Micro Services

MicroServices are a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.

Micro services & Spring Boot, Cloud

MicroServices by Pratap Kumar

- Microservices are distributed, loosely coupled software services that carry out a small number of well-defined tasks.
- Microservices break up your code into small, distributed, and independent services that require careful forethought and design.
- Fortunately, Spring Boot and Spring Cloud simplify your microservice applications, just as the Spring Framework simplifies enterprise Java development.
- Spring Boot removes the boilerplate code involved with writing a REST-based service.
- Spring Cloud provides a suite of tools for the discovery, routing, and deployment of Microservices to the enterprise and the cloud.
- Spring Boot and Spring Cloud will provide an easy migration path from building traditional, monolithic Spring applications to microservice applications that can be deployed to the cloud.

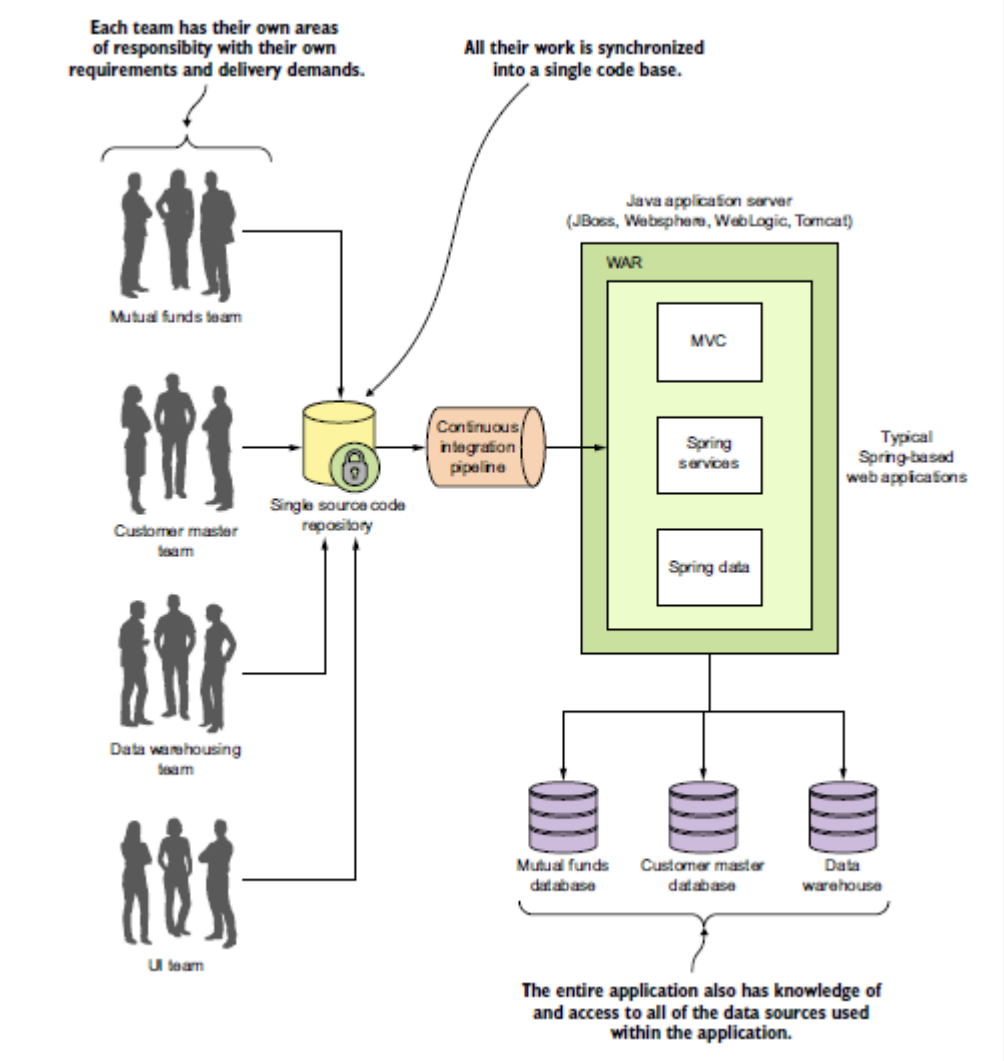
Monolithic

- A software system is called "monolithic" if it has a monolithic architecture, in which functionally distinguishable aspects (for example data input and output, data processing, error handling, and the user interface) are all interwoven, rather than containing architecturally separate components.

