**MICROSERVICES**

KEY CONCEPTS AND CONSIDERATIONS

INTRODUCTION

Microservices architecture is an approach to application development where applications are built as a collection of small, independently deployable services. This report provides an overview of the key concepts of microservices, their advantages, and the challenges associated with their implementation. The evolution of microservices from traditional monolithic architectures offers new possibilities for scalability and flexibility in modern software systems.

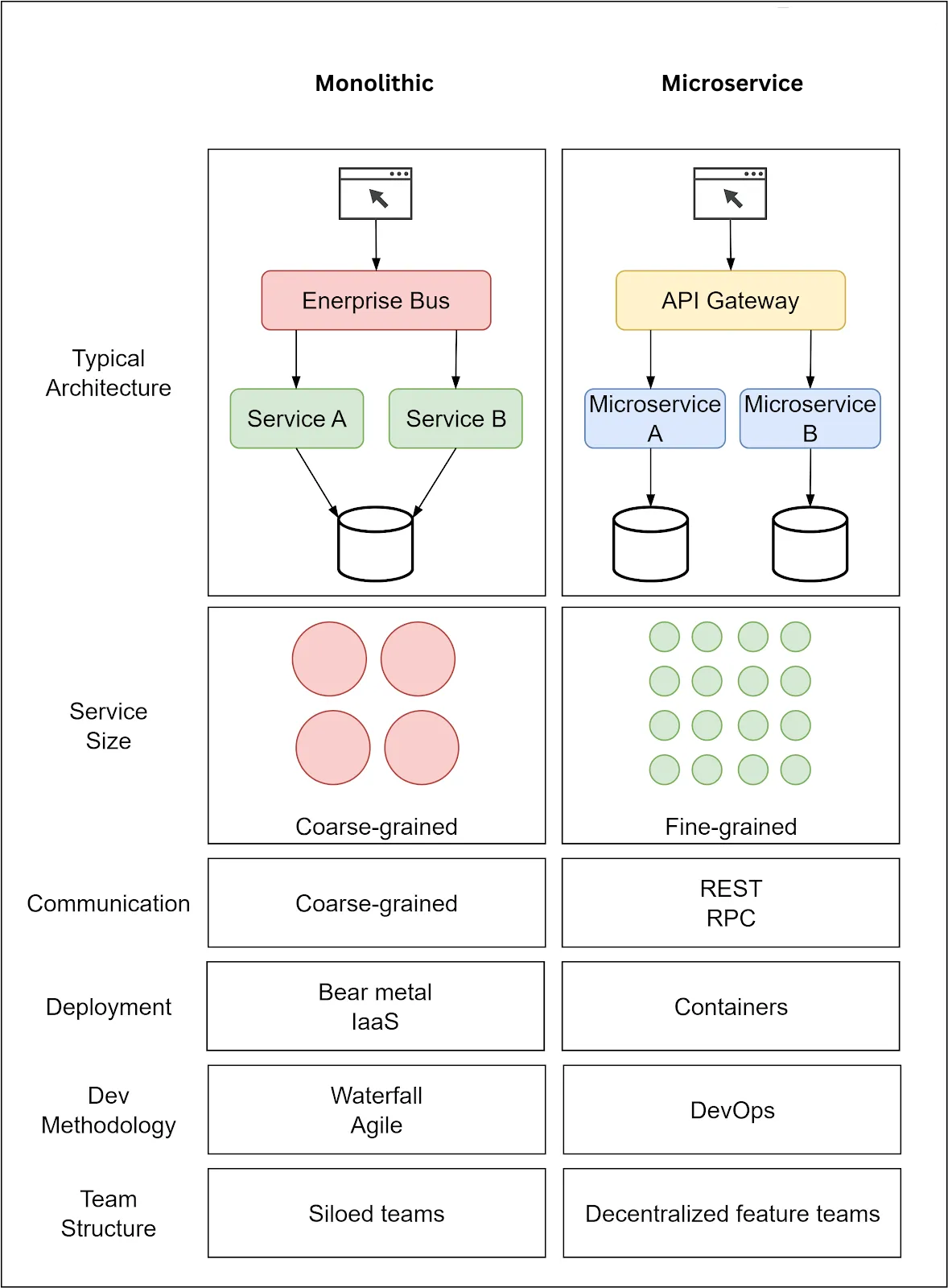


Figure 1 Monolithic vs Microservice Architecture

PRINCIPLES OF MICROSERVICES

1. **Decentralization**

In a microservices architecture, each service is responsible for its own data and logic. Services operate independently and don't share a central codebase or database.

* **Independent Development & Deployment:** Teams can develop, test, deploy, and scale services without heavily affecting other parts of the system.
* **Resilience:** If one service fails, it's less likely to cause a cascading failure throughout the entire application.

1. **Componentization**

Microservices are designed as self-contained, replaceable components. They focus on a single, well-defined business capability.

* **Ease of Replacement/Upgrade:** A problematic service can be replaced or upgraded with minimal impact on the overall system if it has well-defined interfaces.
* **Black Box Functionality:** The internal implementation of a service is hidden. Changes within a service won't break other services as long as the API contract remains the same.

