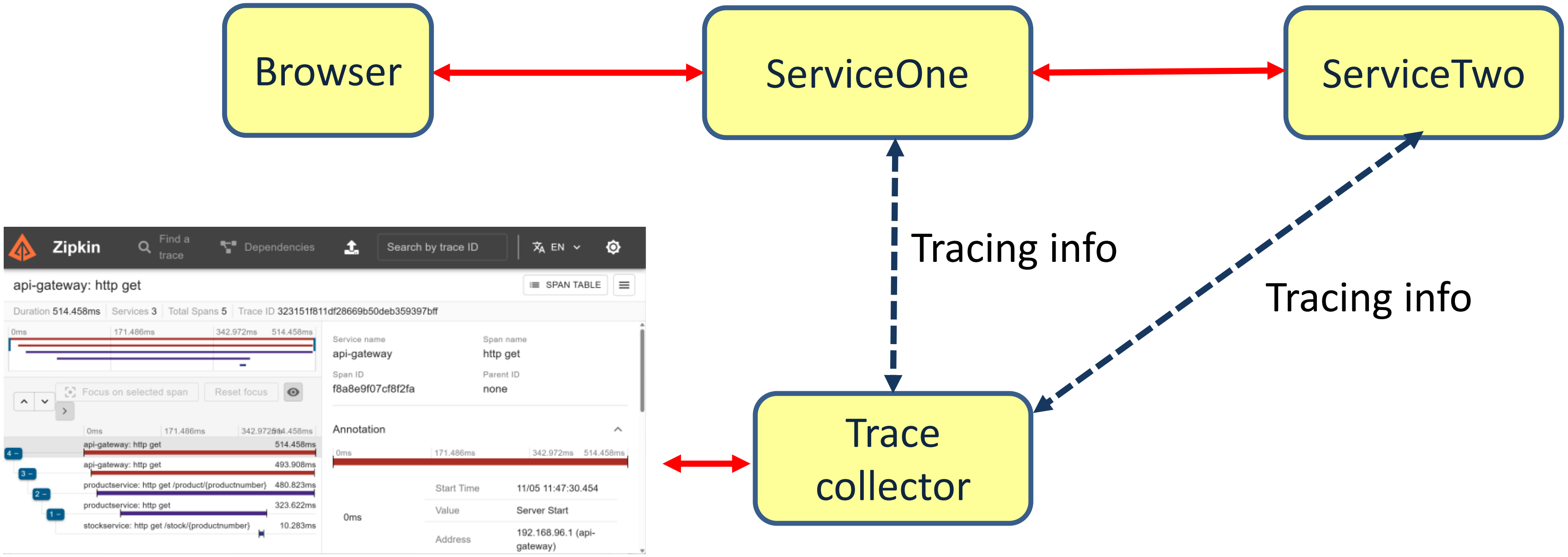
Lesson 8

MICROSERVICES

DISTRIBUTED TRACING: ZIPKIN

Distributed Tracing

- One central place where one can see the end-to-end tracing of all communication between services



api-gateway: http get

Duration 514.458ms | Services 3 | Total Spans 5 | Trace ID 323151f811df28669b50deb359397bff

```
0ms          171.486ms          342.972ms   514.458ms
api-gateway: http get                                  514.458ms
api-gateway: http get                                  493.908ms
productservice: http get /product/{productnumber}      480.823ms
productservice: http get                               323.622ms
stockservice: http get /stock/{productnumber}           10.283ms
```

Service name: api-gateway
Span name: http get
Span ID: f8a8e9f07cf8f2fa
Parent ID: none

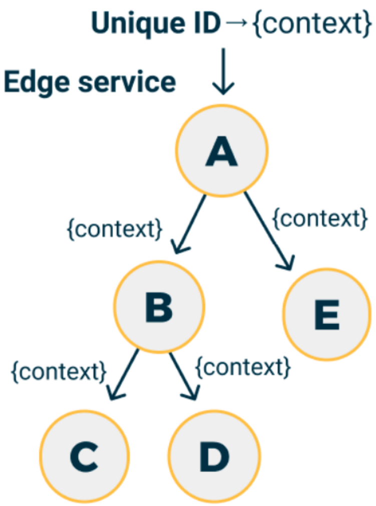
Annotation

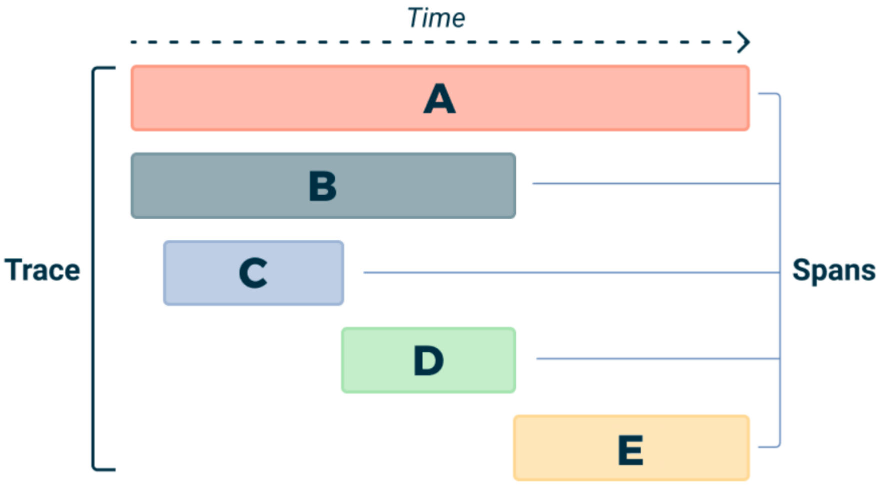
0ms 171.486ms 342.972ms 514.458ms

0ms	Start Time	11/05 11:47:30.454
	Value	Server Start
	Address	192.168.96.1 (api-gateway)

Micrometer

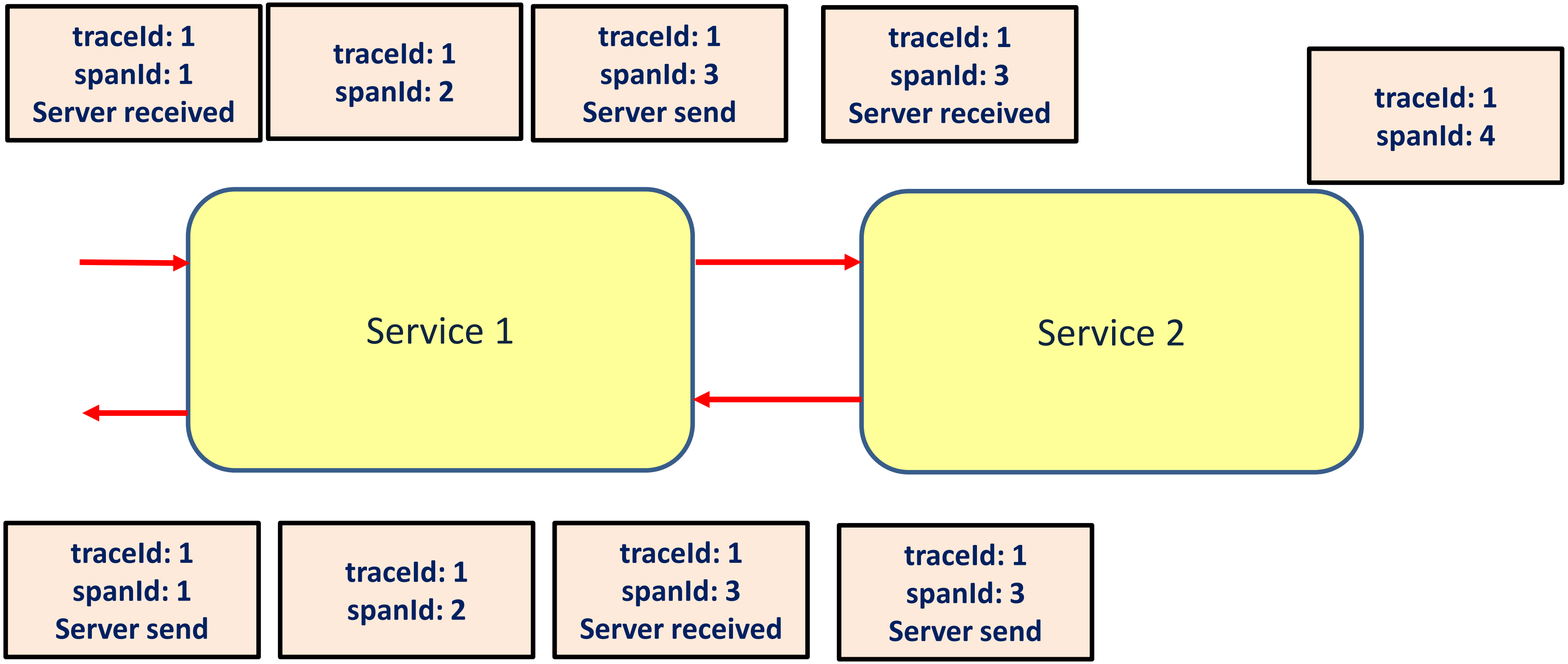
- Adds unique id’s to a request so we can trace the request
  - Span id: id for an individual operation
  - Trace id: id for a set of spans





