What is a Microservices Architecture?

A SOA variation, microservice architecture is an approach to developing a single application as a suite of small services. Each service:

- Runs its own processes
- Can be written in any programming language and is platform agnostic
- Is independently deployable and replaceable
- May use different storage technologies
- Requires a minimum of centralized management

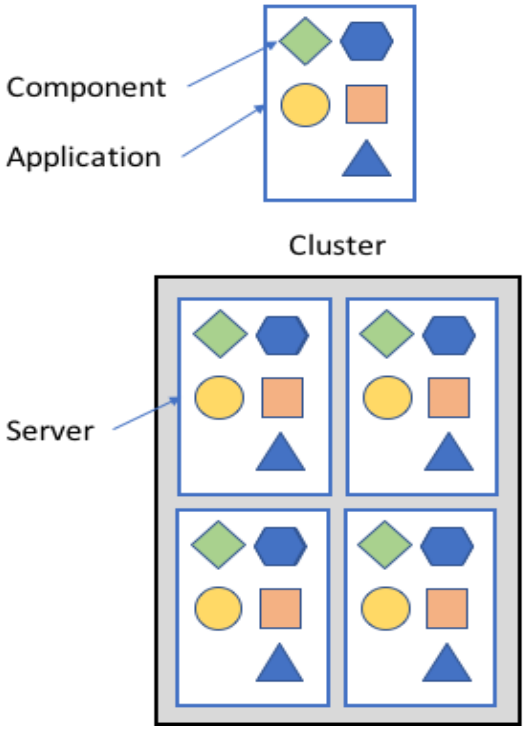
Large companies, such as Amazon, eBay, and Netflix, have already adopted microservice architecture as their main architecture.

Modular Space System Analogy

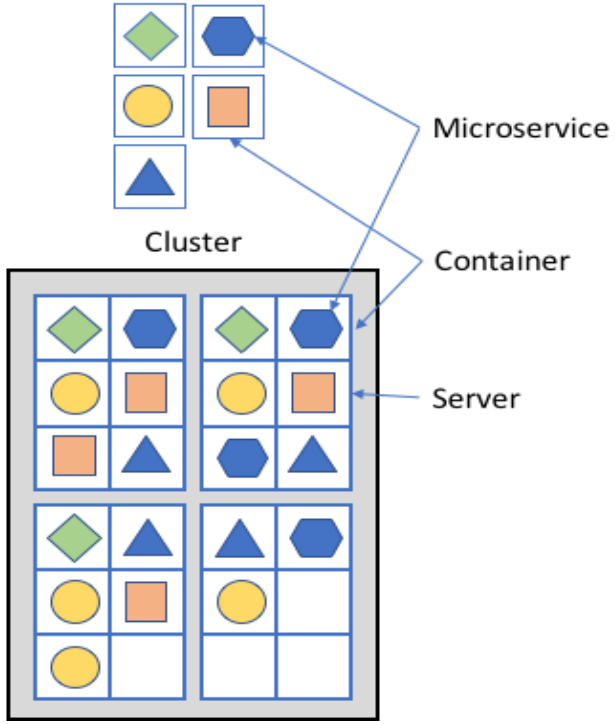
Rather than using a large, complex, single satellite, agencies are migrating to a suite of connected, independent, low-cost, and replaceable small satellites, each with its own function (e.g., communication, imagery, sensors). This decomposition and migration to autonomous modules offers these agencies ease of functionality, ease of replaceability, and ease of modernization with newer modules.

Visual Interpretation of Architecture Patterns

Traditional Monolithic Architecture Pattern



Microservice Architecture Pattern



"Monoliths and microservices are not a simple binary choice.

