Scalable, Distributed E-commerce Design

Uchenna Kevin Anyanwu

10/22/2018

Background

Monolithic service architecture have several advantages such as being simple to deploy, have less moving parts in its design, deterministic memory footprint, and good for rapid design. However, when handling a large size of traffic, at any given time, monolitic designs, can have dire consequences to the overall operation of a system. To remedy these shortcomings, system architects choose, whenever possible, the microservice architecture. In this architecutre the monolithic system is split into smaller, independent componts or resources. These systems are flexible, maintainable, fault-tolerant, and scalable. However, these benifits are without flaws, for example, more time will be spent on designing the interfaces between componets, and, if care is not taken, will lead to negitive riple effects across the system.

Aim

This paper intends to design and impliment a scalable, distributed ecommerce backend system using a functional monolith ecommorce system.

Method

First, we choose a high-level programming language that is easy to learn, good for rapid prototyping and industry-leading in software development.

Second, we develop an actor system diagram for two display the interactions between all of the actors of a ecommerce system.

Fourth, we design the interfaces between the actors of the system, this will help in understanding the set of APIs needed to interact with the system. Fifth, we develop an architectural system diagram that encompasses the previous two steps Sixth, we develop an api documentation with accompaning simple test cases.

Seventh, we code CRUD operations to CREATE, READ, UPDATE, and DELETE resources of the system. In addition, we write other APIs listed in the previous step.

Lastely, Test each component in isoloation, stubbing external dependencies, and progressively add add another component into the environment, until we have included all of the components to form the entire system.

Software Requirements

Describe a general architecture for a distributed, scalable system that would serve as a backend for a Describe a general architecture for a distributed, scalable system that would serve as a backend for a hypothetical webstore and shopping cart type application. The requirements would be that individual users would have records of what was purchased and their properties, and be able to return to an existing cart of items if they had a cart that wasn't emptied between sessions by check-out.

Implement in code using a bare-bones version of this shopping backend cart and the purchase history component as a 'Proof of Concept. It must understand that different products have certain fixed properties (name, price), and many specific properties (ie, t-shirt sizes, color of hat, flavor of candy, etc).

The code should be modular enough to easily add new products or integrate with a hypothetical frontend. It should also have the ability for a specified user to return to an existing cart of items, then update the user's purchase history and emptying the cart on check-out.

Design and Developlement Ecommerce Design

Monolithic Ecommerce Design

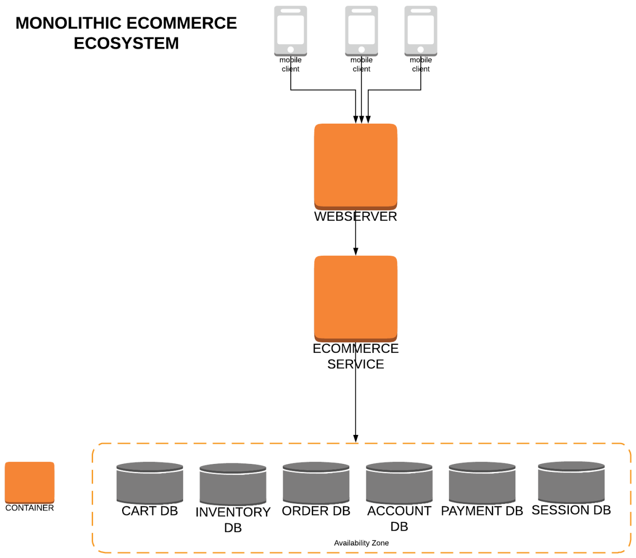
The following is a monolithic design of a generic backend for ecommerce system. In a very simple system the database, a complete eccomerce service, and a webserver would be one one machine. This approach is most favorable for rapid development.