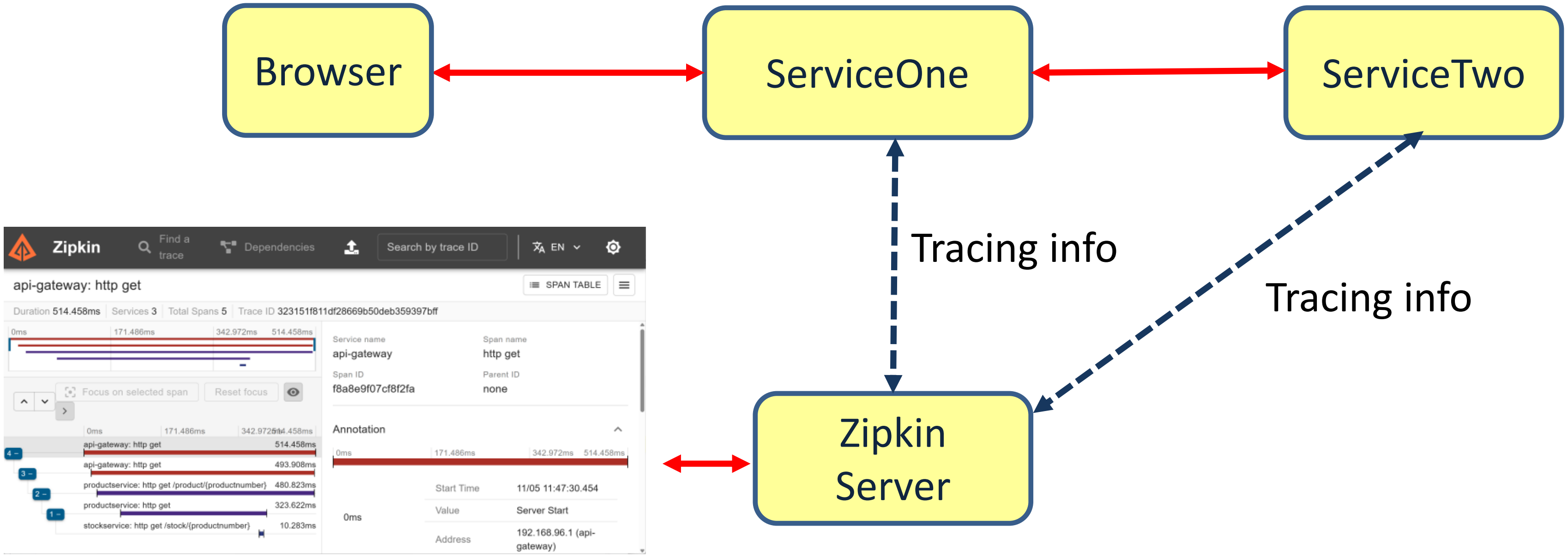
Span and Trace id’s

- Span: an individual operation
- Trace: a set of spans



Zipkin

- Centralized tracing server
  - Collects tracing information
- Zipkin console shows the data



api-gateway: http get

Duration 514.458ms | Services 3 | Total Spans 5 | Trace ID 323151f811df28669b50deb359397bff

0ms 171.486ms 342.972ms 514.458ms

4 –	api-gateway: http get	514.458ms
3 –	api-gateway: http get	493.908ms
2 –	productservice: http get /product/{productnumber}	480.823ms
1 –	productservice: http get	323.622ms
	stockservice: http get /stock/{productnumber}	10.283ms

Service name
api-gateway

Span name
http get

Span ID
f8a8e9f07cf8f2fa

Parent ID
none

Annotation

0ms 171.486ms 342.972ms 514.458ms

0ms	Start Time	11/05 11:47:30.454
	Value	Server Start
	Address	192.168.96.1 (api-gateway)

ServiceOne

```
@SpringBootApplication
@EnableFeignClients
@EnableDiscoveryClient
public class ServiceOneApplication {

    public static void main(String[] args) {
      SpringApplication.run(ServiceOneApplication.class, args);
    }

}
```

ServiceOne

```java
@RestController
public class ServiceOneController {

   @Autowired
   ServiceTwoClient serviceTwoClient;

   @RequestMapping("/text")
   public String getText() {
      String service2Text = serviceTwoClient.getText();
      return "Hello "+ service2Text;
   }

   @FeignClient("ServiceTwo")
   interface ServiceTwoClient {
      @RequestMapping("/text")
      public String getText();
   }
}
```

ServiceOne

application.yml

```yaml
server:
  port: 9093

spring:
  application:
    name: ServiceOne
  cloud:
    consul:
      host: localhost
      port: 8500
      discovery:
        enabled: true
        prefer-ip-address: true
        instance-id: ${spring.application.name}:${random.value}


otlp:
  tracing:
    endpoint: http://localhost:9411/api/v2/spans  # Zipkin endpoint

management:
  tracing:
    sampling:
      probability: 1.0
```

probability =1.0 means send tracing info for every call

Typically set probability to 0.1 which sends trace info only for 10% of the calls

ServiceOne

pom.xml

```xml
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-consul-discovery</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-otel</artifactId>
  </dependency>
  <dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-micrometer</artifactId>
    <version>13.6</version>
  </dependency>
  <dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-zipkin</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
  </dependency>
</dependencies>
```

ServiceTwo

```
@SpringBootApplication
@EnableDiscoveryClient
public class ServiceTwoApplication {

   public static void main(String[] args) {
     SpringApplication.run(ServiceTwoApplication.class, args);
   }
}
```

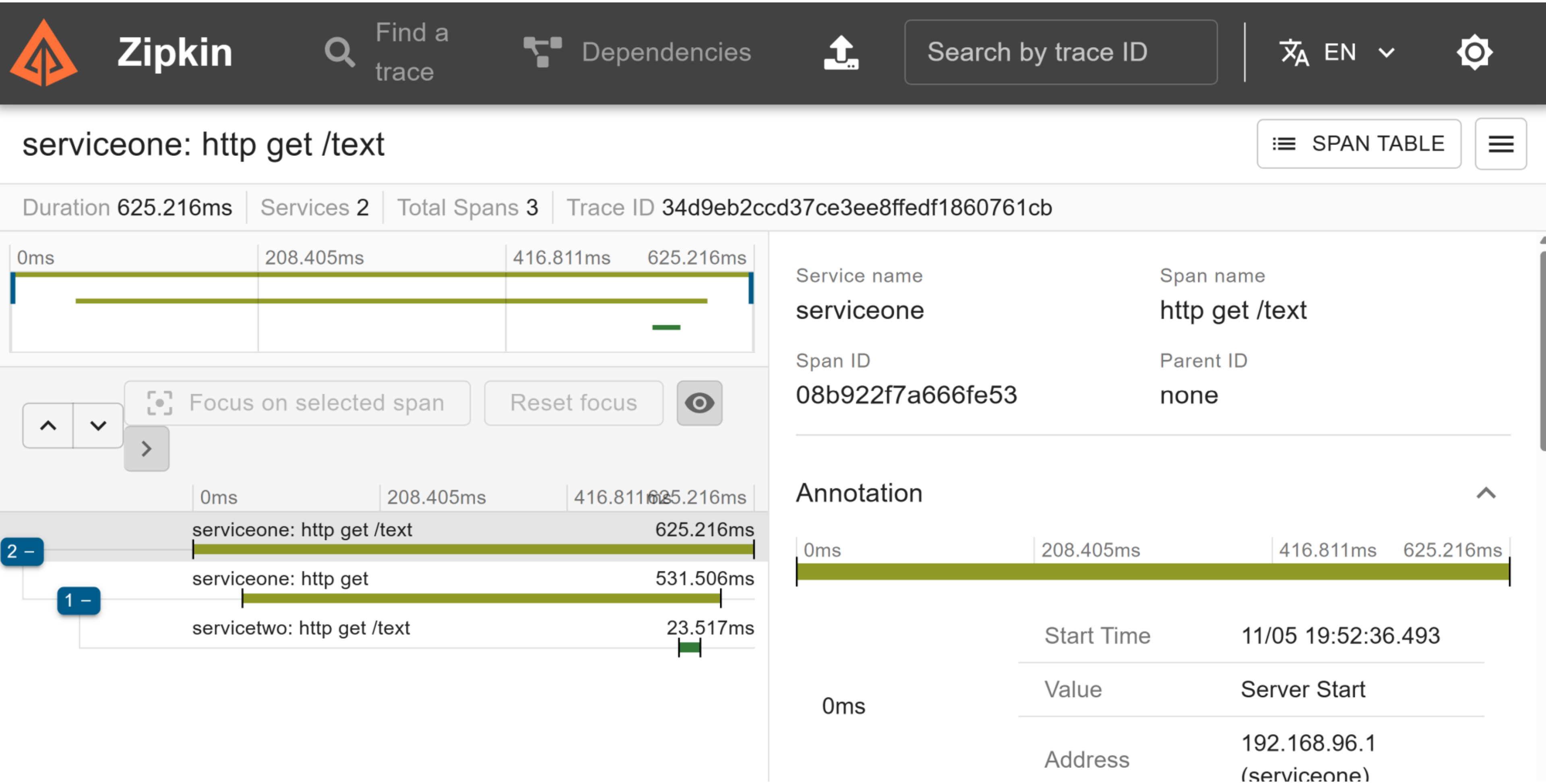
```
@RestController
public class ServiceTwoController {
   @RequestMapping("/text")
   public String getText() {
      return "World";
   }
}
```

ServiceTwo

application.yml

```yaml
server:
  port: 9091

spring:
  application:
    name: ServiceTwo
  cloud:
    consul:
      host: localhost
      port: 8500
      discovery:
        enabled: true
        prefer-ip-address: true
        instance-id: ${spring.application.name}:${random.value}


otlp:
  tracing:
    endpoint: http://localhost:9411/api/v2/spans  # Zipkin endpoint

management:
  tracing:
    sampling:
      probability: 1.0
```

