

7 Microservices Interview Questions



BYTEBYTEGO

DEC 14, 2023 · PAID



195



5



13

Share



Microservices are a popular way to structure software systems today. As companies grow bigger and use more cloud computing, microservices help tackle complexity.

In this issue, we review some key microservices concepts and common questions that come up in interviews:

1. What are microservices?
2. What issues do microservices aim to solve?
3. What new challenges do microservices introduce?
4. What are some popular microservices solutions?
5. How does monitoring and alerting work with microservices?
6. How are logs collected and analyzed?
7. What is a Service Registry?

Now let's start with the definition of microservices.

1. What Are Microservices?

We can quote from Martin Fowler and Adrian Cockcroft on key aspects of microservices.

From [Martin Fowler](#):

“The microservice architectural style is an approach to developing a single application as a suite of **small services**, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and **independently**

deployable by fully automated deployment machinery. There is a bare minimum of **centralized management** of these services, which may be written in different programming languages and use different data storage technologies.”

From [Adrian Cockcroft](#):

“[Microservices are loosely coupled service-oriented architecture with bounded contexts.](#)”

Key Aspects of Microservices

From the insightful definitions by Martin Fowler and Adrian Cockcroft, we can summarize these key aspects of a microservice architecture:

1. Decompose monolithic application into small, independent services. This allows different product teams to develop, test, and deploy services that align to specific business capabilities. Useful for large organizations to break down monolithic systems and improve productivity.
2. Loosely coupled services communicating via APIs. Frontend/backend components communicate via REST, while inter-service communications use RPC for efficient request/response.
3. Carefully designed around bounded contexts. Each service has clear module boundaries and encapsulated domain logic to avoid tight coupling between services.
4. Enables effective DevOps practices. It is an important part of the microservice development methodology. Small, full-stack teams fully own specific services end-to-end. Containerization, automation, and container orchestration are used to effectively deploy microservices.
5. Horizontally scalable by design, resilient to failures. Services can scale out independently as needed. Built with fault tolerance in mind.
6. Decentralized governance and flexibility. Teams can choose whatever technology makes sense for their service.
7. Requires extensive monitoring and instrumentation due to:
 - a. Growth in services - As monoliths decompose into many independent services, the number of components to track grows quickly.

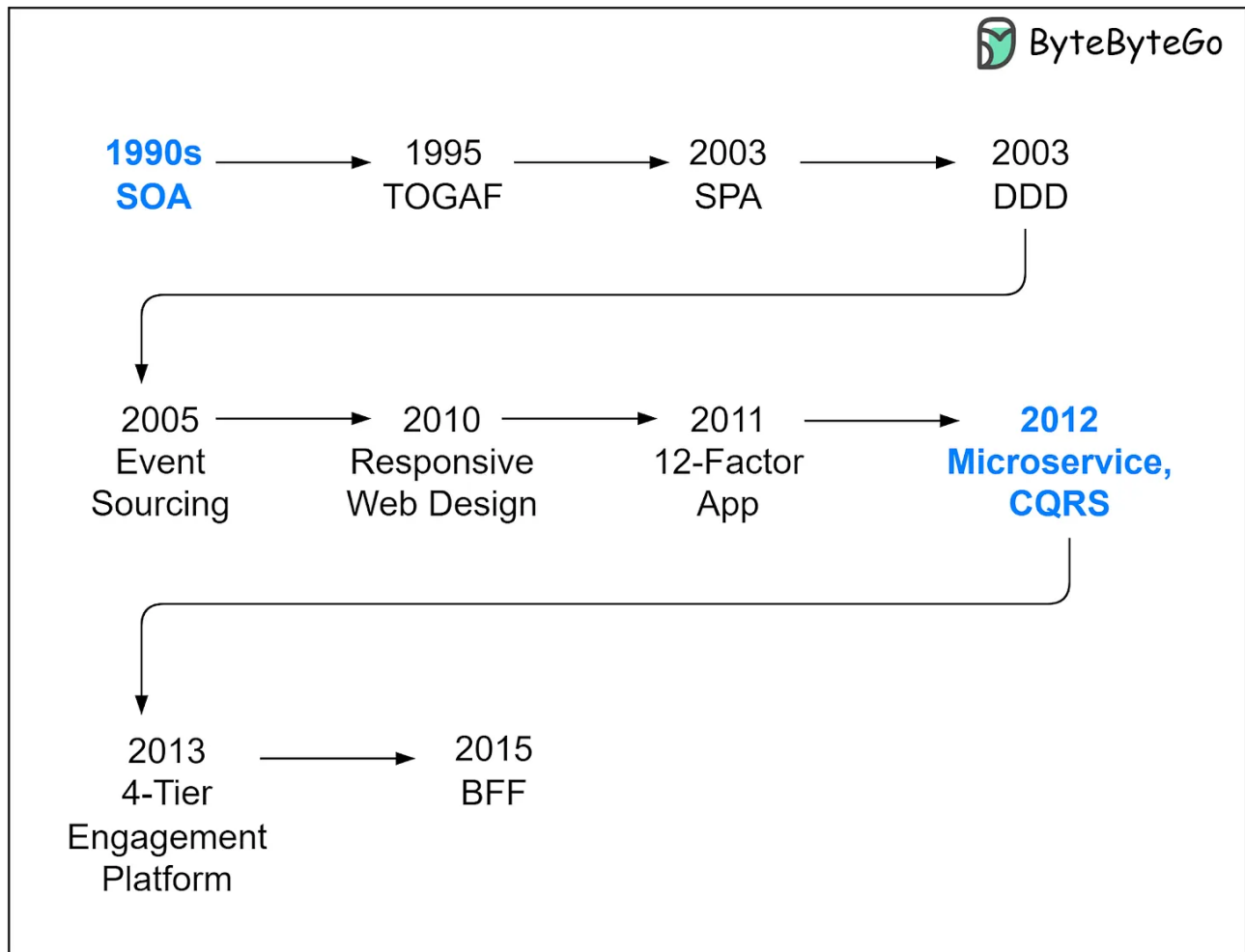
- b. Infrastructure abstraction via containers/orchestrators - Runtime platforms like Kubernetes handle infrastructure. So monitoring must happen at the application code level using sidecars to aggregate logs, metrics and traces.