Both are fuzzy definitions that mean many systems would lie in a blurred boundary area among the two"

-Martin Fowler

Martinfowler.com

Microservices
vs
Web Services

Comparison Chart

Microservices	Web Services
Microservices are a software development architecture that structures an application as a collection of loosely coupled modules.	A web service is an application accessed over a network using a combination of protocols like HTTP, XML, SMTP, or Jabber.
It is an architectural style organized around business capabilities and can be included into a web service.	It's a service offered by an application to another application which can be accessed via the World Wide Web.
It can be implemented in different technologies and deployed independent of each other.	It's a platform that provides the functionality to build and interact with distributed applications by sending XML messages.

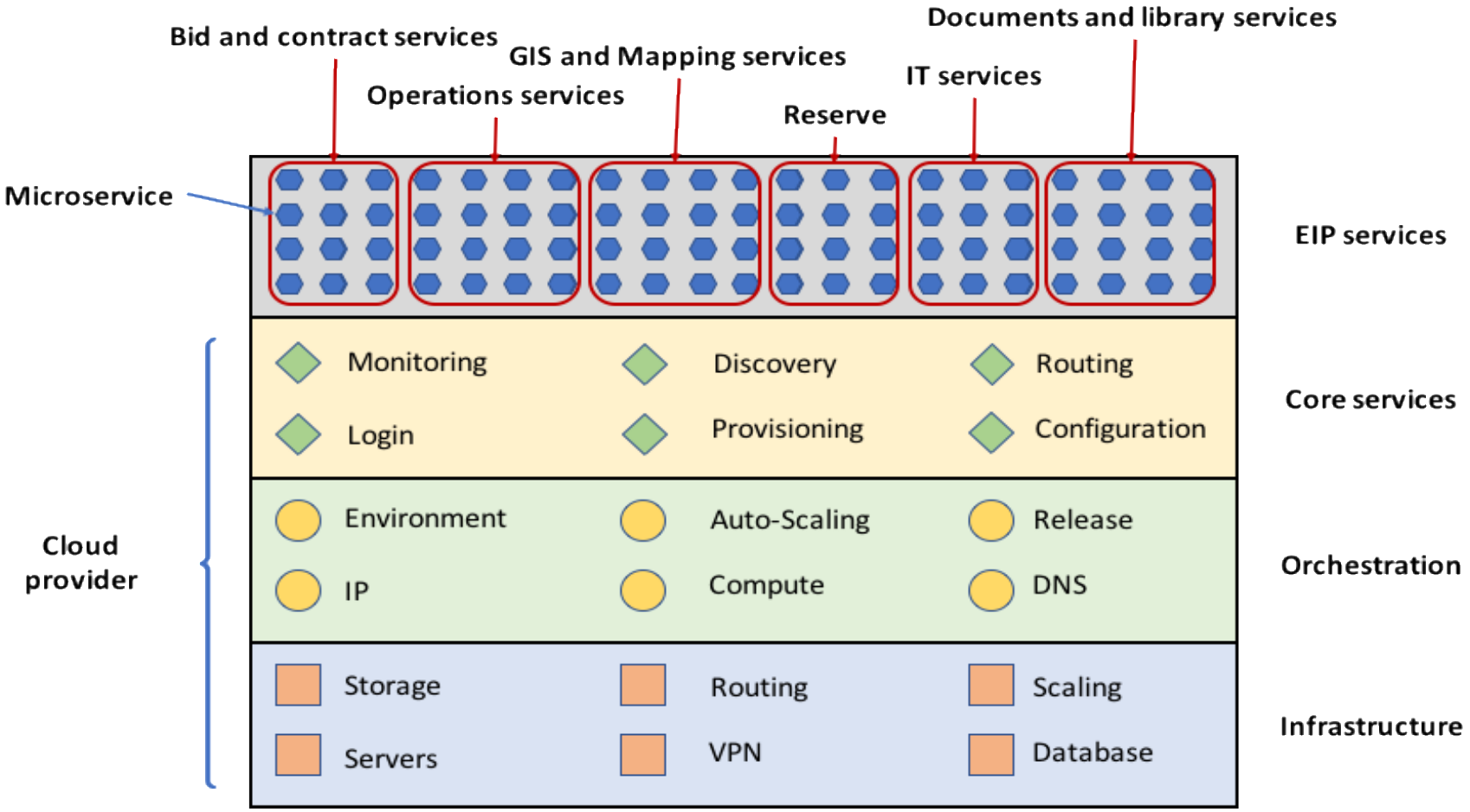
Microservice Architecture Advantages

- Greater efficiencies as they are typically paired with auto-scalers and load balancers
- Easily scalable in three ways:
  - by decomposition into simple functions (microservices)
  - by cloning microservices on demand
  - by partitioning data across a cluster
- Services can be cloned or decommissioned based on user requests in real-time

Auto-scaling - A cloud computing service feature that automatically adds or removes computing resources based on actual demand

Load balancing -The process of distributing workload evenly across computing resources in a cloud computing environment

Microservice Implementation Example of a DOT EIP on Cloud Infrastructure



IT teams shift from actively managing hardware and software to managing service layers in the microservice environment

Tradeoffs in Adopting a Microservice Architecture