Zipkin console



serviceone: http get /text

Duration 625.216ms | Services 2 | Total Spans 3 | Trace ID 34d9eb2ccd37ce3ee8ffedf1860761cb

Span	Duration
serviceone: http get /text	625.216ms
serviceone: http get	531.506ms
servicetwo: http get /text	23.517ms

Service name
serviceone

Span name
http get /text

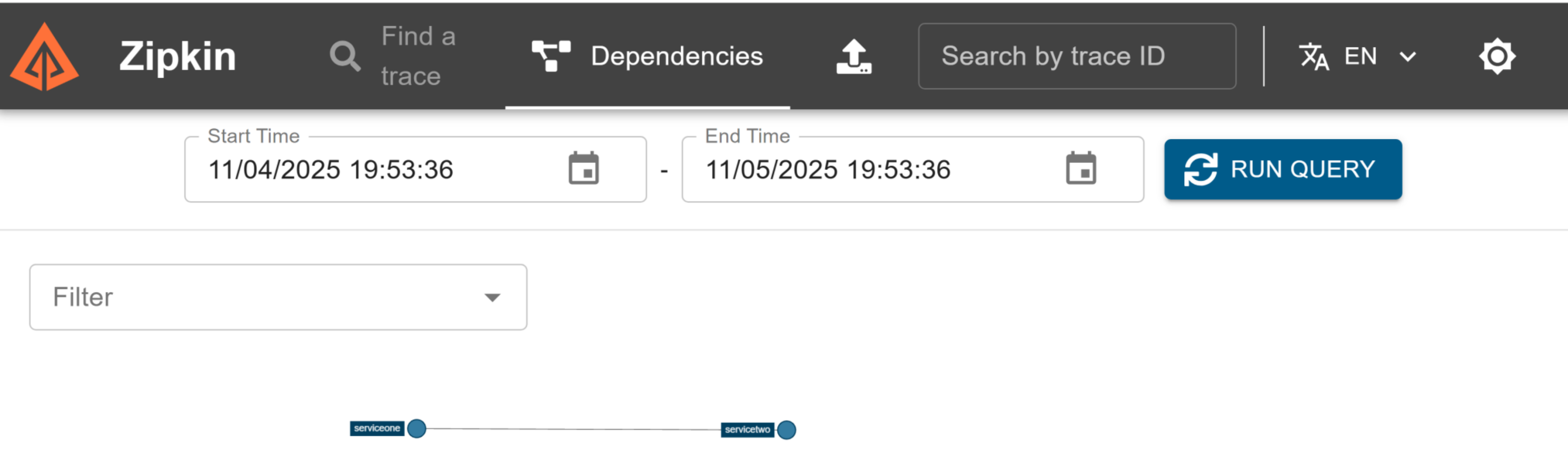
Span ID
08b922f7a666fe53

Parent ID
none

0ms

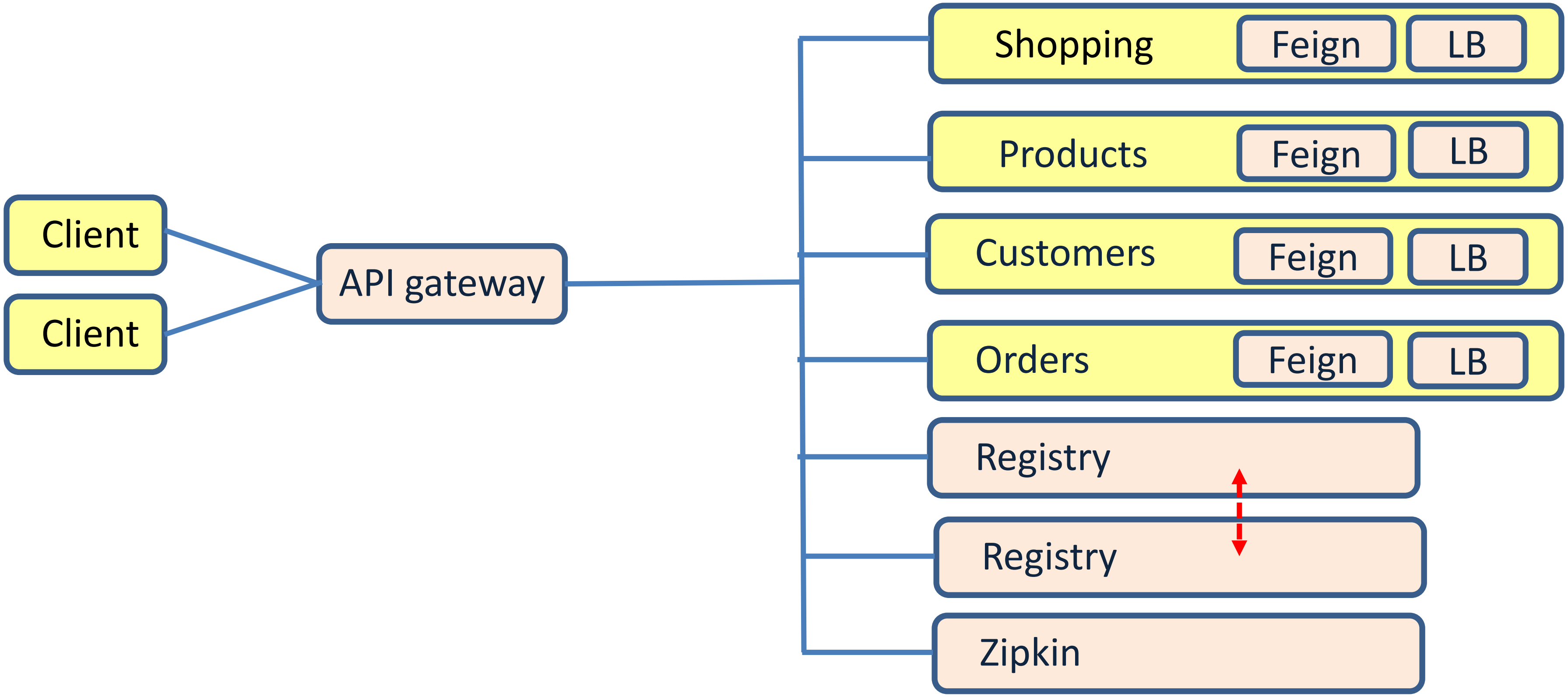
Start Time	11/05 19:52:36.493
Value	Server Start
Address	192.168.96.1 (serviceone)

Zipkin console



Start Time
11/04/2025 19:53:36
-
End Time
11/05/2025 19:53:36

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	
Follow/monitor business processes	Zipkin