2. What Are the Differences Between SOA and Microservices?

Service-oriented architecture (SOA) and microservices architecture styles are important milestones in software architecture's evolution. The diagram below shows the progression of key architectural styles.

Service-oriented architecture emerged in the late 1990s to help manage enterprise software systems' growing complexity. In the 2000s, SOA gained more industry attention and adoption by companies. However, SOA faced implementation complexity challenges.

Then in the 2010s, microservices architecture emerged in response to SOA's limitations. Many large internet companies started adopting microservices to break down their services into smaller components. Microservices gained momentum with cloud computing's evolution, as containers and orchestration tools made microservices' development, deployment, and monitoring easier.



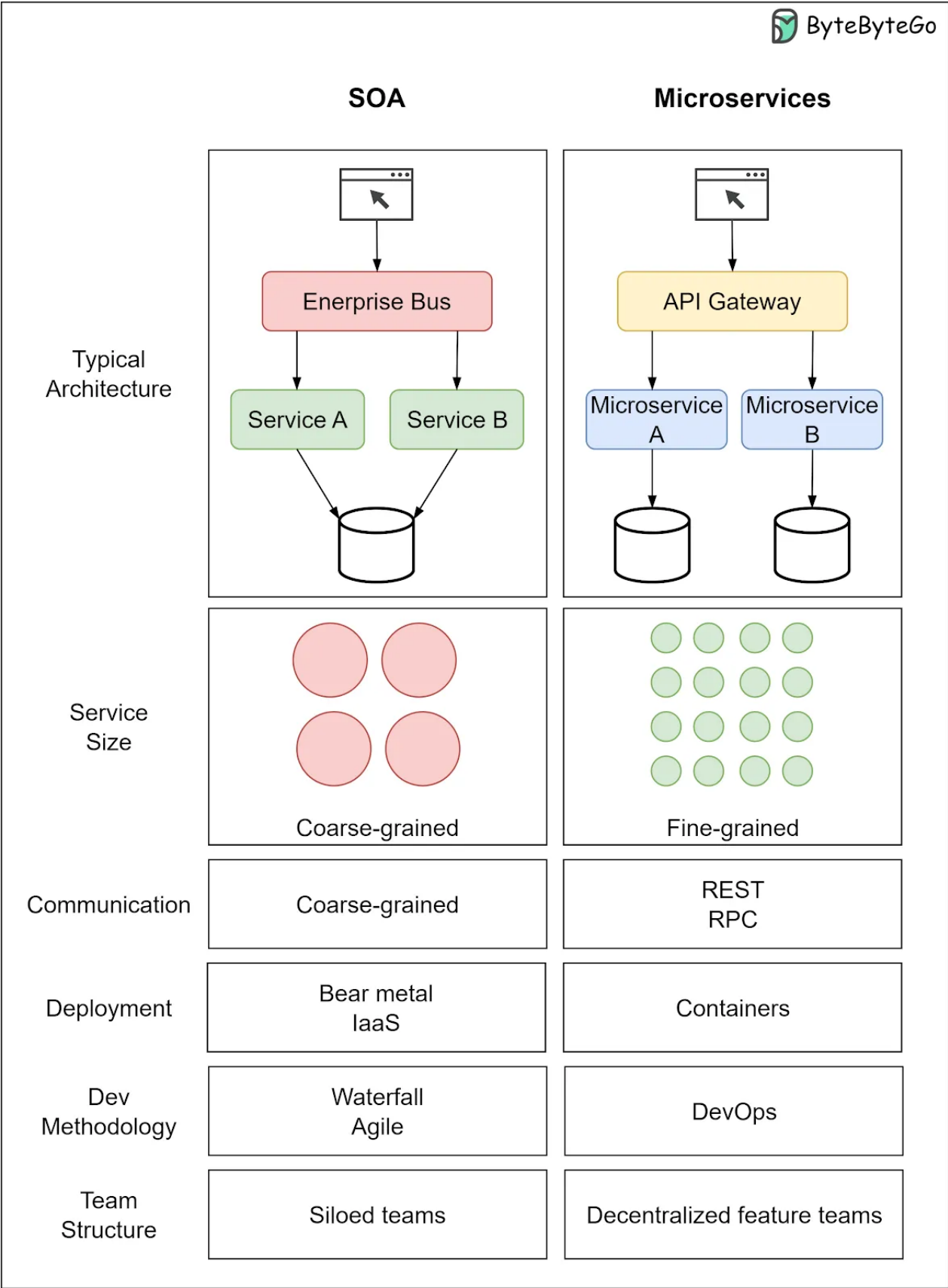
Let's compare their differences in more detail. The diagram below lists some of the differences.

The SOA architectural style offers coarse-grained services, typically a centralized approach where services are grouped by business functions and shared across multiple applications. The microservice style offers fine-grained service granularity through a decentralized approach where small, independent services perform specific functions within an application context.

The communication methods also evolve over time. SOA emphasizes uniform communication protocols and standardized interfaces for services to interact. Microservices lean towards diverse communication protocols and interfaces, often based on REST or message queues.

Cloud computing has evolved from Infrastructure-as-a-Service (IaaS) to Platform-as-a-Service (PaaS) to container-based PaaS. So microservice-based applications deploy on containers by default.

