LinuxFoundationX LFS158x  
Introduction to Kubernates

INSTRUCTOR: Chris Pokorni and Neependra Khare

learningedx.org

horizontal line

# 16 January 2022

## Chapter 1. From Monolith to Microservices

* **Monolithic applications** with all components tightly coupled and almost impossible to separate, a nightmare to manage and deploy on **super-expensive hardware**.
* If working for an organization that refers to their main business application "**the black box**”, where nobody knows what happens inside and most logic was never documented.

**Learning Objects**

By the end of this chapter, you should be able to :

* + Explain what a monolith is.
  + Discuss the monolith's challenges in the cloud.
  + Explain the concept of microservices.
  + Discuss microservices advantages in the cloud.
  + Describe the transformation path from a monolith to microservices.

1.1 The Legacy Monolith

* Most enterprises believe that the cloud will be the new home for legacy apps.
* But not all legacy apps are a fit for the cloud, at least not yet.
  + Monolith app = a 100-ton boulder
    - thousands of lines of code
    - written in a single
    - not so modern programming language
    - based on outdated software architecture patterns and principles
    - during upgrades, patches, or migrations of the monolith application downtime is inevitable
  + Microservices = pebbles
    - much easier to handle
    - each service has the host system itself
    - lower compute resource expenses
    - aligned with **Event-driven Architecture** and **Service-Oriented Architecture (SOA)** principles
    - written in a modern programming language
    - with specific hardware when required
    - allowing deployments on inexpensive commodity hardware
    - the greatest benefits of microservices is scalability
    - each microservice can be scaled individually
    - either manually or automated through demand-based autoscaling
    - no downtime and no service disruption to clients

**Refactoring**

Newer, more modern enterprises possess the knowledge and technology to build cloud-native applications that power their business.

* Some have tried to run monoliths as microservices, and as one would expect, it did not work very well
* refactoring poses serious challenges and the enterprise faces the refactoring approach dilemma
* a **"Big-bang" approach** or an **incremental refactoring**
* The "Big-bang" approach focuses all efforts on the refactoring of the monolith, postponing the development and implementation of any new features
* An incremental refactoring approach guarantees that new features are developed and implemented as modern microservices which are able to communicate with the monolith through APIs, without appending them to the monolith's code
* most of its functionality is modernized into microservices
* This incremental approach offers a gradual transition from a legacy monolith to modern microservices architecture and allows for phased migration of application features into the cloud.
* Once an enterprise chose the refactoring path,
  + how to decouple the databases from the application to separate data complexity from application logic
  + how to test the new microservices and their dependencies
* The refactoring phase slowly transforms the monolith into a cloud-native application that takes full advantage of cloud features
  + a legacy monolith application receives a second chance at life

Challenges

* a monolith to microservices is not smooth and without challenges
* Not all monoliths are perfect candidates for refactoring, while some may not even "survive" such a modernization phase.
* . A poorly designed legacy application should be re-designed and re-built from scratch following modern architectural patterns for microservices and even containers.
* Choosing runtimes may be another challenge.
  + deploying many modules on a single physical or virtual server
  + deployments of single modules per server in order to separate their dependencies - not an economical way of resource management
  + no real segregation of libraries and runtimes
  + each server also has an underlying Operating System running with its libraries, thus consuming server resources - at times the OS consumes more resources than the application module itself.
* Ultimately application containers came along, providing encapsulated lightweight runtime environments for application modules.
  + Containers promised consistent software environments for developers, testers, all the way from Development to Production
  + multiple applications deployed on the very same server
  + each running in their own execution environments isolated from one another
  + avoiding conflicts, errors, and failures
  + Other features of containerized application environments are higher server utilization, individual module scalability, flexibility, interoperability and easy integration with automation tools.

Success Stories

* AppDirect - an end-to-end commerce platform provider, started from a complex monolith application and through refactoring was able to retain limited functionality monoliths receiving very few commits, but all new features implemented as containerized microservices.
* box - a cloud storage solutions provider, started from a complex monolith architecture and through refactoring was able to decompose it into microservices.
* Crowdfire - a content management solutions provider, successfully broke down their initial monolith into microservices.
* GolfNow - a technology and services provider, decided to break their monoliths apart into containerized microservices.
* Pinterest - a social media services provider, started the refactoring process by first migrating their monolith API.

Learning Objectives (Review)

* Explain what a monolith is.
* Discuss the monolith's challenges in the cloud.
* Explain the concept of microservices.
* Discuss microservices advantages in the cloud.
* Describe the transformation path from a monolith to microservices.