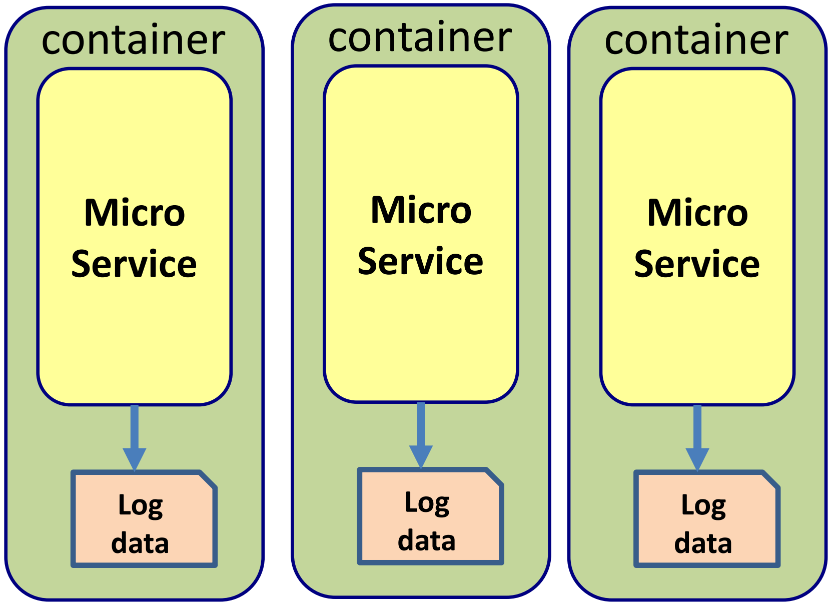
Main point

- We need zipkin in order to monitor and debug service-to-service communication

- The Unified Field is the field of perfection

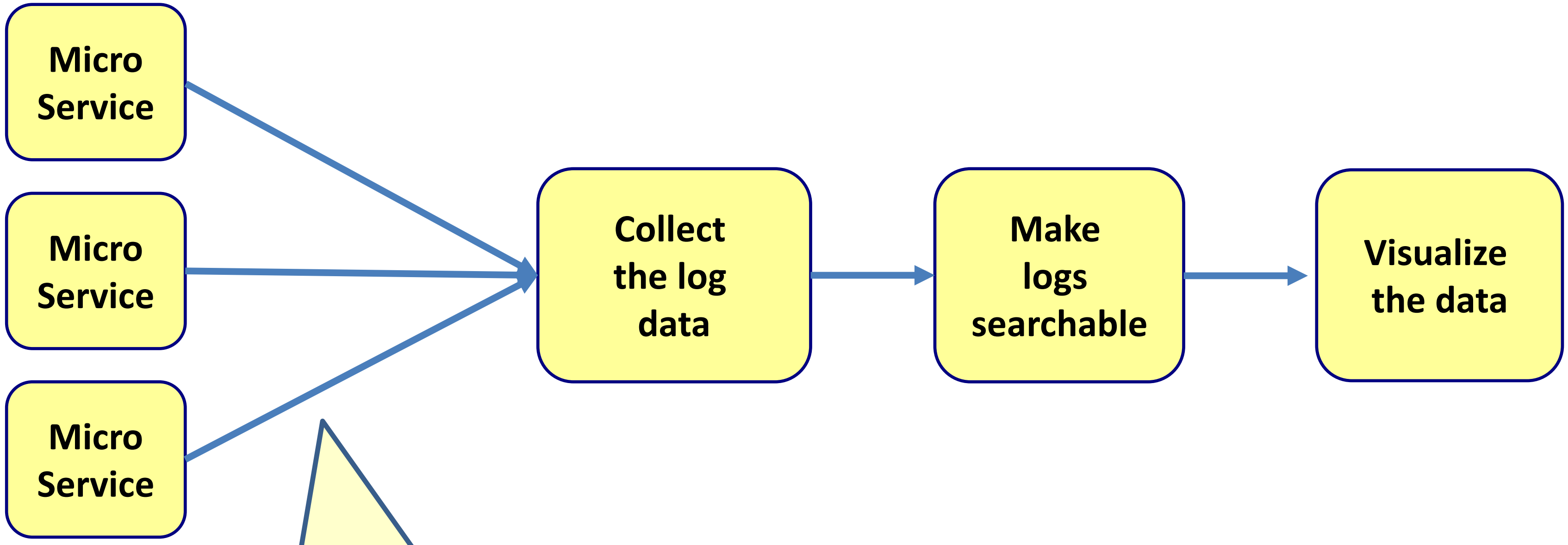
DISTRIBUTED LOGGING
ELK STACK

The need for centralized logging



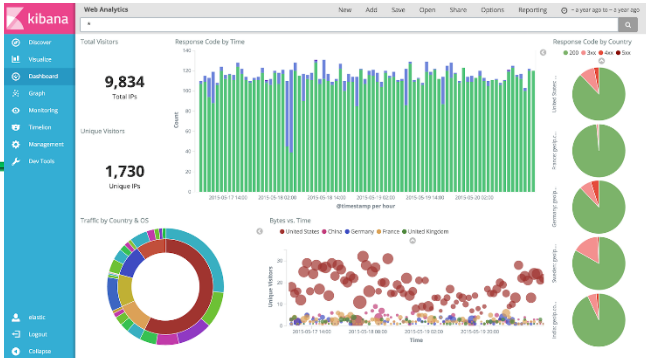
- Local logging does not work
  - Containers come and go
  - Containers have no fixed address

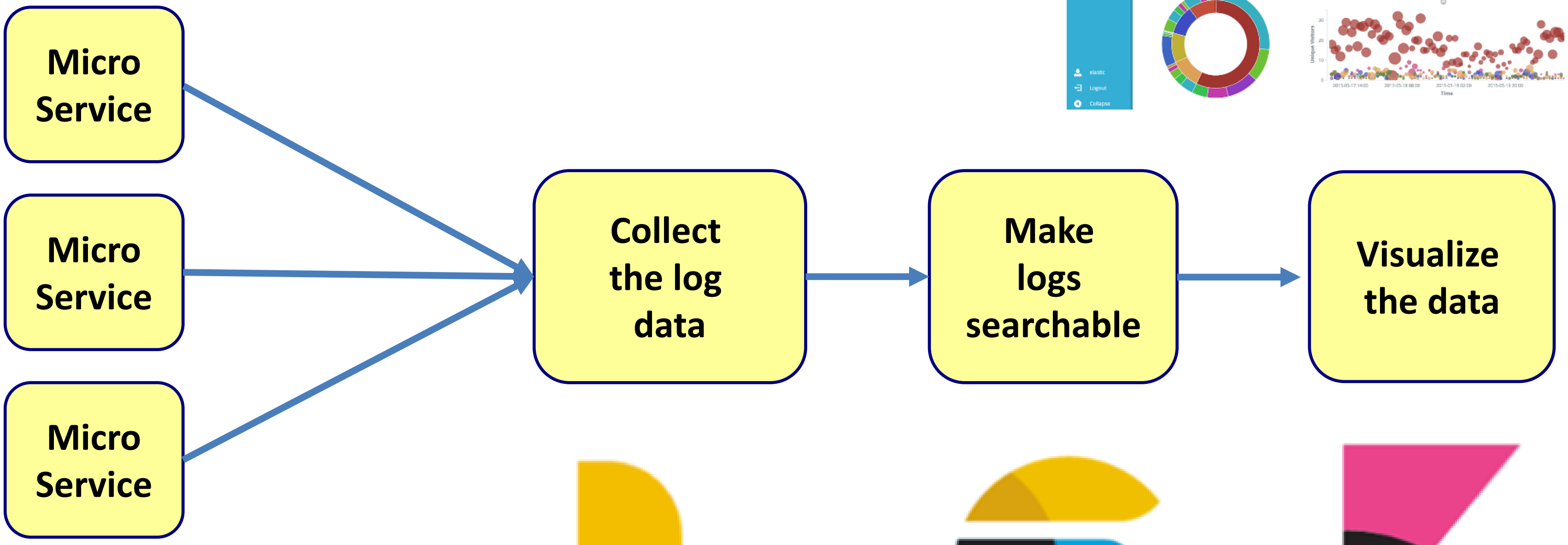
Microservice logging architecture



- Add unique ID to every log
- Dynamic logging
- Asynchronous logging

ELK stack





logstash

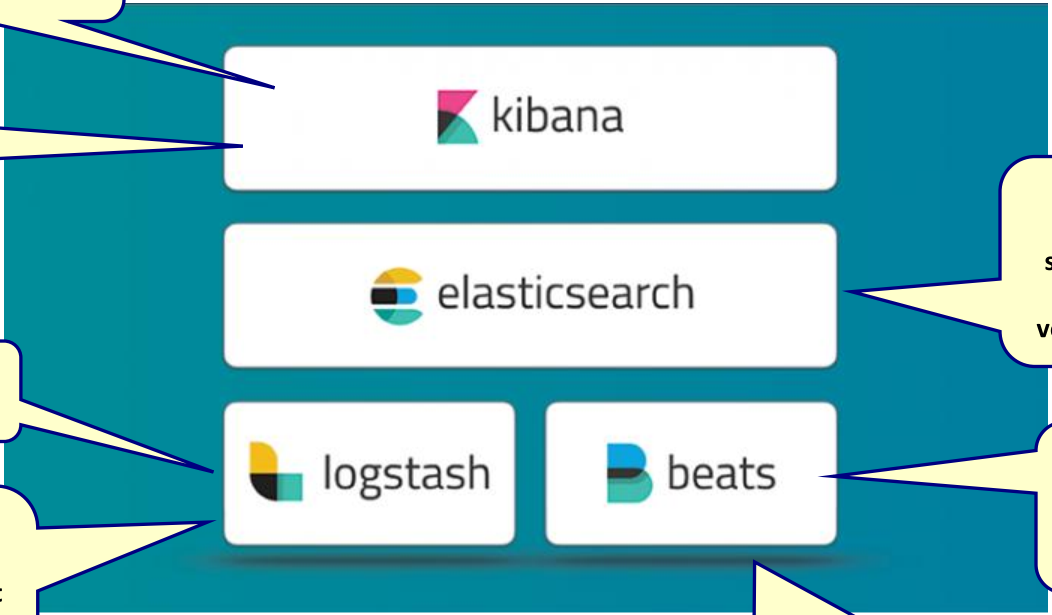
elasticsearch

kibana

Collect logs
Transform
Time normalization

Schema less search DB
Highly scalable
Fast searching

Elastic stack components



Analytics & visualization platform to visualize data in elasticsearch

Web based UI for elasticsearch

(dashboards)

Data store that allows you to store, search and analyze big volumes of data in near real time

Data processing pipeline

input->process->output

Takes log data (from different places), converts it to JSON and puts it into elasticsearch.

(server side)

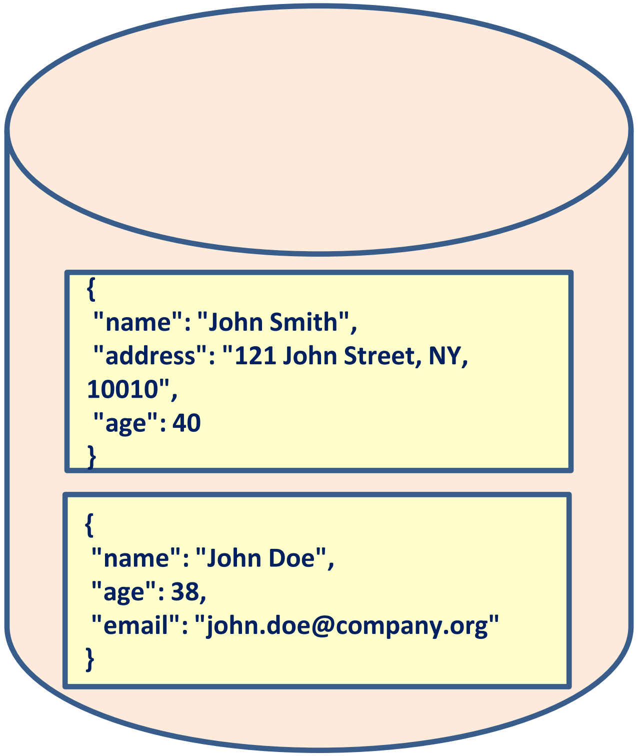
(Client side) Data shipper that sends data into elasticsearch

- Filebeat: collect log files
- Metricbeat: collect system metrics
- Packetbeat: collect network data

…

What is Elasicsearch?