1. **Autonomy**

Microservices are designed to have a high degree of autonomy, meaning they can function with minimal dependence on other services. Teams responsible for a service have ownership of its full development and operational lifecycle.

* **Independent Teams:** Teams have the freedom to choose technologies, development processes, and deployment schedules that best suit their service, reducing coordination overhead and bureaucracy.

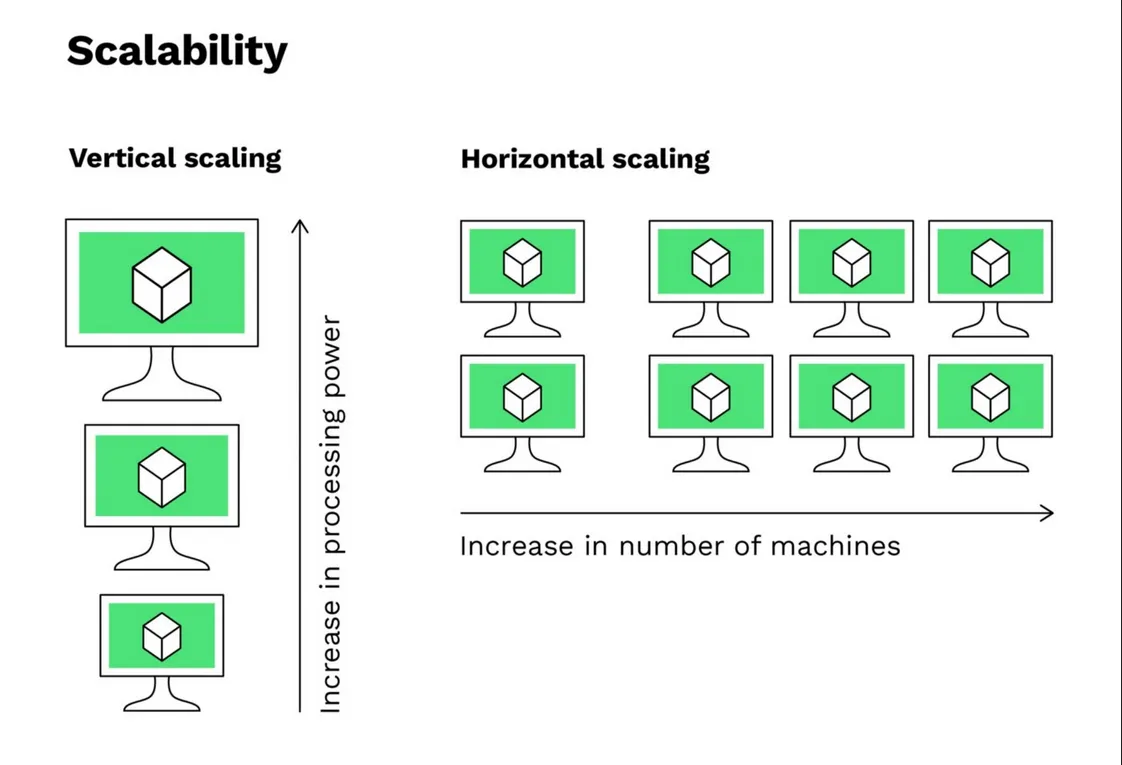
1. **Technology Diversity**

Microservices architecture encourages the use of different technologies, programming languages, frameworks, and databases for individual services. This allows teams to choose the best tools for the specific needs of each service, rather than being limited by a single technology stack for the entire application.