- In a monolithic architecture, an application is delivered as a single deployable software artifact. All the UI (user interface), business, and database access logic are packaged together into a single application artifact and deployed to an application server.

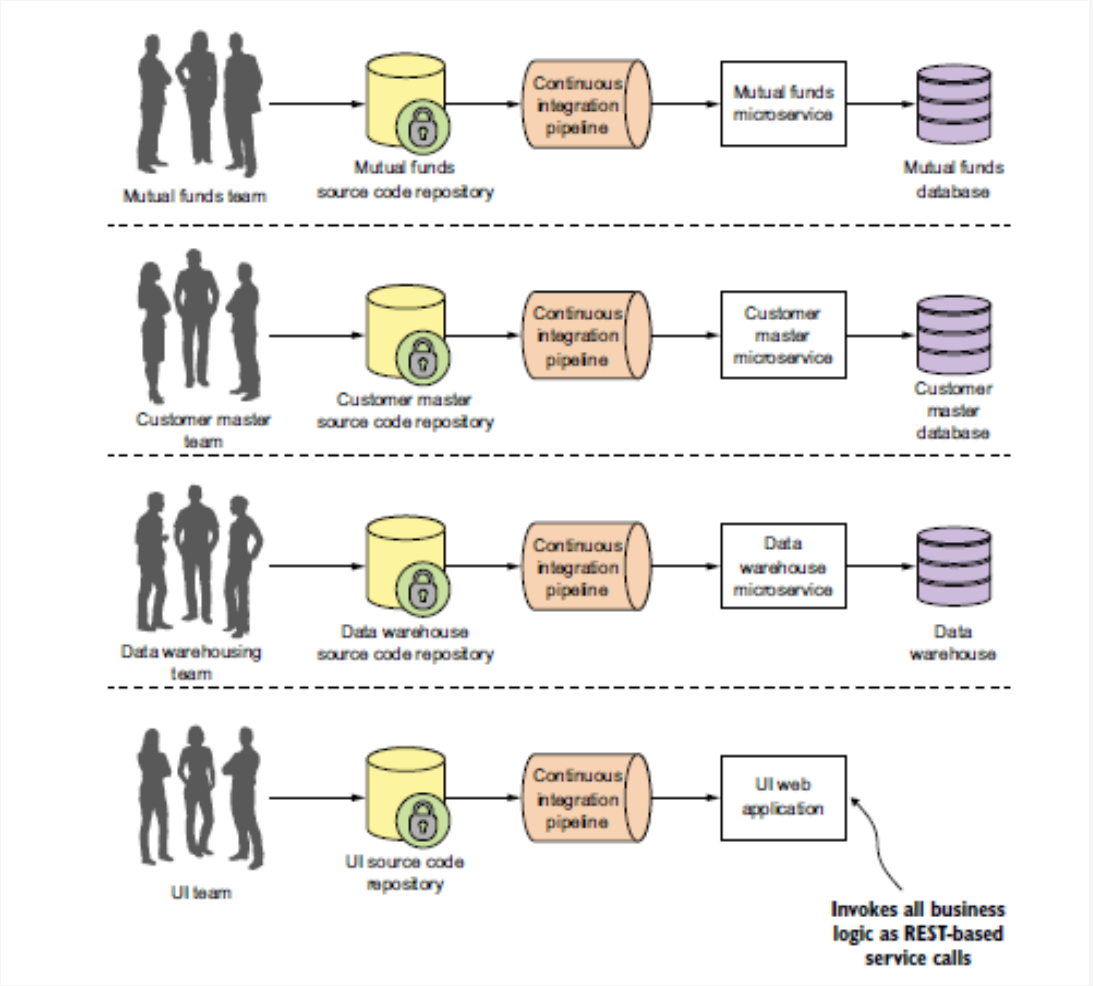
- As the size and complexity of the monolithic application grew, the communication and coordination costs of the individual teams working on the application didn't scale. Every time an individual team needed to make a change, the entire application had to be rebuilt, retested and redeployed.

