MICROSERVICE

- Architectural approach to building software systems that decomposes an application into **smaller, independent services** that **work together to provide the overall functionality of the application**
- Each microservice is designed to perform a specific function or task within the application, and **communicates with other microservices through a well-defined API**

MICROSERVICE

- **Independence**: Each microservice is designed to be independent and self-contained, with **its own data storage, processing logic, and user interface.**
- **Decentralization**: Microservices are designed to be **distributed and run independently** of each other, with no single point of failure.
- **Scalability**: Microservices can be **scaled horizontally** by adding more instances of the service, allowing the application to handle more traffic and users.
- **Resilience**: Microservices are designed to be **resilient to failures**, with each service having its own error handling and recovery mechanisms.
- **Flexibility**: Microservices can be written in **different programming languages and technologies**, allowing developers to use the best tool for each job.

MONOLITH VS. MICROSERVICE

- Architecture
  - A monolith application is **a single unit of code that contains all the application components and functionalities**. On the other hand, a microservice application is **divided into multiple smaller, independent services that communicate with each other through APIs**.
- Scalability
  - Monolith applications are **not scalable**, as all components of the application are tightly coupled. **If a particular component receives high traffic, the entire application must be scaled**, even if the other components are not experiencing high traffic. In contrast, microservice applications can be **easily scaled, as each service can be scaled independently, based on its usage and requirements**.

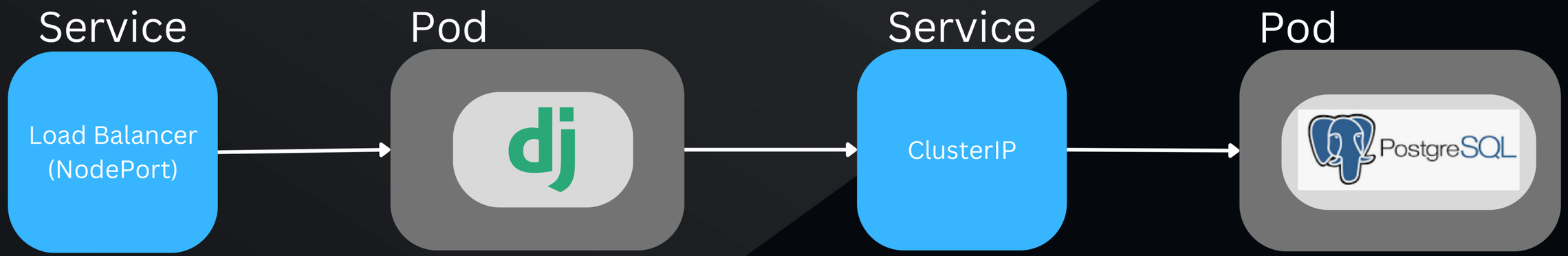
MONOLITH VS. MICROSERVICE

- Deployment
  - In a monolith application, any changes made to one component **require the entire application to be re-deployed**. This can be a time-consuming and challenging task, particularly for large applications. In contrast, microservices allow developers to **deploy changes to individual services independently, without affecting the rest of the application**.
- Maintenance
  - In a monolith application, maintaining the codebase can become challenging as the application grows. It becomes **difficult to understand** how the components of the application are interconnected and to make changes without breaking other parts of the application. In contrast, microservices are **easier to maintain** as each service is independent and can be updated or replaced without affecting the other services.

MONOLITH VS. MICROSERVICE

- Development
  - Developing a monolith application is **relatively simple, as all components are part of a single codebase**. In contrast, developing microservices **requires more effort** as developers need to create multiple services and ensure that they can communicate with each other through APIs.

MICROSERVICE



BUILD A MICROSERVICE

- Create a Dockerfile
- Create a docker-compose to test the code
- Deploy to minikube

BUILD A DJANGO APP

- Steps
  - create a virutal environment
  - create a requirement.txt
  - create a project
  - create a dockerfile
  - create a docker-compose file
  - deploy all to minikube
- For more information
  - 실리콘밸리 엔지니어가 가르치는 파이썬 장고 웹프로그래밍(https://inf.run/ZkPF)