As the technical architecture changes, the organizational structure mirrors it (Conway’s law). So with microservices, the team structure requires multi-functional product teams. Each team focuses on a specific domain.



3. What Problems Do Microservices Solve?

Microservice architecture emerged in the 2010s to solve some key challenges:

Cloud Computing Needs New Architecture

Traditional software architectures don't fully utilize the flexibility cloud computing enables. Containerization technology decomposes hardware resources into smaller units. Orchestration tools easily scale services. So microservices architecture emerged to better leverage these cloud computing advancements.

Decouple Services and Teams for Speed

Monolithic and SOA applications often have tight dependencies among modules, making adaptation to fast-changing business requirements difficult given the internet industry's explosive growth. Microservices architecture segregates each business domain into a bounded context. Product teams can focus on specific domain requirements. Services and teams communicate via predefined API specifications, facilitating quicker feature delivery and more iterative market testing.

Enable Robust Development Methodologies

A monolithic application failure can bring down the entire application, while microservices are designed to be fault-tolerant. Failures in a single microservice are isolated from other services. Additionally, management tools provide flexible request routing or service downgrade. Chaos engineering, which deliberately injects failures to proactively test vulnerabilities, takes this fault-tolerant design principle to an extreme. DevOps, a set of practices and tools to emphasize collaboration between functions, is a natural fit. These methodology changes allow rapid, reliable product delivery.