Monolothic



Micro services

MicroServices by Pratap Kumar



Microservice Architecture Characteristics

MicroServices by Pratap Kumar

- Application logic is broken down into small-grained components with well defined boundaries of responsibility that coordinate to deliver a solution.

- Each component has a small domain of responsibility and is deployed completely independently of one another. Microservices should have responsibility for a single part of a business domain. Also, a microservice should be reusable across multiple applications.

- Microservices communicate based on a few basic principles (notice I said principles, not standards) and employ lightweight communication protocols such as HTTP and JSON (JavaScript Object Notation) for exchanging data between the service consumer and service provider.

Microservice Architecture Characteristics

- The underlying technical implementation of the service is irrelevant because the applications always communicate with a technology-neutral protocol (JSON is the most common). This means an application built using a microservice application could be built with multiple languages and technologies.

- Microservices—by their small, independent, and distributed nature—allow organizations to have small development teams with well-defined areas of responsibility. These teams might work toward a single goal such as delivering an application, but each team is responsible only for the services on which they're working.

Spring & Micro Service

MicroServices by Pratap Kumar

- **What is Spring and why is it relevant to Microservices?**
  - Dependency Injection
  - Light Weight
  - Spring Boot strips away many of the "enterprise" features found in Spring and instead delivers a framework geared toward Java-based, REST-oriented (Representational State Transfer)1 microservices. With a few simple annotations, a Java developer can quickly build a REST microservice that can be packaged and deployed without the need for an external application container.
  - the core concept behind REST is that your services should embrace the use of the HTTP verbs (GET, POST, PUT, and DELETE) to represent the core actions of the service and use a lightweight web-oriented data serialization protocol, such as JSON, for requesting and receiving data from the service.
  - The Spring Cloud framework makes it simple to operationalize and deploy microservices to a private or public cloud. Spring Cloud wraps several popular cloud-management microservice frameworks under a common framework and makes the use and deployment of these technologies as easy to use as annotating your code

Micro Service

MicroServices by Pratap Kumar

- ***Why change the way we build applications?***
  - Complexity has gone way up
  - Customers want faster delivery
  - Performance and scalability
  - Customers expect their applications to be available

- To meet the expectation , If we "unbundle" our applications into small services and move them away from a single monolithic artifact, we can build systems that are
  - Flexible
  - Resilient
  - Scalable
- ***Small, Simple, and Decoupled Services = Scalable, Resilient, and Flexible Applications***

What exactly is the cloud?

- The term "cloud" has become overused. Every software vendor has a cloud and everyone's platform is cloud-enabled, but if you cut through the hype, three basic models exist in cloud-based computing. These are
  - Infrastructure as a Service (IaaS)
  - Platform as a Service (PaaS)
  - Software as a Service (SaaS)

Microservice?

- Set of practices meant to increase speed and efficiency of developing and managing software solution that can scale.
- These set of practices are technology agnostic.
- In fact you can build microservice with roughly all programming language
- It is all about applying certain number of principles and architectural pattern that will finally create a microservice architecture.

Micro?

- Big or small
- No universal measure
- "Does one thing"
- Scope of functionalities
- Bounded context
- Identify Sub Domains

Service?

- Independently deployable component
- Interoperability
- Message based communication
- Service Oriented Architecture (SOA)

Micro Service?

- "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 mechanism."

- "Microservices are small, autonomous services that work together"

MicroServices are more than writing the code

Micro Service?

- While the concepts around building individual microservices are easy to understand, running and supporting a robust microservice application (especially when running in the cloud) involves more than writing the code for the service.

- Writing a robust service includes considering several things.
  - Right-sized
  - Location transparent
  - Resilient
  - Repeatable
  - Scalable

Micro Service?

