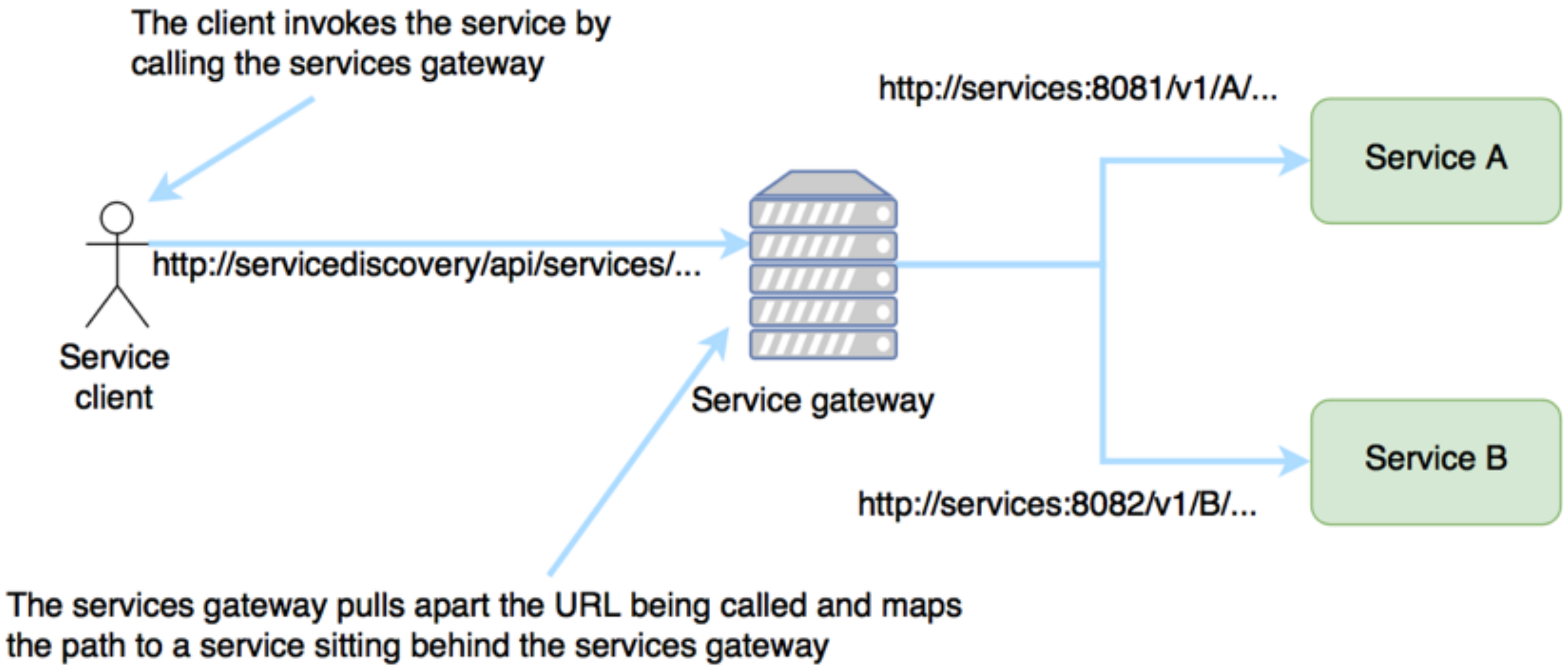
Service Routing with Spring Cloud and Zuul

Chapter Content

- 1. Service Gateway
- 2. Filters

1. Service Gateway

- A service gateway acts as an intermediary between the service client and a service being invoked
- Service clients only talk to a **single** URL managed by service gateway
- A service gateway sits between all calls from the client to the individual services, it also acts as a central Policy Enforcement Point (PEP) for service calls


The client invokes the service by calling the services gateway
http://services:8081/v1/A/...
Service A
http://servicediscovery/api/services/...
Service client
Service gateway
Service B
http://services:8082/v1/B/...
The services gateway pulls apart the URL being called and maps the path to a service sitting behind the services gateway

- Examples of cross-cutting concerns that can be implemented in a service gateway include:
    - **Static routing** - A service gateway places all service calls behind a single URL and API route
    - **Dynamic routing** - A service gateway can inspect incoming service requests and, based on data from the incoming request, perform intelligent routing based on who the service caller is
    - **Authentication and Authorization** - All service calls route through a service gateway, it can then check the caller's authenticated and authorized
    - **Metric collection and logging** - A service gateway can be used to collect metrics and log information as a service call passes through the service gateway