Shifting to microservices architecture means

- Reducing code complexity while increasing operational complexity
- Distributing data across multiple processes while ensuring data consistency across processes
- Simplifying individual services while raising the complexity of coordinating many services

Microservice Premium

Microservices impose a cost on productivity that can only be made up for in more complex systems.

Change in EIP Practices, part 1 of 3

When Adopting a Microservice Architecture

- **Design for failure** – applications must be designed and developed to tolerate the failures of other services or applications.
- **Single lifecycle ownership of the product** – "You build it, you run it," is the motto rather than the traditional practice of a software development team handing off a service to an IT operations team.
- **Exercise choice in platforms, language, & libraries** that best suits the microservice functionality and team expertise rather than a single language or platform for the enterprise.

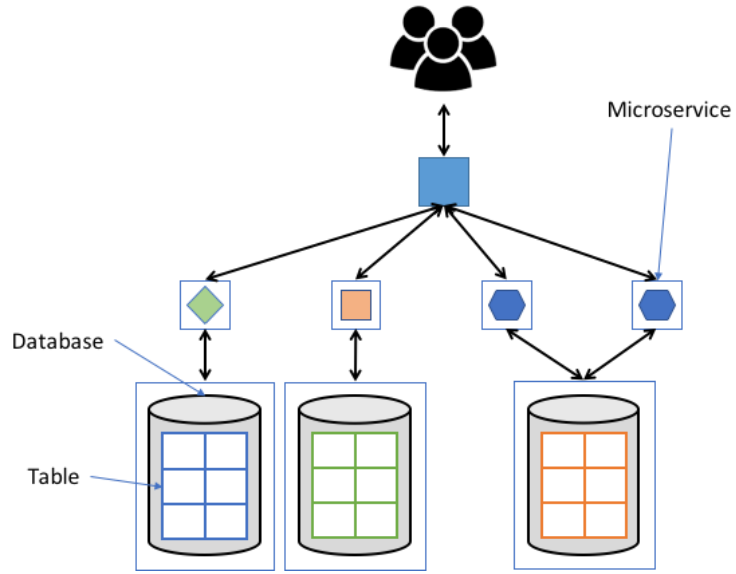
Services Development - You build it AND you run it.

Development is Platform and Language Agnostic

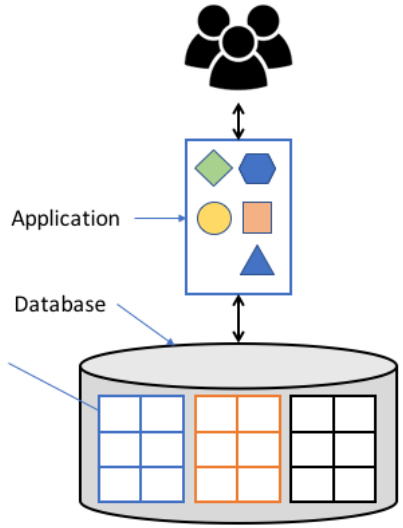
Change in EIP Practices, part 2 of 3

When Adopting a Microservice Architecture

- **Decentralized business and messaging rules** – complex centralized message handling such as Enterprise Service Bus must give way to simple communications protocols. This is referred to as “smart endpoints, dumb pipes.”
- **Decentralized governance** – minimize top down standards or technology sets and allow in-house open source model for sharing useful, tested code
- **Decentralized data management** to service-level data management, often called polyglot persistence.