- **Right-sized :** How do you ensure that your microservices are properly sized so that you don't have a microservice take on too much responsibility? Remember, properly sized, a service allows you to quickly make changes to an application and reduces the overall risk of an outage to the entire application.

- **Location transparent :** How you manage the physical details of service invocation when in a microservice application, multiple service instances can quickly start and shut down?

- **Resilient :** How do you protect your microservice consumers and the overall integrity of your application by routing around failing services and ensuring that you take a "fail-fast" approach?

- **Repeatable :** How do you ensure that every new instance of your service brought up is guaranteed to have the same configuration and code base as all the other service instances in production?

- **Scalable :** How do you use asynchronous processing and events to minimize the direct dependencies between your services and ensure that you can gracefully scale your microservices?

Micro Service Pattern

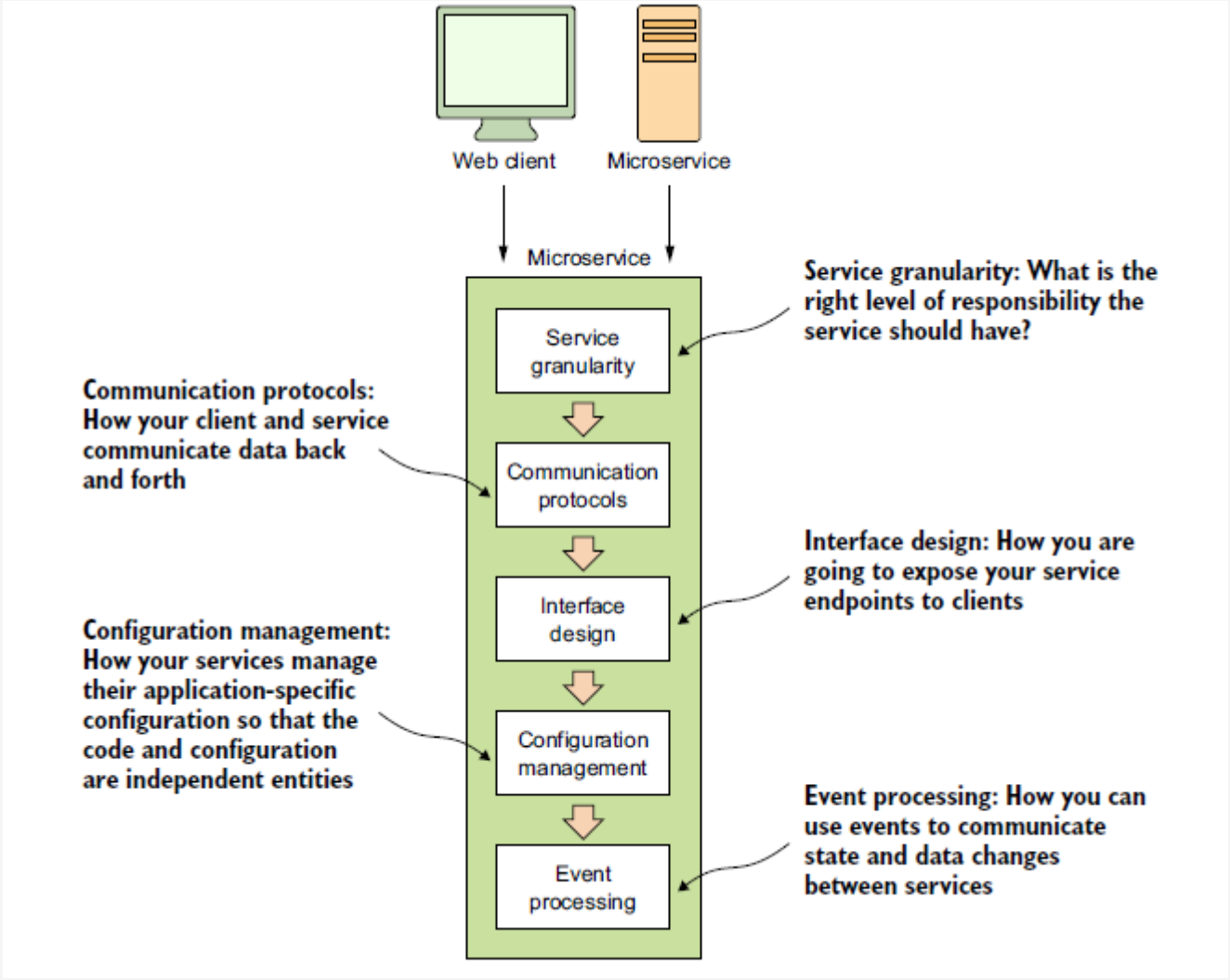
- Core development patterns
- Routing patterns
- Client resiliency patterns
- Security patterns
- Logging and tracing patterns
- Build and deployment patterns