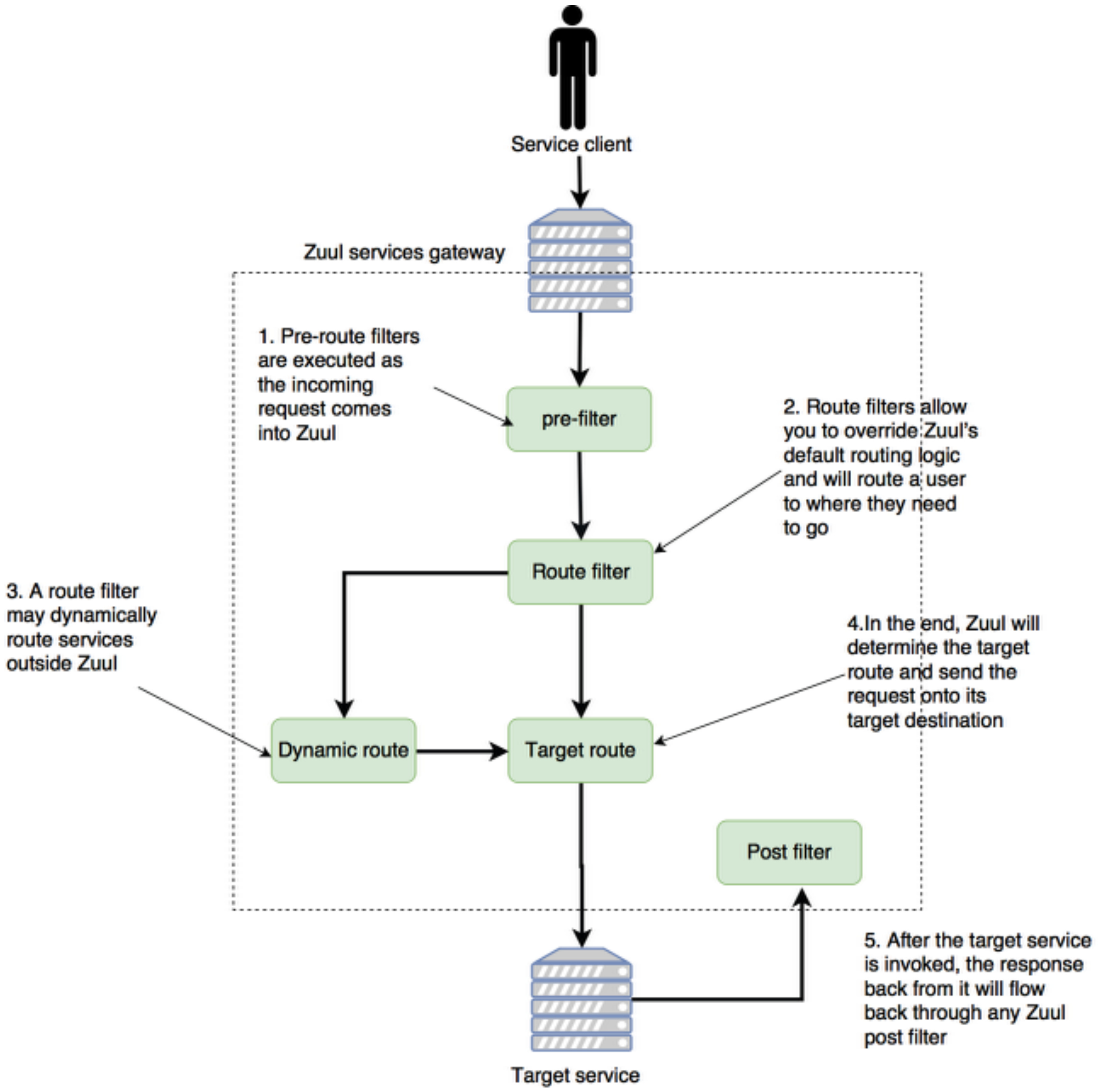
- Netflix open source project Zuul is a service gateway that’s easy to set up
- Zuul provides:
  - Mapping the routes for all the services in your application to a single URL
  - Building filters that can inspect and act on the requests coming through the gateway

