Cloud Computing - Mini Project Report
Breaking down monoliths
April 2023

Submitted By: Chetan Reddy Bandi PES1UG20CS109
Deeraj KN PES1UG20CS121
Bhanu Shekhar V PES1UG20CS103
Basavaraj N PES1UG20CS102
VI Semester Section_B
PES University

Short Description and Scope of the Project

Short Description:
Debugging a monolithic application: The first objective is to get familiar with reading error stack traces and debugging code. This may involve identifying and fixing bugs in the code, optimizing the application's performance, and improving its functionality.

Converting a monolithic architecture to microservices: The second objective is to convert a monolithic application based on Docker Compose into a microservices-based architecture. This involves breaking down the

monolithic application into smaller, independent services, each responsible for a specific task. This may involve re-architecting the application, rewriting code, and setting up communication between the services.

Scope:
The scope of the project is to take an existing monolithic application based on Docker Compose and convert it into a microservices-based architecture. This may involve identifying the various components of the monolithic application and breaking them down into smaller, independent services. Additionally, the project may involve setting up communication between the different services, implementing fault-tolerance and scalability, and testing the new architecture to ensure it meets the required performance and functionality requirements.

Methodology

Involves these steps

1. Understand the monolith architecture and the error stack trace: Before you start debugging the code, it's important to understand how the monolith application works and what the error stack trace means. This will help you narrow down the scope of the problem and identify the root cause of the issue.

2. Identify the microservices: Once you have a good understanding of the monolith architecture, you can start breaking down the application

into microservices. Identify the different components of the application and decide which ones can be separated into microservices.

3. Define the interfaces: Once you have identified the microservices, you need to define the interfaces between them. This includes the APIs that each microservice will expose and the data that they will share with each other.

4. Refactor the code: With the interfaces defined, you can start refactoring the code to separate the different components into microservices. This may involve creating new code, modifying existing code, or even deleting code that is no longer needed.

5. Deploy the microservices: Once the code has been refactored, you can deploy the microservices using a container orchestration platform like Kubernetes. Make sure that the microservices are properly configured and that they can communicate with each other.

6. Test the microservices: Finally, you need to test the microservices to ensure that they are working correctly. This includes unit testing each microservice and integration testing the entire system to ensure that all the components are working together as expected.

Testing

To test the Project the in local computer the steps involved are:

1. Clone the repository containing the monolith application code.
2. Navigate to the directory containing the code in your terminal.
3. Build the Docker images for all the services using the command docker-compose build.
4. Start the containers for all the services using the command docker-compose up.

5. Access the landing-service at http://localhost:5050/ and test the arithmetic operations to verify that the monolith application is working.
6. Access the other services you have created using their respective endpoints and test their functionality.
7. Modify the index.html file to include links to the newly added services.
8. Stop the containers using Ctrl +C and remove them using the command docker-compose down.
9. Repeat steps 3-8 as necessary to test any changes you make to the code.
10. Once you have verified that the microservices-based application is working as expected, commit the changes and push them to the repository.

Results and Conclusions

Results:

- Conversion of monolith architecture into microservices: This objective involves breaking down the application into smaller, independent components or services. Each service can be developed, tested, and deployed independently, allowing for more flexibility and scalability.

- Familiarity with reading error stack trace: This objective can be achieved by analyzing the error stack trace and identifying the root cause of the issue. It requires understanding the different components and modules involved in the application, and their dependencies.

- Debugging code: Debugging the code involves identifying and fixing the errors that are causing the application to fail. This requires an in-depth understanding of the codebase and the ability to analyze the problem at a granular level.

Conclusions:

- Monolithic architecture can become complex and difficult to manage as the application grows. Breaking down the application into smaller, independent microservices can help in managing the complexity and improving scalability.

- Microservices architecture comes with its own set of challenges, such as managing communication between services, ensuring data consistency, and handling failure scenarios.

- Reading error stack trace and debugging code are essential skills for developers, as they help in identifying and fixing issues in an application.

- The decision to move from a monolithic architecture to microservices architecture should be made based on the specific needs and requirements of the application, and the potential benefits and drawbacks of each approach should be carefully considered.