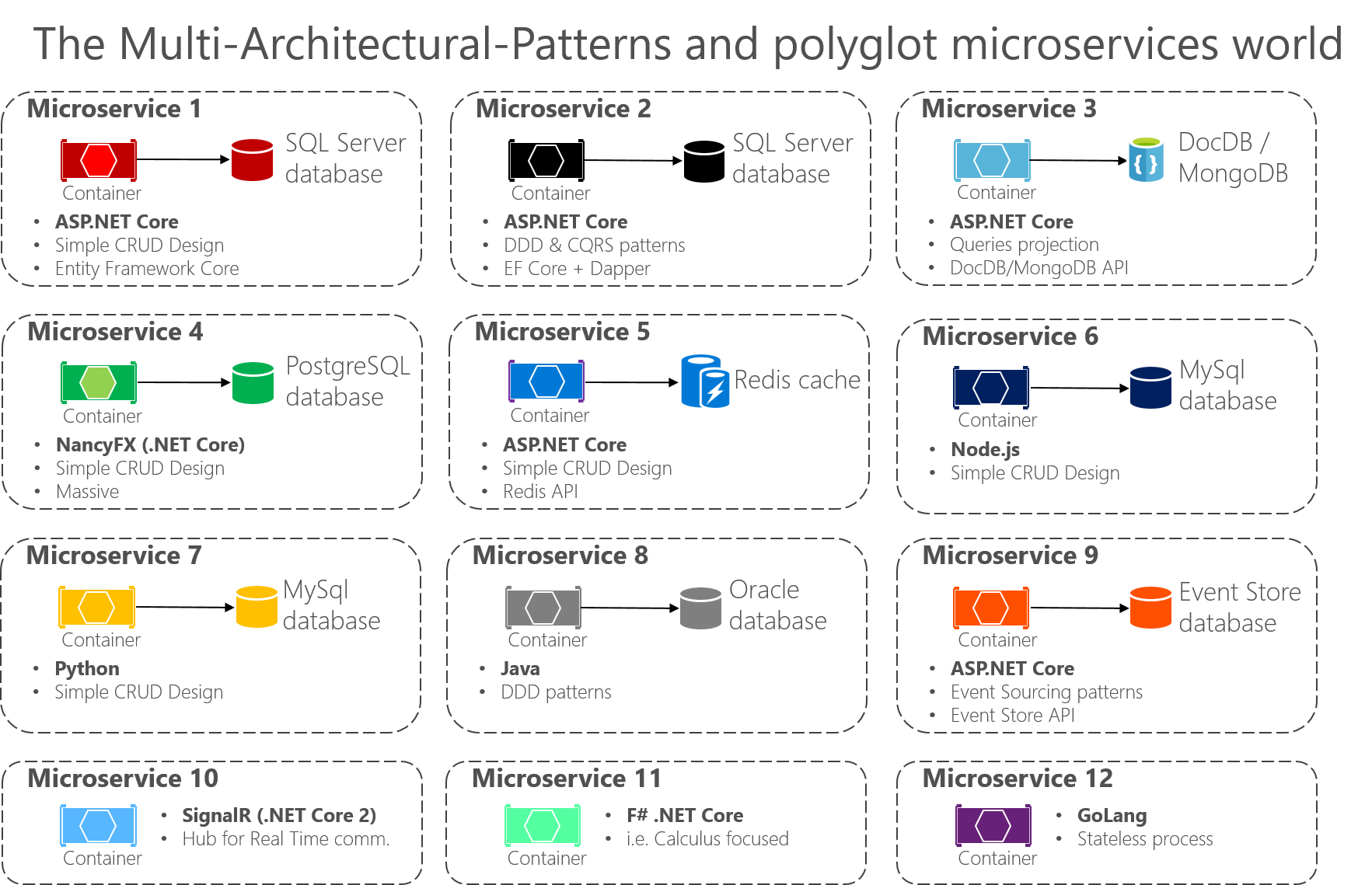
* **Innovation:** The freedom to choose different technologies fosters innovation and experimentation, allowing teams to leverage the latest advancements in specific areas.

ADVANTAGES OF MICROSERVICES

* **Scalability:** Microservices are like independent building blocks. If you need more processing power for a specific function, you can easily scale up just that microservice without affecting the others. This allows for much more efficient use of resources.

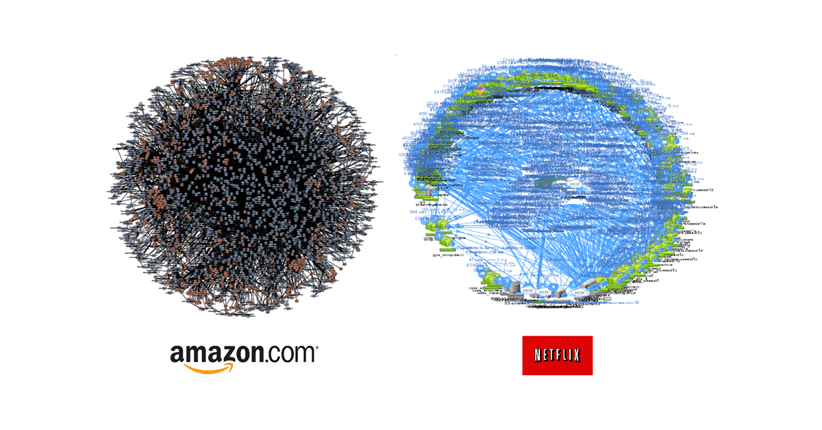


* **Resilience:** Because each microservice is separate, a failure in one service won't necessarily bring down the entire application. Other parts of the system can continue to function while the issue is addressed in the isolated microservice. This makes applications built with microservices much more fault tolerant.
* **Technological Agility:** Microservices allow you to choose the best tool for the job. Different parts of your application can be built using different programming languages and technologies, depending on what's most efficient for that specific function. This freedom makes it much easier to adopt new technologies as they emerge.



DISADVANTAGES OF MICROSERVICES

* **Complexity:** Microservices introduce a lot of moving parts. You must manage multiple services, their interactions, and ensure they all work together seamlessly. This can be significantly more complex than a monolithic architecture.



* **Data Integrity:** With data potentially spread across multiple services, maintaining consistency becomes a challenge. Transactions that involve multiple services need careful coordination to ensure all changes are made successfully or rolled back if something fails.
* **Network Issues:** Microservices communicate with each other over a network. Network latency, congestion, or outages can disrupt communication and lead to application failures. Extra care needs to be taken to ensure reliable communication between services.
* **Skill Set:** Developing, deploying, and maintaining microservices requires a broader skill set from development teams. They need expertise in distributed systems, API design, DevOps practices, and potentially multiple programming languages.