4. What Challenges Do Microservices Introduce?

While microservice architecture solves many problems posed by traditional architectures and fits cloud computing's evolution, it also brings challenges.

Operational Overhead

In a microservices system, it is easy to provision containers, often leading to an explosion of instances. Monitoring and debugging these without proper tooling can be a nightmare.

Increased Complexity

Microservices architecture encourages segregated domains, so managing service communications and relationships can be complicated. We need to leverage service discovery and configuration management to efficiently manage service instances. This also poses challenges for integration and testing.

Organizational Transformation

As Conway's law states, a software architecture shapes the organizational structure. Microservices architecture requires team structured accordingly.

Distributed Transaction

Data becomes more distributed as each domain has an independent database. While traditional architectures leverage database transactions to ensure data consistency, microservices may need distributed transactions for the same purpose.

Performance Overhead and Challenging Error Handling

In monolithic applications, different modules communicate via fast in-process function calls. If a failure occurs, the function call gets an immediate response, allowing quick error handling. In contrast, microservices use slower service-to-service network communications, introducing extra latency. Also, microservices typically communicate via RPC. This makes error handling across services less predictable compared to monoliths.

5. What Are the Basic Components of Microservices Architecture?

A typical microservices architecture includes:

Load Balancer

The load balancer sits in front of the API Gateway and distributes incoming requests across multiple API gateway instances. This prevents any single gateway instance from becoming a bottleneck. Common algorithms the load balancer can use to distribute requests include: round robin, sticky sessions, least connections.

API Gateway

The API gateway handles all incoming requests and routes them to the relevant microservices. It communicates with the identity provider to handle authentication and authorization of requests. The gateway leverages the service discovery component to locate the required microservices for a particular requests. Additionally, it can apply filtering rules, policies, and transformations to the requests before routing them to the backend.

CDN (Content Delivery Network)

A CDN (Content Delivery Network) is a group of geographically distributed servers that cache static content in locations closer to end users for faster content delivery. The application first checks the CDN to serve static content. If the content is not in the CDN, the CDN handles the fallback to retrieve that content from the slower backend services.

Identity Provider

An identity provider handles authentication and authorization for users.

Microservices

Microservices are designed and deployed around specific domains or areas of capability. Each microservice domain has its own database or data store. The API Gateway communicates with the microservices using REST APIs or other protocols. Microservices within the same domain talk to each other via RPC (Remote Procedure Call) for internal interactions.

Service Registry and Discovery

The service registry and discovery component handles the registration and discovery of microservices. As new microservice instances spin up, they register themselves

with this service. The API Gateway leverages this component to look up and locate relevant microservices that it needs to route requests to.

Message Queue

A message queue allows microservices to communicate with each other asynchronously. For example, the payment service may take a long time to complete a payment transaction. In this case, the upstream checkout system can send a request message to a queue such as Kafka and await the response asynchronously instead of blocking. This prevents delays in upstream processes while long-running downstream services finish.

Cache

A cache can be used to store hot data, improving responsiveness for frequent queries. However, when adding caching to microservices architecture, pay careful attention to ensure data consistency between the cache and the underlying databases. Stale, out-of-date cached data could lead to incorrect application behavior.

Data Storage: database and object store

Databases and object stores are used to persist the application's data. Different types of databases or data stores can be selected depending on the data model and scalability requirements. For example, in an eCommerce application, a wide-column database like Cassandra might be chosen to store shopping cart items due to its scalability.

Configuration Management

A configuration management system is a centralized system to manage and distribute service configurations.

Monitoring and Logging

Observability through monitoring and logging is crucial for maintaining a microservices architecture. Since microservices systems comprise far more service instances than traditional applications, automated monitoring and distributed tracing are important for identifying and debugging problems. Continuous

monitoring also enables proactive capacity planning and forecasting to detect bottlenecks.