- Database
  - Data is stored as JSON documents
- Full text search engine

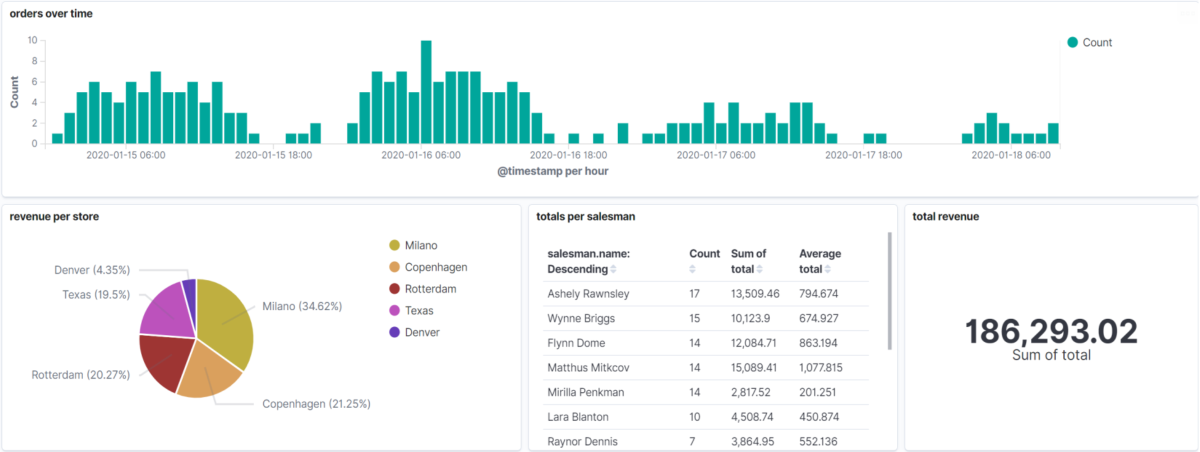


```
{
 "name": "John Smith",
 "address": "121 John Street, NY,
10010",
 "age": 40
}
```

```
{
 "name": "John Doe",
 "age": 38,
 "email": "john.doe@company.org"
}
```

Kibana

- Web UI on top of elasticsearch
- Has its own Kibana query language (KQL)
- Objects (Queries, visualizations, dashboards, etc.) are saved in elasticsearch

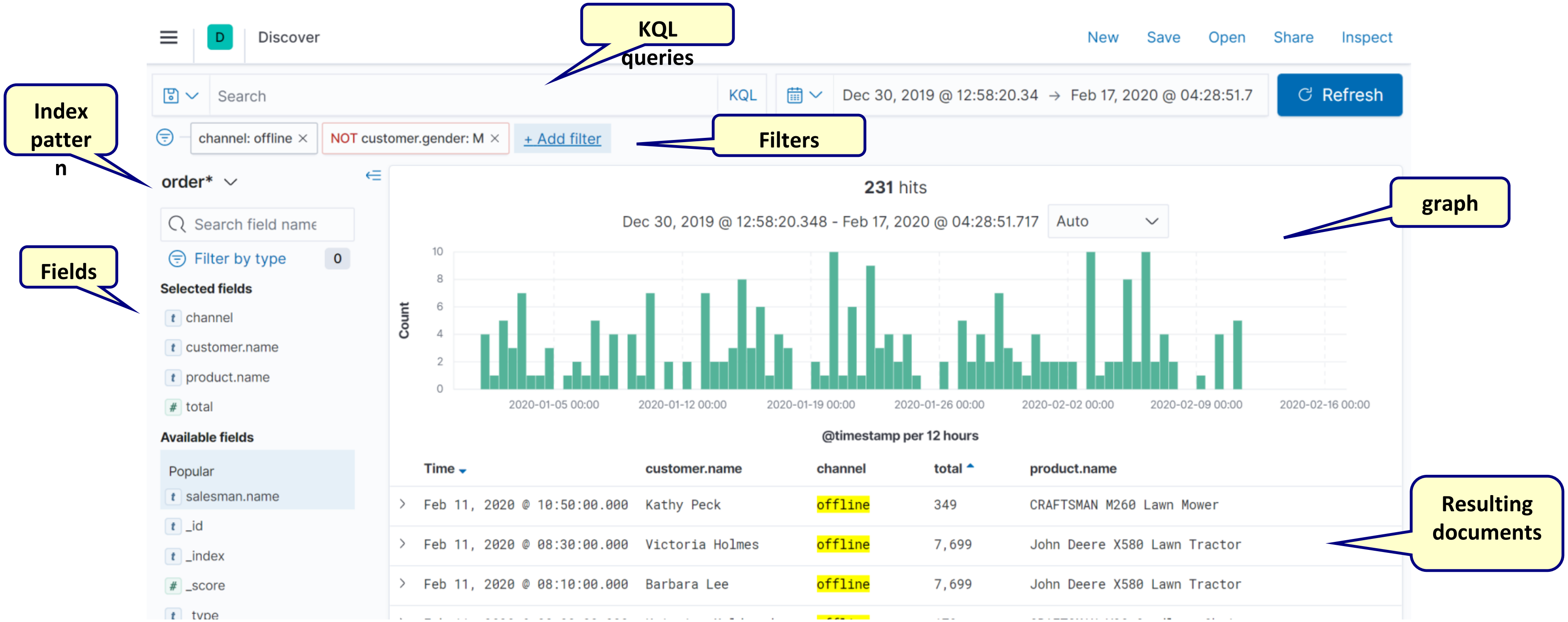


salesman.name: Descending	Count	Sum of total	Average total
Ashely Rawnsley	17	13,509.46	794.674
Wynne Briggs	15	10,123.9	674.927
Flynn Dome	14	12,084.71	863.194
Matthus Mitkcov	14	15,089.41	1,077.815
Mirilla Penkman	14	2,817.52	201.251
Lara Blanton	10	4,508.74	450.874
Raynor Dennis	7	3,864.95	552.136

186,293.02
Sum of total

Discover app

- Good for exploring and analyzing data



Index pattern

KQL queries

Filters

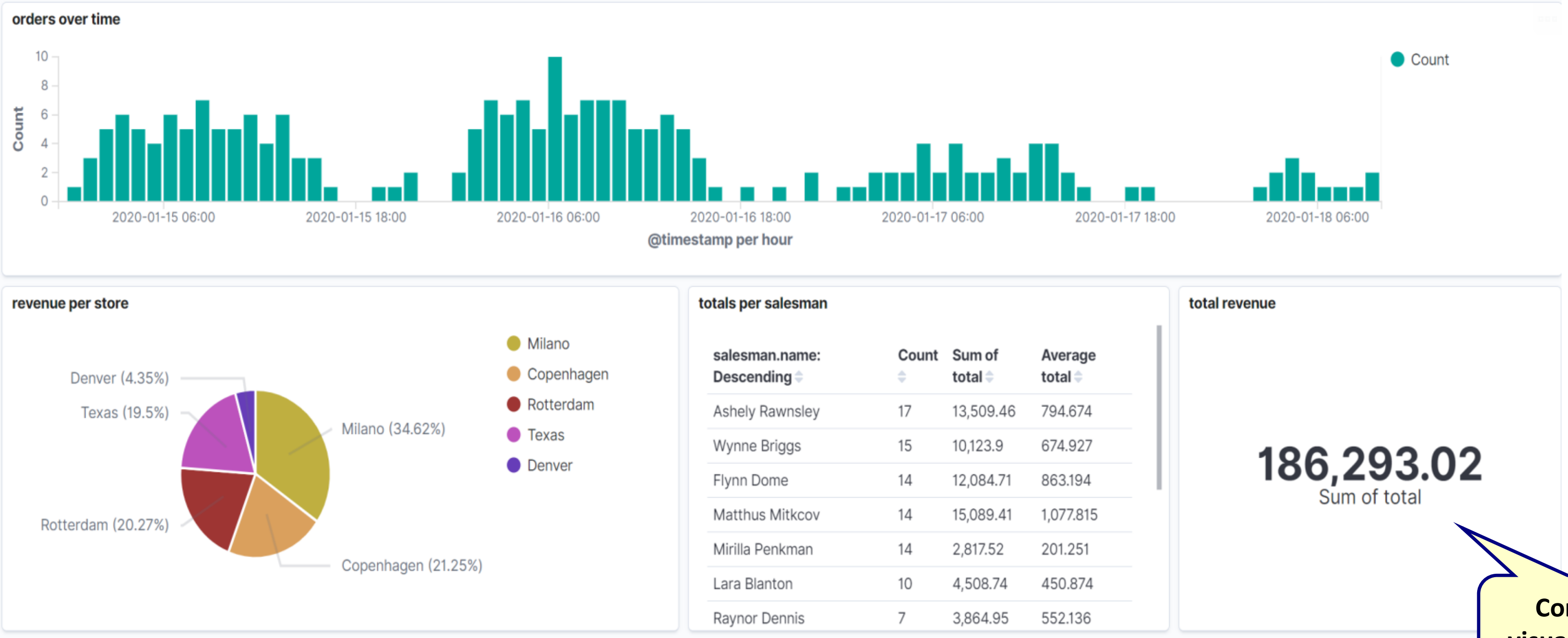
graph

Fields

Resulting documents

Time	customer.name	channel	total	product.name
Feb 11, 2020 @ 10:50:00.000	Kathy Peck	offline	349	CRAFTSMAN M260 Lawn Mower
Feb 11, 2020 @ 08:30:00.000	Victoria Holmes	offline	7,699	John Deere X580 Lawn Tractor
Feb 11, 2020 @ 08:10:00.000	Barbara Lee	offline	7,699	John Deere X580 Lawn Tractor