Core development patterns

- The core microservice development pattern addresses the basics of building a microservice

- **Service granularity**: What is the right level of responsibility the service should have?
- **Communication protocols**: How your client and service communicate data back and forth
- **Interface design**: How you are going to expose your service endpoints to clients
- **Configuration management**: How your services manage their application-specific configuration so that the code and configuration are independent entities
- **Event processing**: How you can use events to communicate state and data changes between services

Core development patterns



Microservice routing patterns

- The microservice routing patterns deal with how a client application that wants to consume a microservice discovers the location of the service and is routed over to it.

- In a cloud-based application, you might have hundreds of microservice instances running.

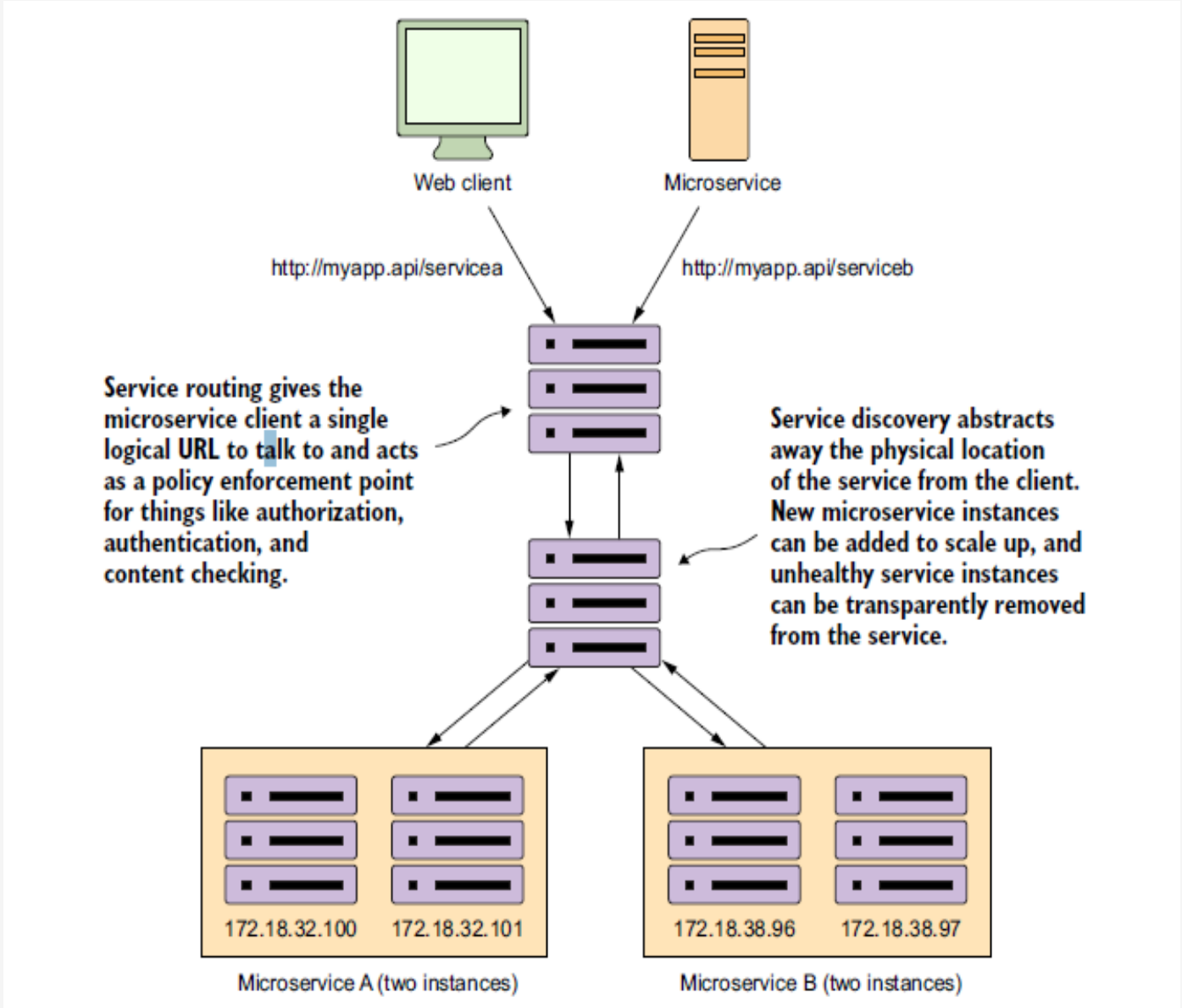
- You'll need to abstract away the physical IP address of these services and have a single point of entry for service calls so that you can consistently enforce security and content policies for all service calls.

