2003), The Art of UNIX Programming, Addison-Wesley.

Richardson, C., and Smith, F. (2016). Microservices: From design to deployment, Nginx Inc.

Thones, J. (2015). Microservices, IEEE Software, January/February 2015.