Dashboard



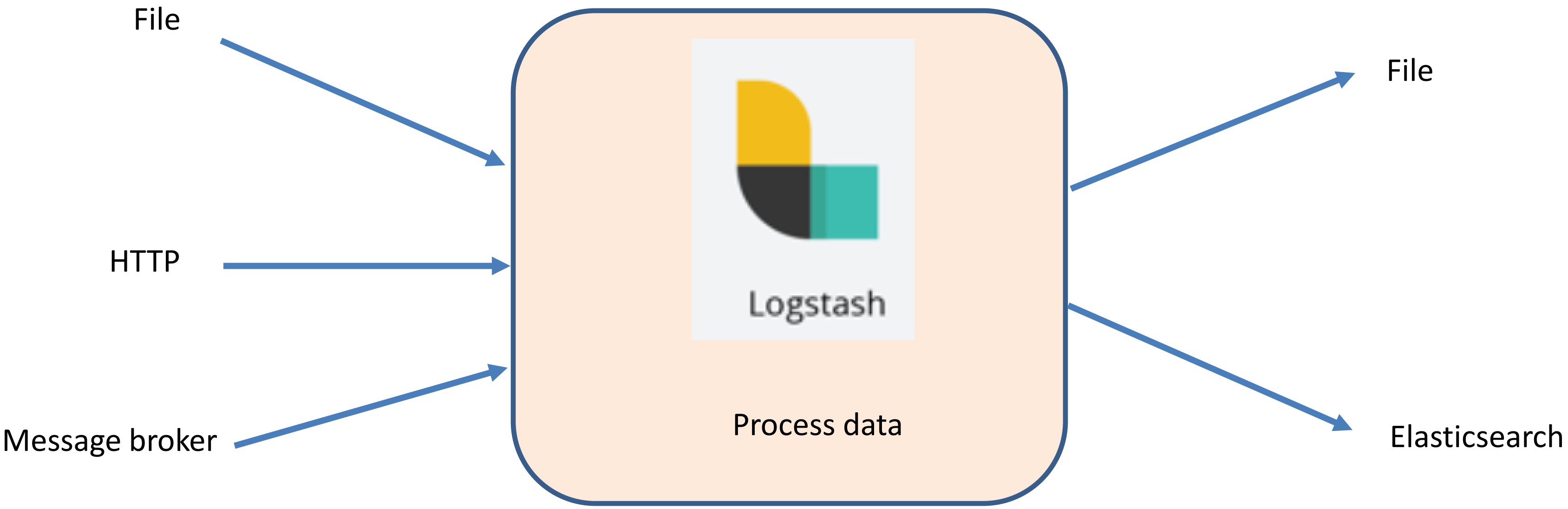
salesman.name: Descending	Count	Sum of total	Average total
Ashely Rawnsley	17	13,509.46	794.674
Wynne Briggs	15	10,123.9	674.927
Flynn Dome	14	12,084.71	863.194
Matthus Mitkcov	14	15,089.41	1,077.815
Mirilla Penkman	14	2,817.52	201.251
Lara Blanton	10	4,508.74	450.874
Raynor Dennis	7	3,864.95	552.136

186,293.02
Sum of total

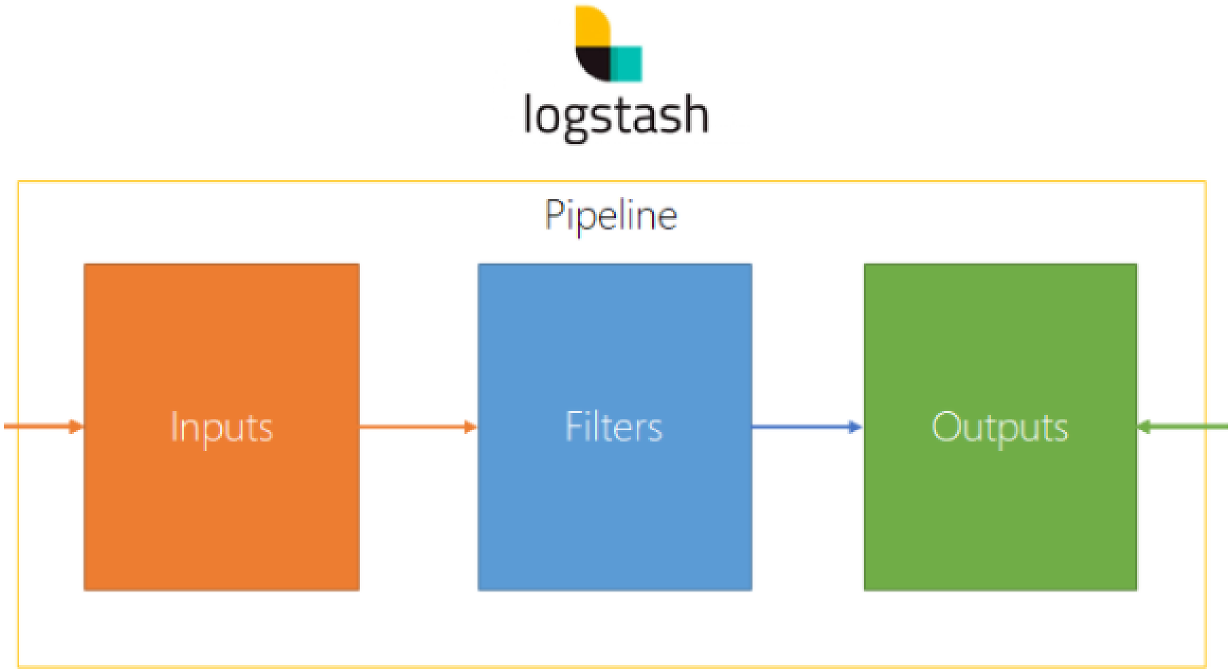
Contains visualizations

Logstash

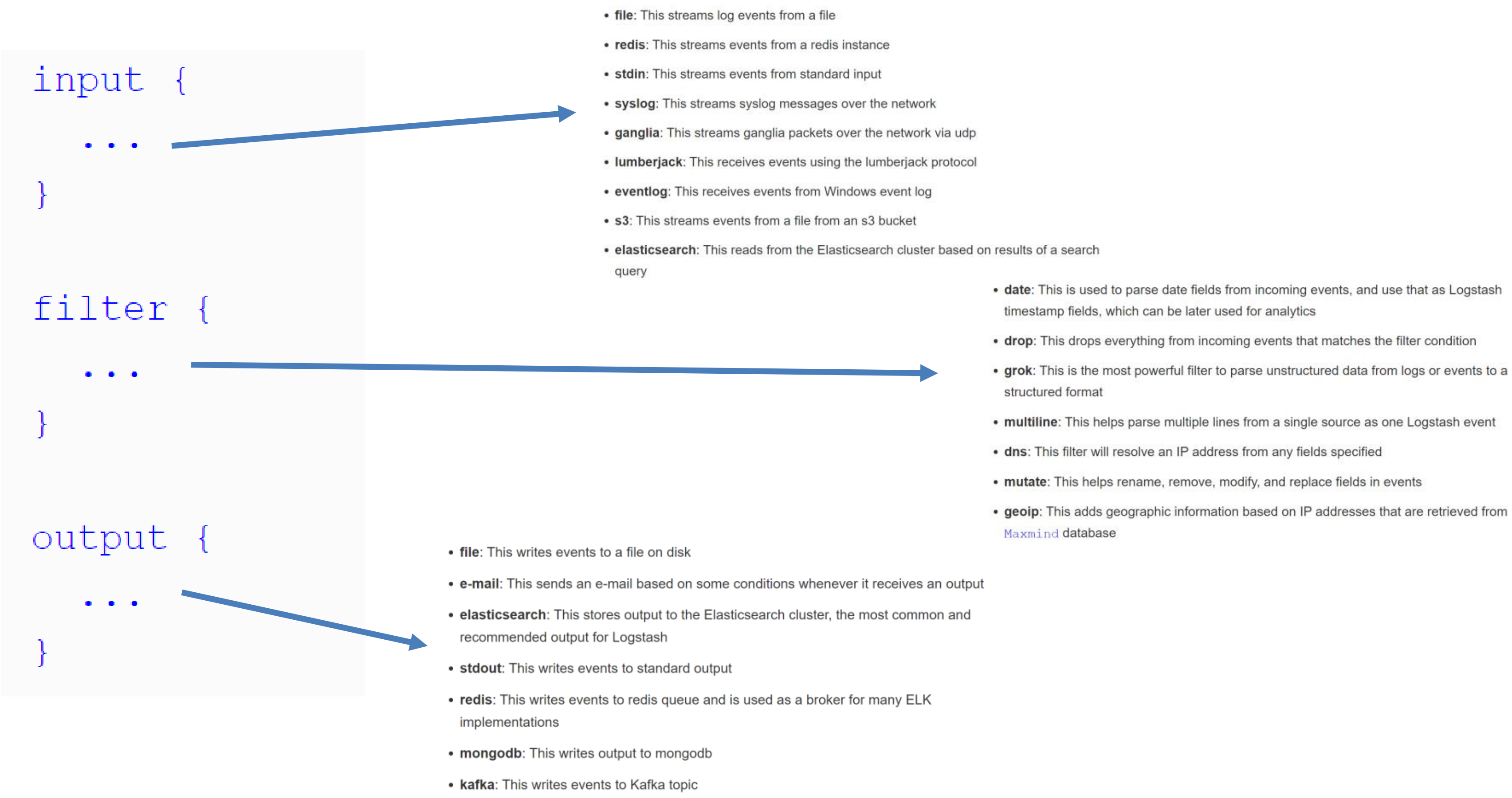
- Event processing engine



How does Logstash work?