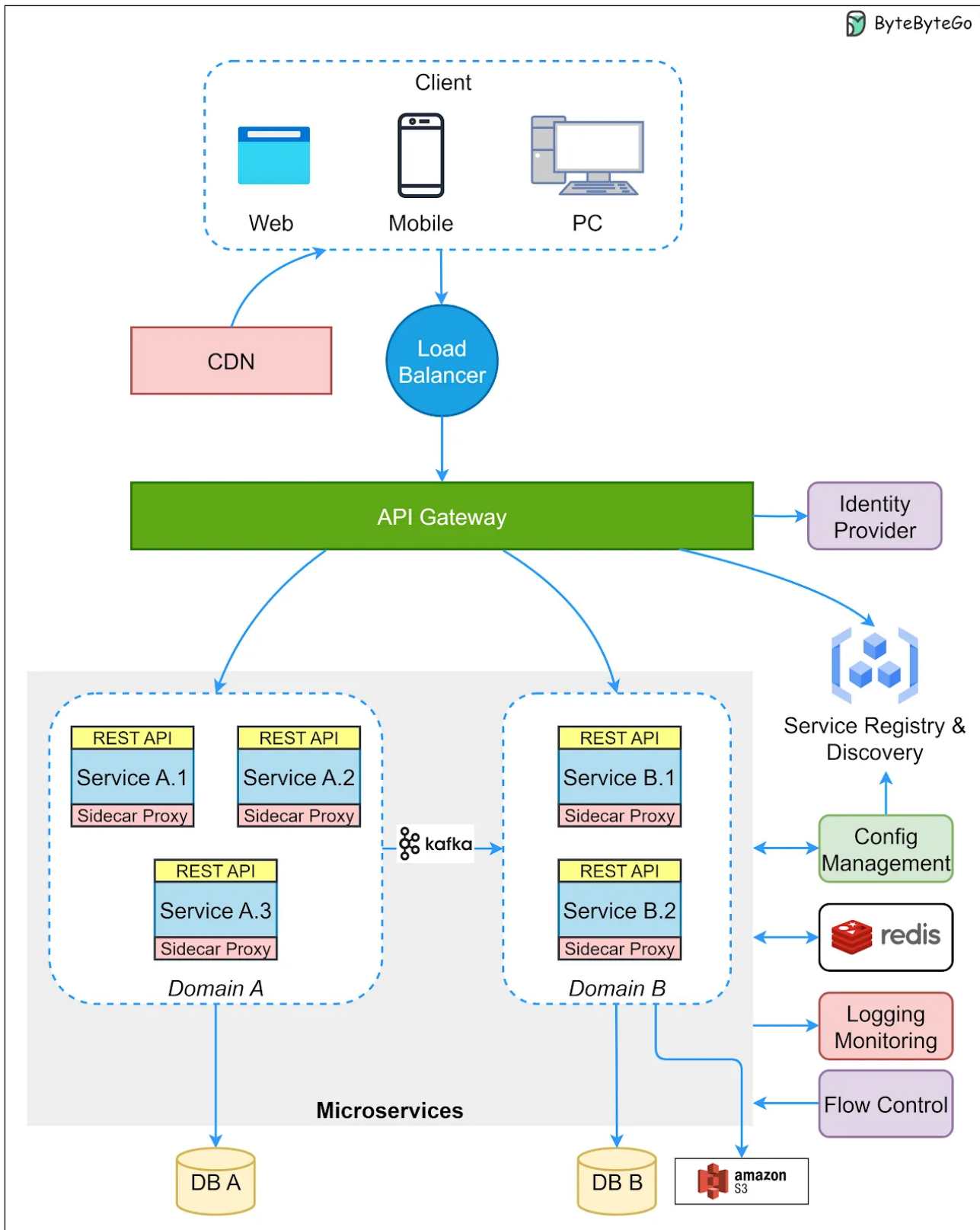
Flow Control

Flow control tools are sometimes necessary for handling faults in complex microservices systems. These tools help restrict or redirect traffic when issues occur. For example, Hystrix controls interactions between microservices by isolating failing services. Sentinel acts as a circuit breaker to control traffic flow to specific microservices as needed.

Service Mesh and Service Orchestration

A service mesh provides infrastructure for handling communications and control across microservices. It uses a sidecar proxy deployed with each service instance to manage incoming and outgoing traffic flows. This enables capabilities like monitoring and traffic management across services.