Microservices Distributed Database



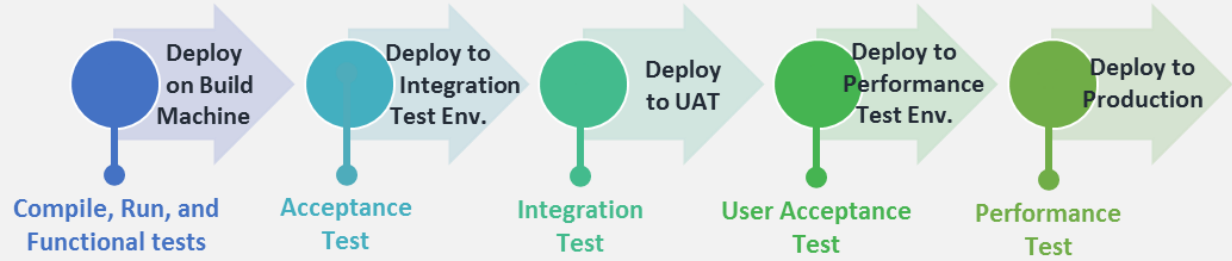
Monolithic Database

Change in EIP Practices, part 3 of 3

When Adopting a Microservice Architecture

- **Organize teams around business capabilities not technology layer**
  - **cross functional teams by business capability are requisite**
  - **traditional technology layer based teams (e.g., a database applications team or a user interface team) means extensive cross-team collaboration and costly delays**
- **Move beyond Continuous Integration to Continuous Delivery**
  - Maintain tight integration requirements within the microservice while easing external integration requirements
  - Continuous Delivery means development teams build software so that the software can be released to production at any time and by anyone on the team

Example of a Basic Automated Build Pipeline that Supports Continuous Integration



Migration Strategy: Use Incremental Refactoring

- The process of transforming a monolithic EIP into a series of microservices is a modernization effort for both the application and its governance.
- This migration is a perfect fit for incremental refactoring. It should not be approached as a complete application rewrite, often called “big bang.”
- Incremental refactoring enables the development team to build experience with microservices
- Incremental refactoring allows teams to build experience with the service extraction process

“Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure."

Martin Fowler

Migration – Choose a Few Services to Extract from the Monolithic Application

- Monolithic applications often consist of hundreds of modules
- Determine which modules to extract first. Choose modules that are
  - **Easy to extract**
  - **change frequently**
  - **have significantly different resource requirements compared to other modules**
  - **Do not share resource requirements**
  - **implement computationally expensive algorithms.**

When migrating from monolithic to microservices, using incremental refactoring is akin to safely servicing your car when driving 70mph on the highway, and stopping the car is not an option.

Migration – How to Extract a Module

- **Step 1 – define a coarse-grained interface between the module and the monolith**
  - **Consider creating a bidirectional API**
  - **Pay close attention to the complexities of business logic refactoring. Significant code changes may be needed to break dependencies**
- **Step 2 – Turn the module into a free-standing service**
  - **Write code for the monolith and service to communicate through an API mechanism**
  - **Combine module with a microservice framework that handles cross-cutting concerns such as service discovery**

Over time, as modules continue to be extracted into services, the amount of functionality implemented by the monolithic application shrinks until either it disappears entirely or it becomes just another microservice

Illustration of Monolithic Module Refactoring