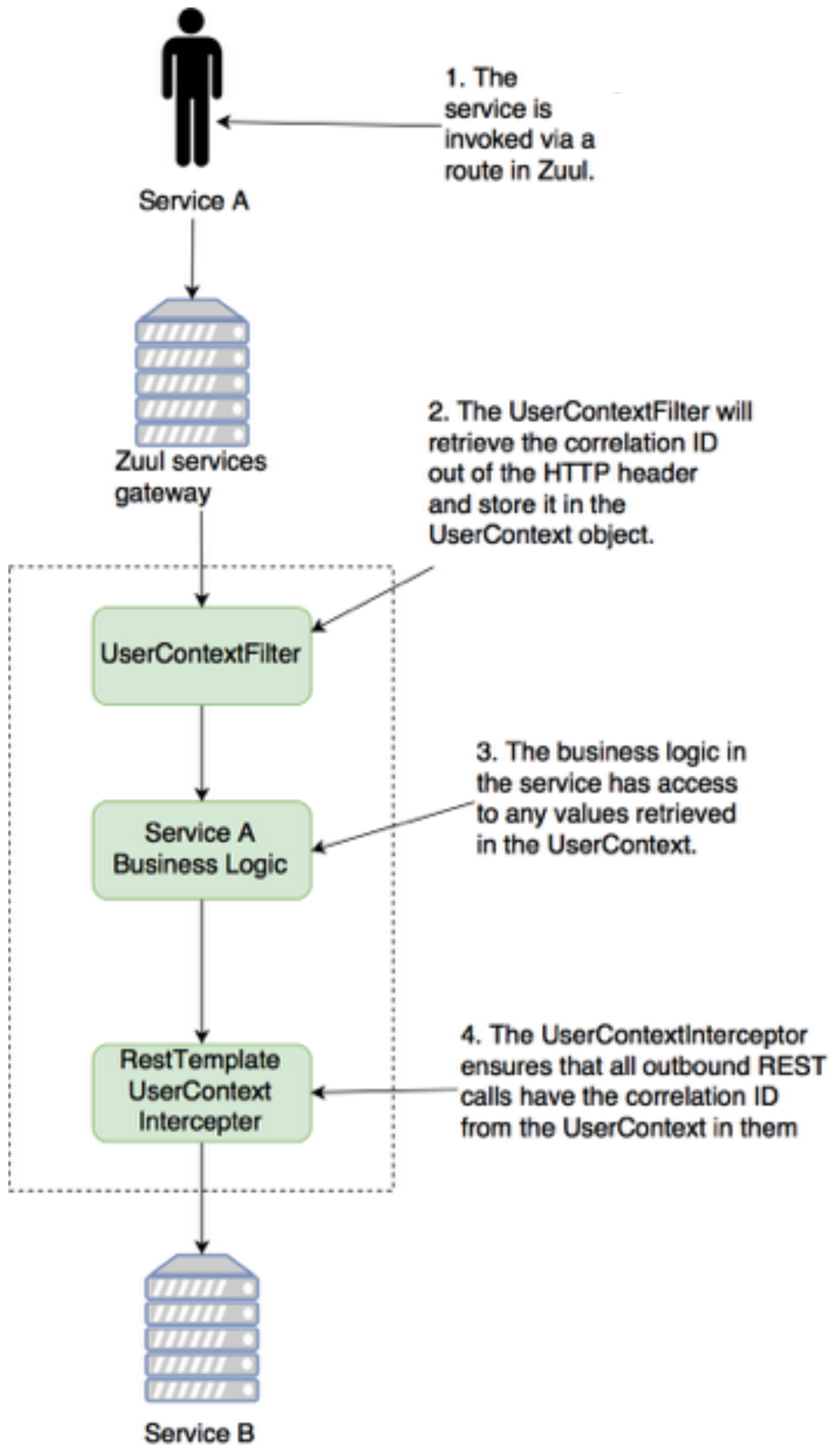
- Configuring Zuul to communicate with Eureka
  - The Zuul proxy server is designed by default to work on the Spring products. As such, **Zuul** will **automatically** use **Eureka** to look up services by their service IDs and then use Netflix **Ribbon** to do client-side load balancing of requests from within Zuul
- Zuul in nature is a **reverse proxy**. A reverse proxy is an intermediate server that sits between the client trying to reach a resource and the resource itself. The client has no idea it's even communicating to a server other than a proxy. The reverse proxy takes care of capturing the client's request and then calls the remote resource on the client's behalf
- In case of a microservice, Zuul takes a microservice call from a client and forwards it onto the downstream service.

2. Filters

- The real power of Zuul comes into play when you want to write custom logic that will be applied against all the services calls flowing through the gateway.
- Most often, this custom logic is used to enforce a consistent set of application policies like security, logging, and tracking against all the services

- Zuul supports three types of filters:
  - Pre-filter - invoked before the actual request to the target destination. Check for HTTP headers, authority
  - Post filters - invoked after the target service has been invoked and a response is sent being sent back to the client
  - Route filters - used to intercept the call before the target service is invoked.


Service client
Zuul services gateway
1. Pre-route filters are executed as the incoming request comes into Zuul
pre-filter
2. Route filters allow you to override Zuul's default routing logic and will route a user to where they need to go
Route filter
3. A route filter may dynamically route services outside Zuul
4.In the end, Zuul will determine the target route and send the request onto its target destination
Dynamic route
Target route
Post filter
5. After the target service is invoked, the response back from it will flow back through any Zuul post filter
Target service


1. The service is invoked via a route in Zuul.
Service A
Zuul services gateway
2. The UserContextFilter will retrieve the correlation ID out of the HTTP header and store it in the UserContext object.
UserContextFilter
3. The business logic in the service has access to any values retrieved in the UserContext.
Service A Business Logic
4. The UserContextInterceptor ensures that all outbound REST calls have the correlation ID from the UserContext in them
RestTemplate UserContext Intercepter
Service B