Logstash configuration



```
input {
    ...
}

filter {
    ...
}

output {
    ...
}
```

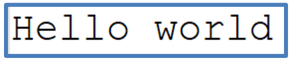
- **file**: This streams log events from a file
- **redis**: This streams events from a redis instance
- **stdin**: This streams events from standard input
- **syslog**: This streams syslog messages over the network
- **ganglia**: This streams ganglia packets over the network via udp
- **lumberjack**: This receives events using the lumberjack protocol
- **eventlog**: This receives events from Windows event log
- **s3**: This streams events from a file from an s3 bucket
- **elasticsearch**: This reads from the Elasticsearch cluster based on results of a search query

- **date**: This is used to parse date fields from incoming events, and use that as Logstash timestamp fields, which can be later used for analytics
- **drop**: This drops everything from incoming events that matches the filter condition
- **grok**: This is the most powerful filter to parse unstructured data from logs or events to a structured format
- **multiline**: This helps parse multiple lines from a single source as one Logstash event
- **dns**: This filter will resolve an IP address from any fields specified
- **mutate**: This helps rename, remove, modify, and replace fields in events
- **geoip**: This adds geographic information based on IP addresses that are retrieved from `Maxmind` database

- **file**: This writes events to a file on disk
- **e-mail**: This sends an e-mail based on some conditions whenever it receives an output
- **elasticsearch**: This stores output to the Elasticsearch cluster, the most common and recommended output for Logstash
- **stdout**: This writes events to standard output
- **redis**: This writes events to redis queue and is used as a broker for many ELK implementations
- **mongodb**: This writes output to mongodb
- **kafka**: This writes events to Kafka topic

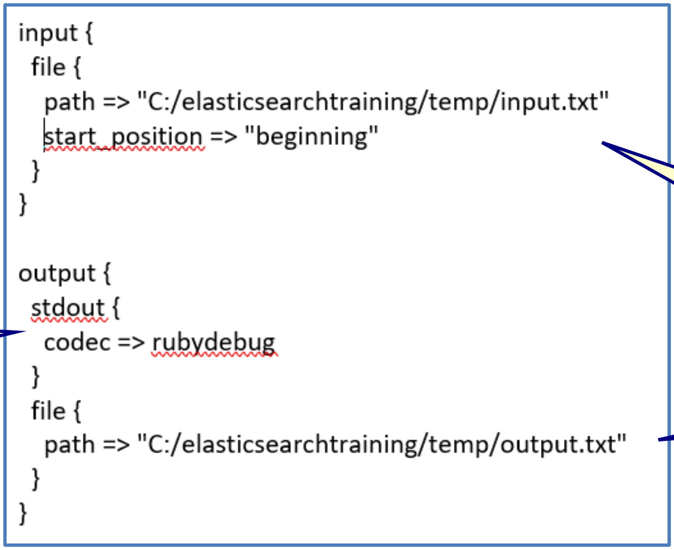
Logstash configuration

input.txt



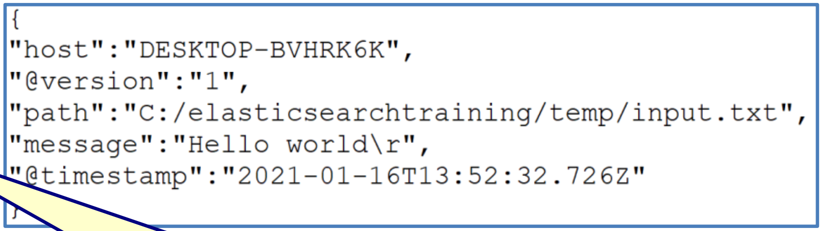
```
Hello world
```

pipeline.conf



```
input {
  file {
    path => "C:/elasticsearchtraining/temp/input.txt"
    start_position => "beginning"
  }
}

output {
  stdout {
    codec => rubydebug
  }
  file {
    path => "C:/elasticsearchtraining/temp/output.txt"
  }
}
```

output.txt



```
{
"host":"DESKTOP-BVHRK6K",
"@version":"1",
"path":"C:/elasticsearchtraining/temp/input.txt",
"message":"Hello world\r",
"@timestamp":"2021-01-16T13:52:32.726Z"
}
```

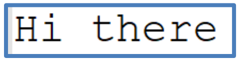
Anytime this file changes, read from this file

Write the output to the console

Write the output to the specified file

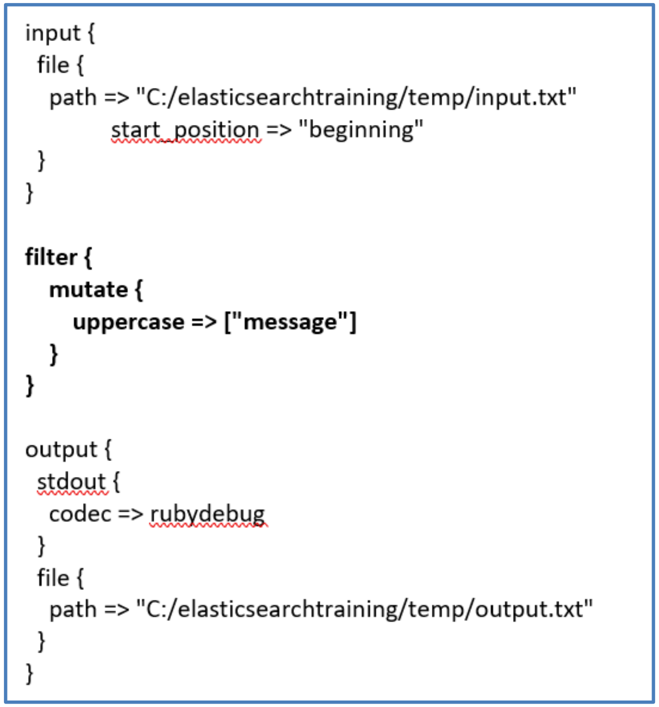
Logstash configuration

input.txt



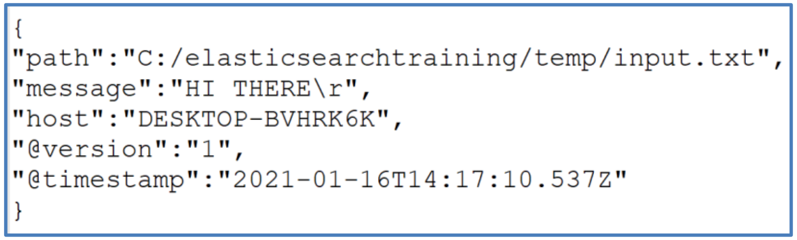
```
Hi there
```

pipeline.conf



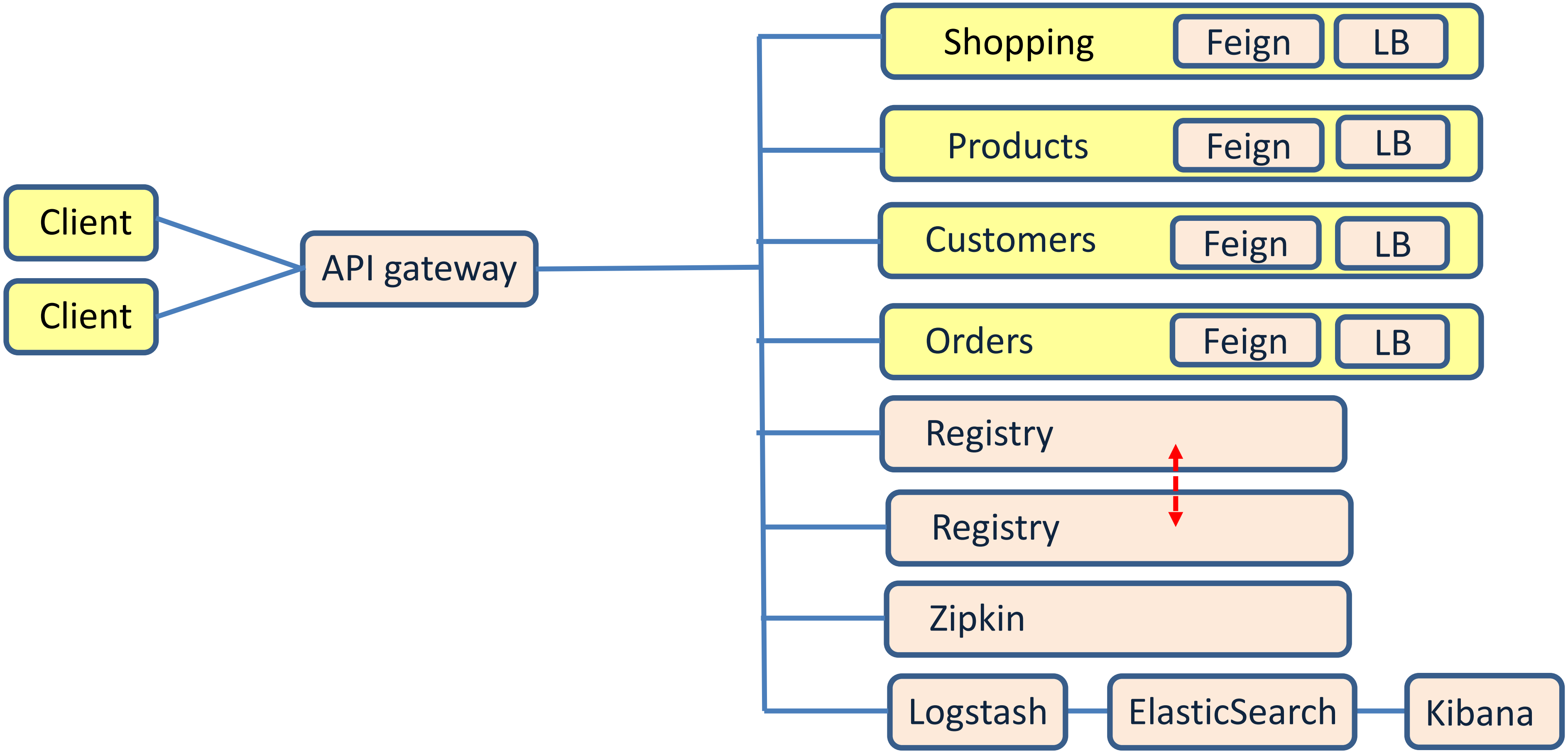
```
input {
 file {
  path => "C:/elasticsearchtraining/temp/input.txt"
       start_position => "beginning"
 }
}

filter {
  mutate {
    uppercase => ["message"]
  }
}

output {
 stdout {
  codec => rubydebug
 }
 file {
  path => "C:/elasticsearchtraining/temp/output.txt"
 }
}
```