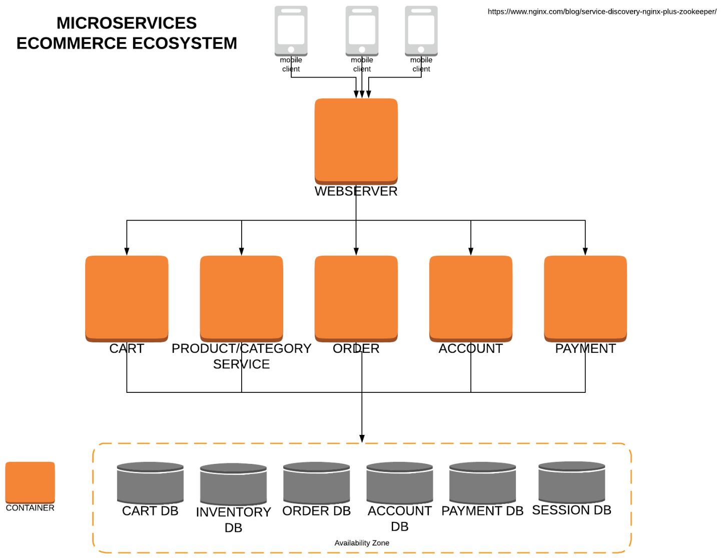
Distributed Ecommerce Design

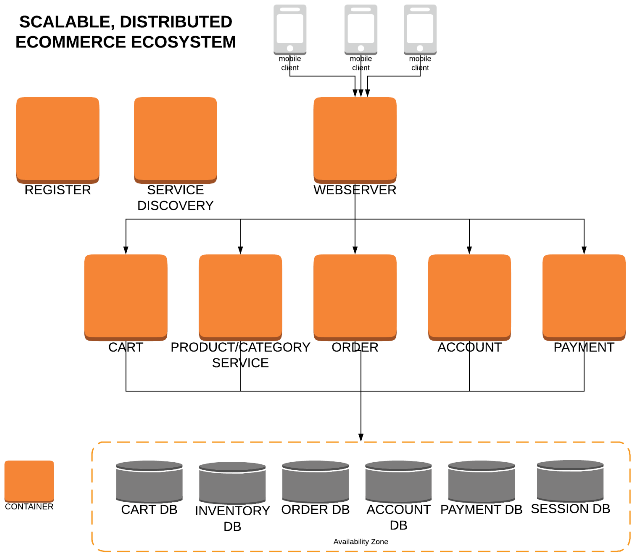
The following is a mircoservices design of a generic backend for ecommerce system.Note that we modeled this design based on a resource-centric outlook on the monolithic design. These services could run at different locations, not just one on the local host, in which case, different IP address would be assigned to each service in the system. This appraoch is most favorable for modularity, maintainability, and scalability, and numerous other reasons.

Scalable, Robust Ecommerce Design

Adding a register, loadbalancer, and service discovery instances to the ecommerce system, we can truley achieve high availability, scalable, and fault-tolerance. The service discovery node, such as Zookeeper node, would track the current network location of a service instance, based on instructions from Registor, which monitors the starting and stopping of containers and updates ZooKeeper about the state changes. Apache Ngnix Plus would be used as a loadbalancer to keep requests distributed in a fair manner amongst all the services in the ecommerce system.







Testing Microservices

Software testing was done on the component-level, function-by-function, System testing was done on the domain-level, client level.

Conclusion

A distributed backend architecuture was achieved within the given time and scope. However, not of enough time was provided to deploy on the cloud, so all testing was done on a local machine. Hypothetically, each service would be deployed on an individual machine and would still work the same. Differences between the two setup would be each service would have a different IP address. In our case each service has different port numbers.

References

[1] D. E. Knuth. *The TEX book.* Addison-Wesley, Reading, Massachusetts, 1984.

[2] L. Lamport. *LATEX : A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1986.

[3] Ken Wessen, Preparing a thesis using LATEX , private communication, 1994.

[4] L. Lamport. Document Production: Visual or Logical, *Notices of the Amer. Maths. Soc.*, Vol. 34, 1987, pp. 621-624.