Microservice discovery patterns

- **Service discovery** — How do you make your microservice discoverable so client applications can find them without having the location of the service hardcoded into the application?

- How do you ensure that misbehaving microservice instances are removed from the pool of available service instances

Microservice routing patterns



Microservice routing patterns

- **Service routing**— How do you provide a single entry point for all of your services so that security policies and routing rules are applied uniformly to multiple services and service instances in your microservice applications?

- How do you ensure that each developer in your team doesn't have to come up with their own solutions for providing routing to their services?

Microservice client resiliency patterns

- Because microservice architectures are highly distributed, you have to be extremely sensitive in how you prevent a problem in a single service (or service instance) from cascading up and out to the consumers of the service.

- **Client-side load balancing** — How do you cache the location of your service instances on the service client so that calls to multiple instances of a microservice are load balanced to all the health instances of that microservice?

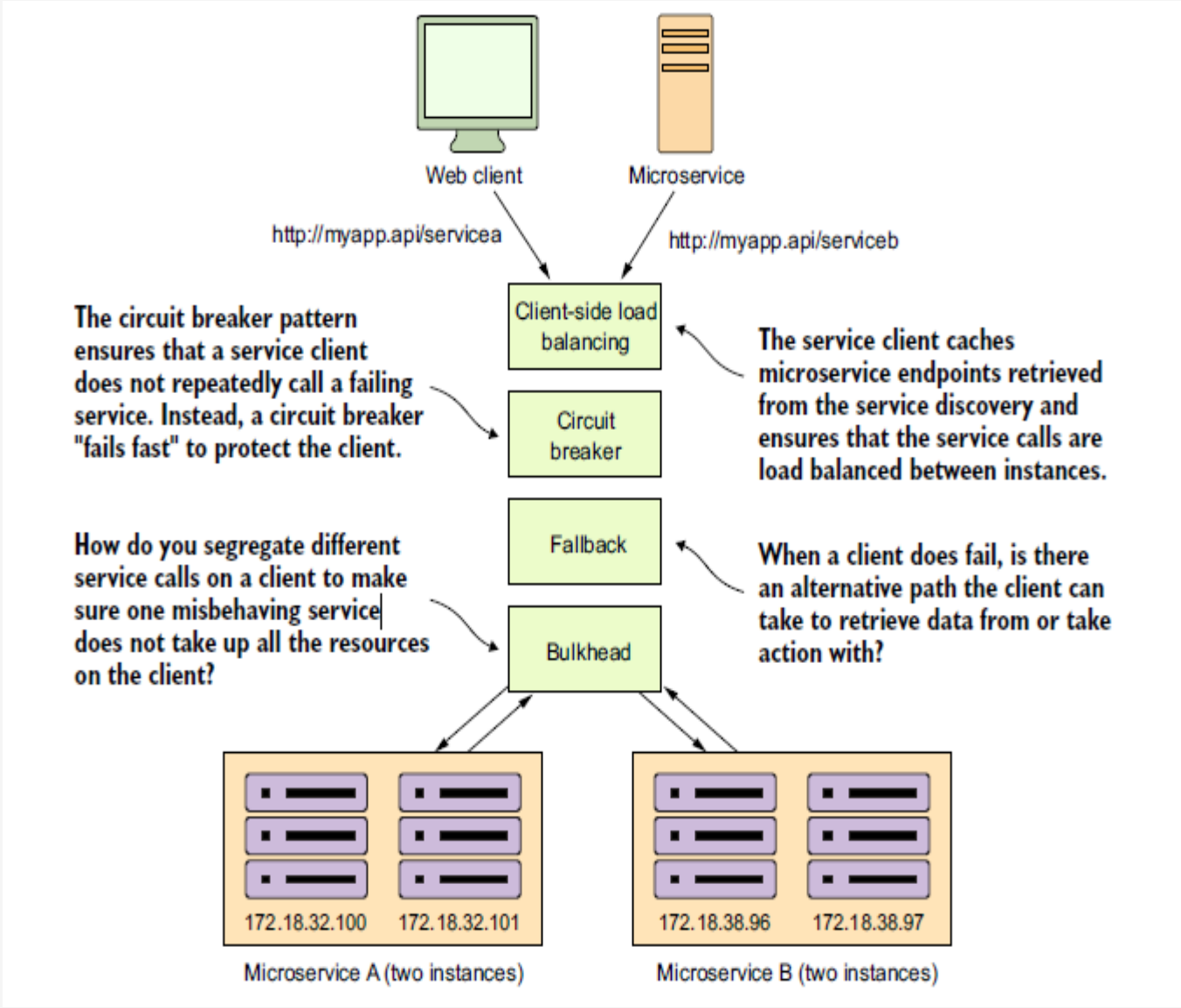
Microservice client resiliency patterns

- **Circuit breakers pattern** — How do you prevent a client from continuing to call a service that's failing or suffering performance problems? When a service is running slowly, it consumes resources on the client calling it. You want failing microservice calls to fail fast so that the calling client can quickly respond and take an appropriate action.

- **Fallback pattern** — When a service call fails, how do you provide a "plug-in" mechanism that will allow the service client to try to carry out its work through alternative means other than the microservice being called?

- **Bulkhead pattern** — Microservice applications use multiple distributed resources to carry out their work. How do you compartmentalize these calls so that the misbehavior of one service call doesn't negatively impact the rest of the application?

Microservice client resiliency patterns



Microservice security patterns

- **Authentication**—How do you determine the service client calling the service is who they say they are?

- **Authorization**—How do you determine whether the service client calling a microservice is allowed to undertake the action they're trying to undertake?

- **Credential management and propagation**—How do you prevent a service client from constantly having to present their credentials for service calls involved in a transaction?