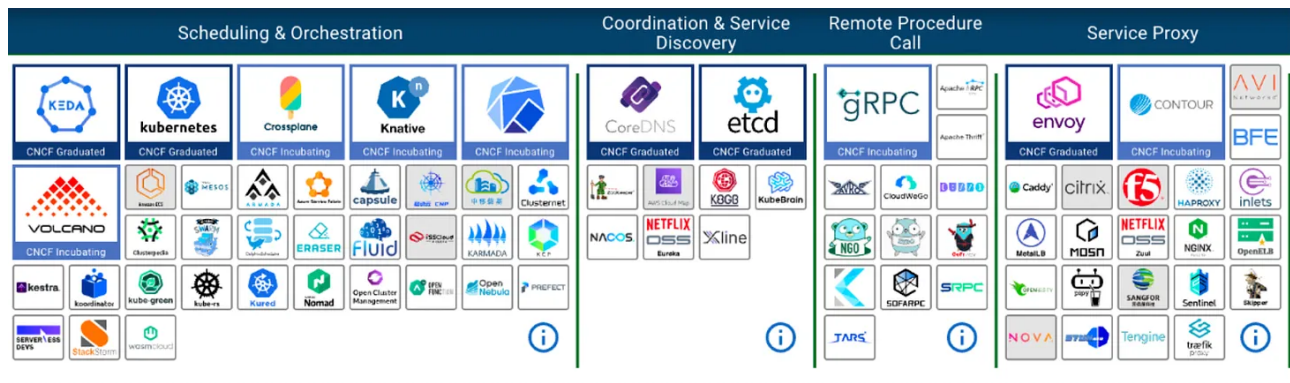
Service orchestration focuses on coordinating the sequence and workflow of steps. It leverages the capabilities provided by the service mesh to manage microservice interactions and control workflow execution. Kubernetes, influenced by Google's Borg cluster manager, is a popular container orchestration system maintained by Cloud Native Computing Foundation (CNCF).



6. What Are Some Popular Microservice Solutions?

The Cloud Native Computing Foundation (CNCF), a Linux Foundation project, was founded in 2015 to help advance [container technology](#) and build an ecosystem for

cloud computing. The CNCF Cloud Interactive [Landscape](https://landscape.cncf.io/) lists tools and their maturity levels. For example, the landscape diagram shows some “orchestration & management” tools.



Source: <https://landscape.cncf.io/>

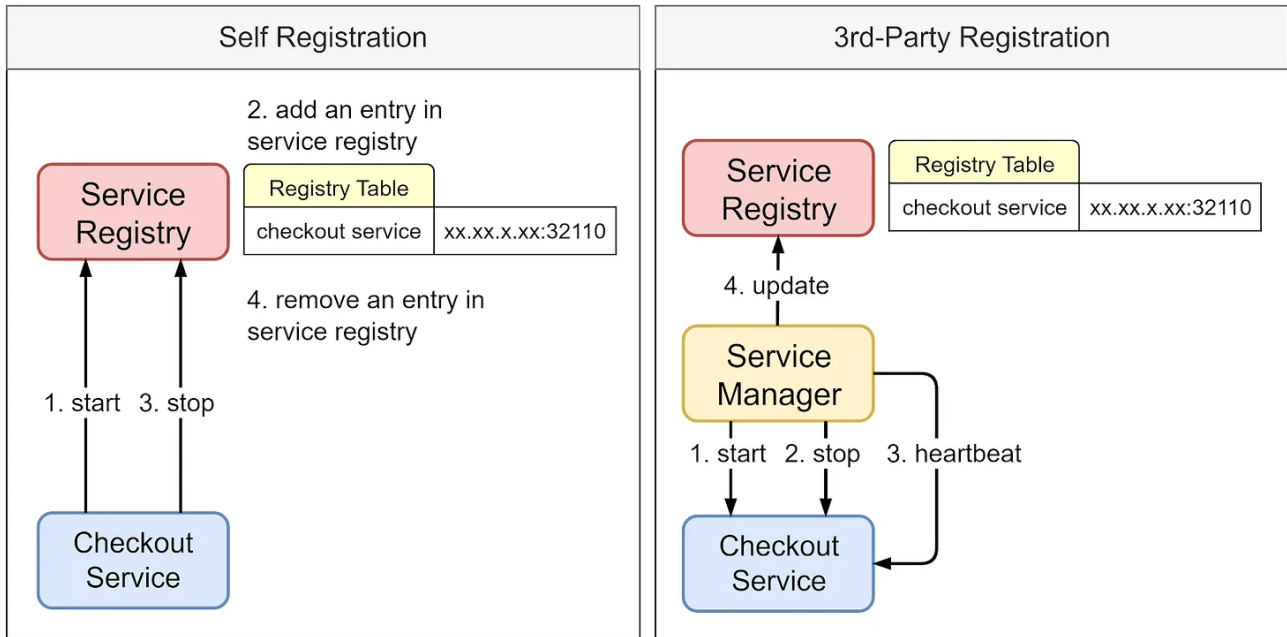
[Spring Cloud](https://springcloud.io/) is an out-of-the-box solution that offers a set of integrated tools to create basic microservices components.