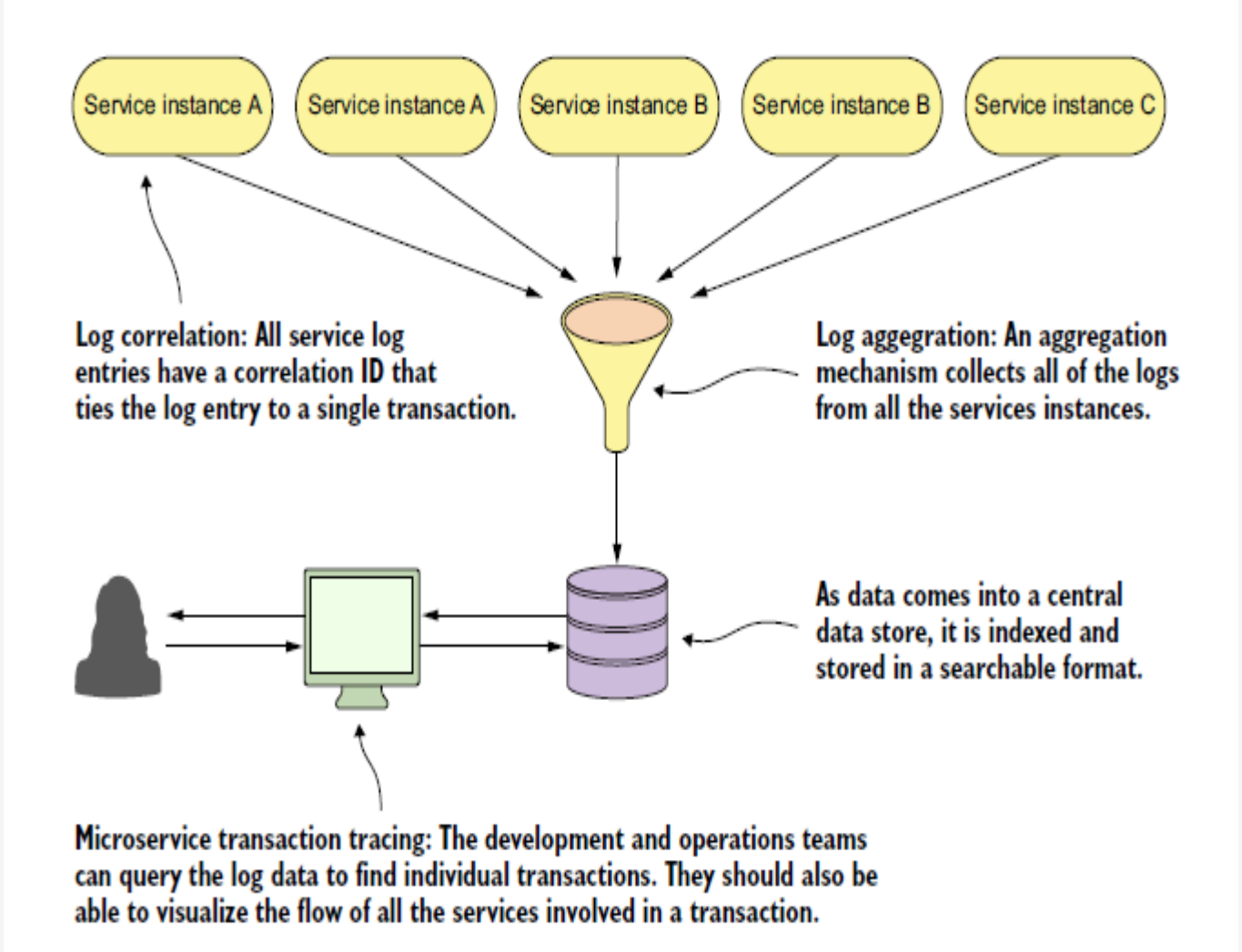
Microservice logging and tracing patterns

- The beauty of the microservice architecture is that a Microservice application is broken down into small pieces of functionality that can be deployed independently of one another. The downside of a microservice architecture is that it's much more difficult to debug and trace

- **Log correlation**—How do you tie together all the logs produced between services for a single user transaction? With this pattern, we'll look at how to implement a correlation ID, which is a unique identifier that will be carried across all service calls in a transaction and can be used to tie together log entries produced from each service.

- **Log aggregation**—With this pattern we'll look at how to pull together all of the logs produced by your microservices (and their individual instances) into a single queryable database.

Microservice logging and tracing patterns



Spring Cloud and Microservices

- Implementing all these patterns from scratch would be a tremendous amount of work. Fortunately for us, the Spring team has integrated a wide number of battletested open source projects into a Spring subproject collectively known as Spring Cloud.

- Spring Cloud wraps the work of open source companies such as Pivotal, HashiCorp, and Netflix in delivering patterns. Spring Cloud simplifies setting up and configuring of these projects into your Spring application so that you can focus on writing code, not getting involved in the details of configuring all the infrastructure that can go with building and deploying a microservice application.

Spring Cloud and Microservices

MicroServices by Pratap Kumar