7. What is a Service Registry?

A service registry manages service registration and discovery. It has a centralized database to store information about services and their locations. It dispatches client requests to relevant microservice instances.

The diagram below illustrates how services register when starting up and shutting down. There are two approaches: self-registration and third-party registration.

In self-registration, services report their status to the service registry, which then updates the registry table. In third-party registration, a service manager handles lifecycles of services by starting and stopping instances and updating the registry. The service manager also pings services to monitor service health.

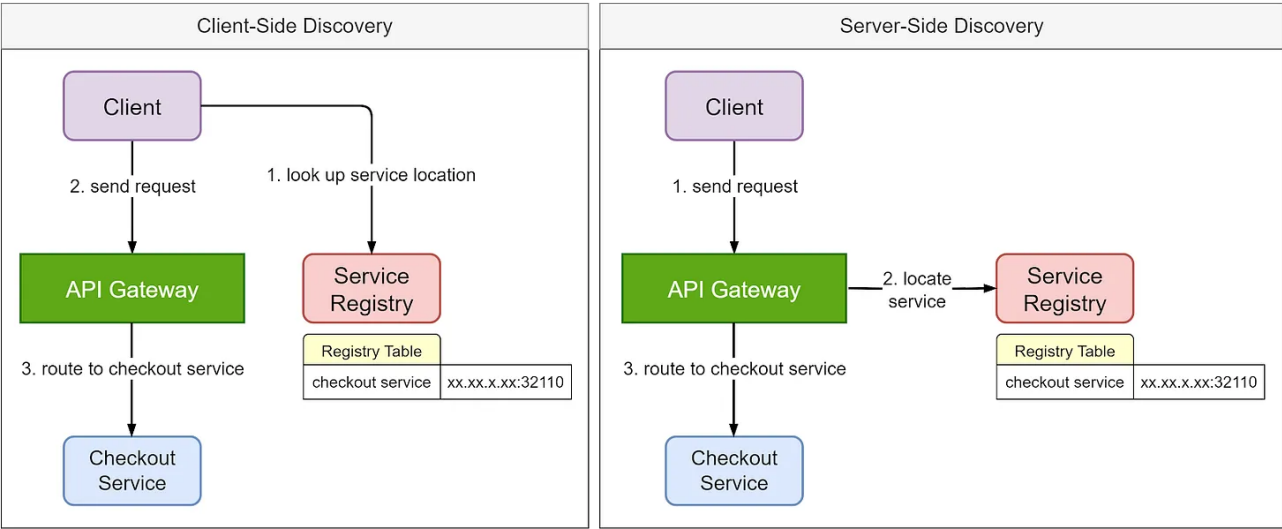


Let's look at another important function of the service registry - service discovery. The diagram below shows two service discovery processes: client-side discovery and server-side discovery.

In client-side discovery, clients first query the service registry for service information. Then they send requests to the API gateway without needing location lookups. In this way, the service discovery function is moved out of the API gateway layer.

In server-side discovery, the API gateway handles locating services. It gets client requests first and redirects them to appropriate services. The API gateway can cache locations to speed up the discovery process. However, this introduces extra complexity to the API gateway in two ways:

1. The API gateway needs to implement service discovery logic.
2. Keeping its location cache consistent with registry data is challenging.




Some examples of service registry products are Netflix Eureka and Hashicorp Consul.

We will cover more microservice interview questions next week. Stay tuned!




195 Likes · 13 Restacks

5 Comments



Write a comment...

- 


Nicholas Jan 5

You have Conway's law backwards

♡ LIKE

💬 REPLY

🔗 SHARE

...
- 

Filip Talev Dec 28, 2023

Poorly written. It requires a rewrite.

♡ LIKE

💬 REPLY

🔗 SHARE

...

3 more comments...

© 2024 ByteByteGo · [Privacy](#) · [Terms](#) · [Collection notice](#)
[Substack](#) is the home for great writing