output.txt



```
{
"path":"C:/elasticsearchtraining/temp/input.txt",
"message":"HI THERE\r",
"host":"DESKTOP-BVHRK6K",
"@version":"1",
"@timestamp":"2021-01-16T14:17:10.537Z"
}
```

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

RESILIENCE

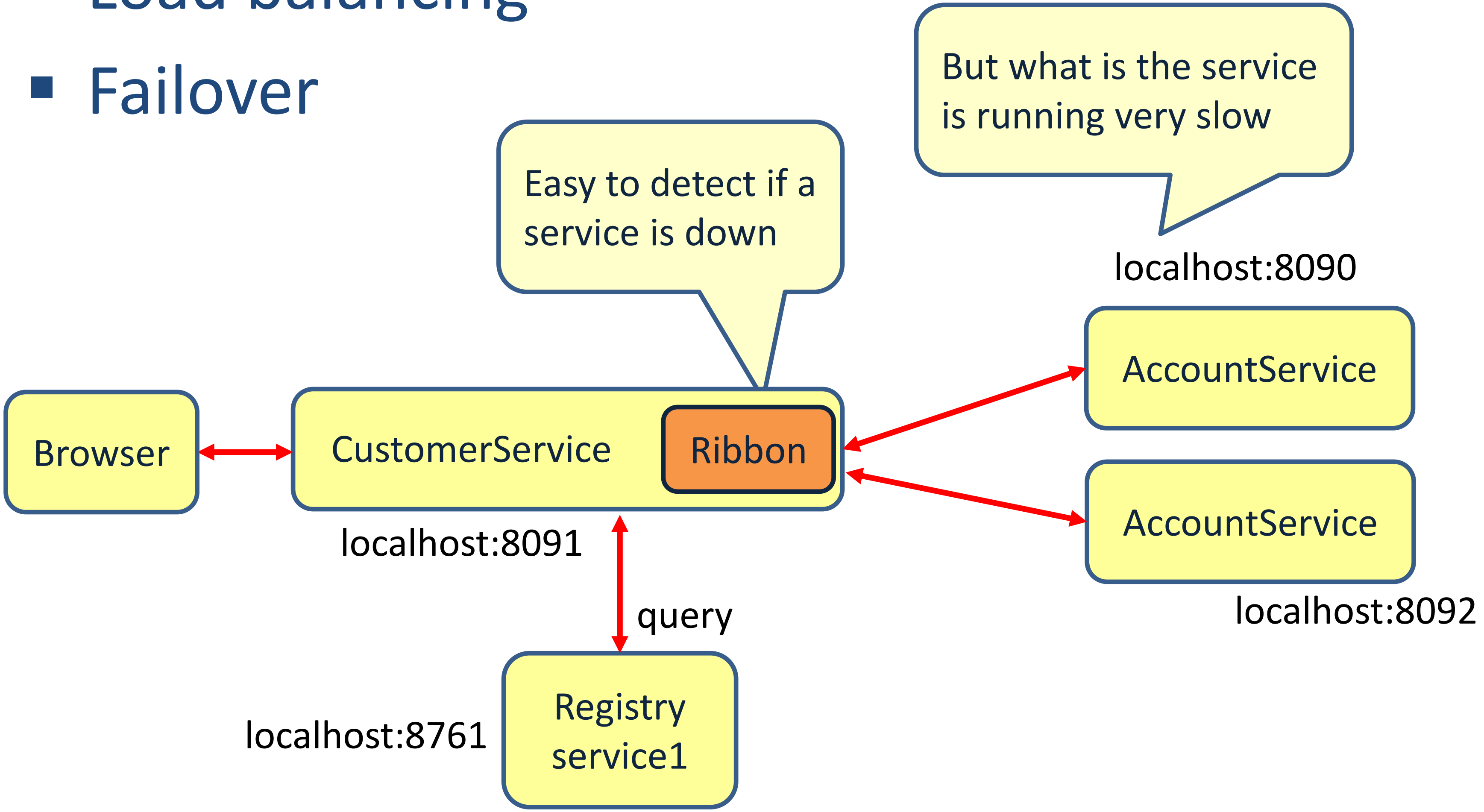
The ability to recover from failures

Fallacies of distributed computing

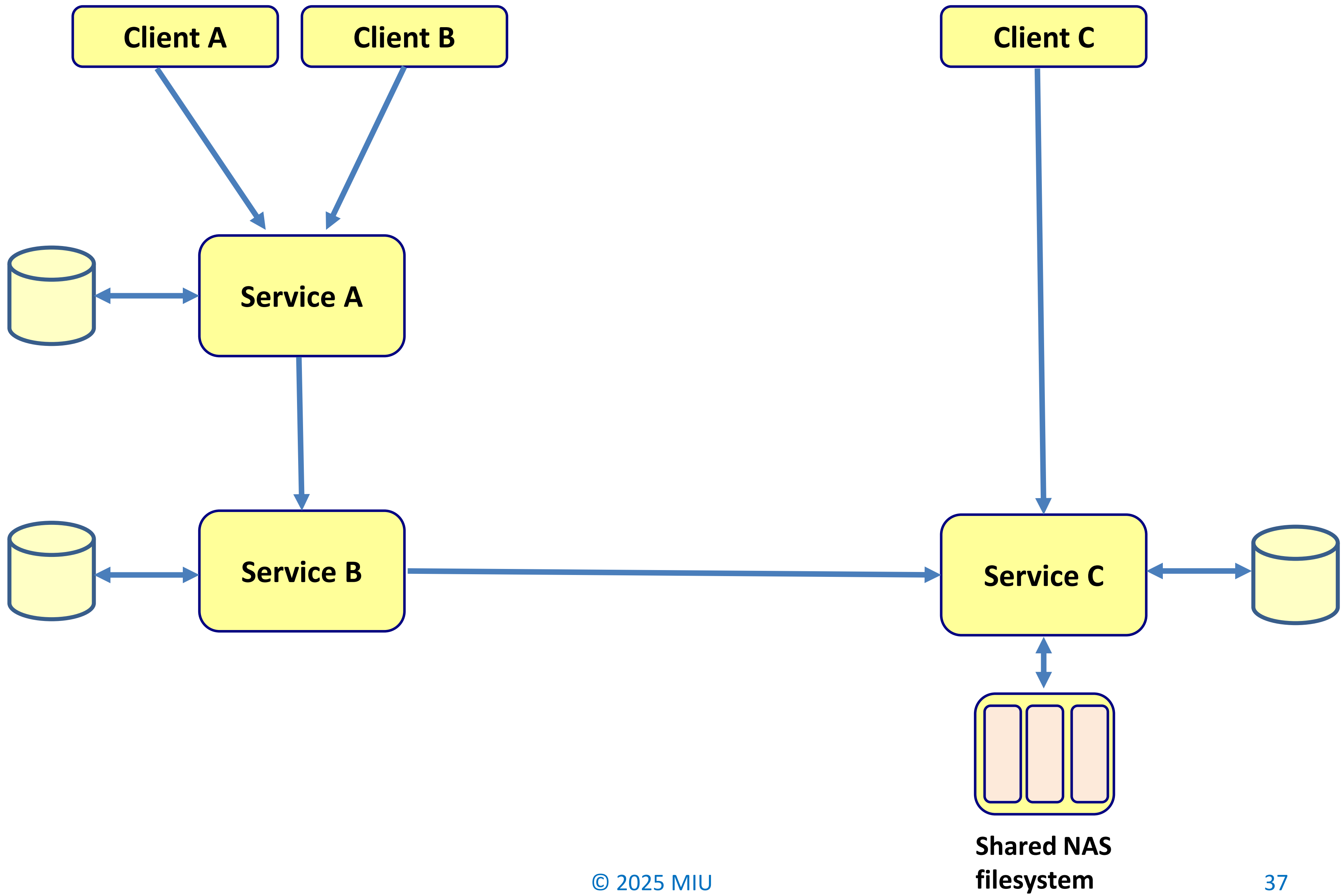
- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn’t change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Clustering

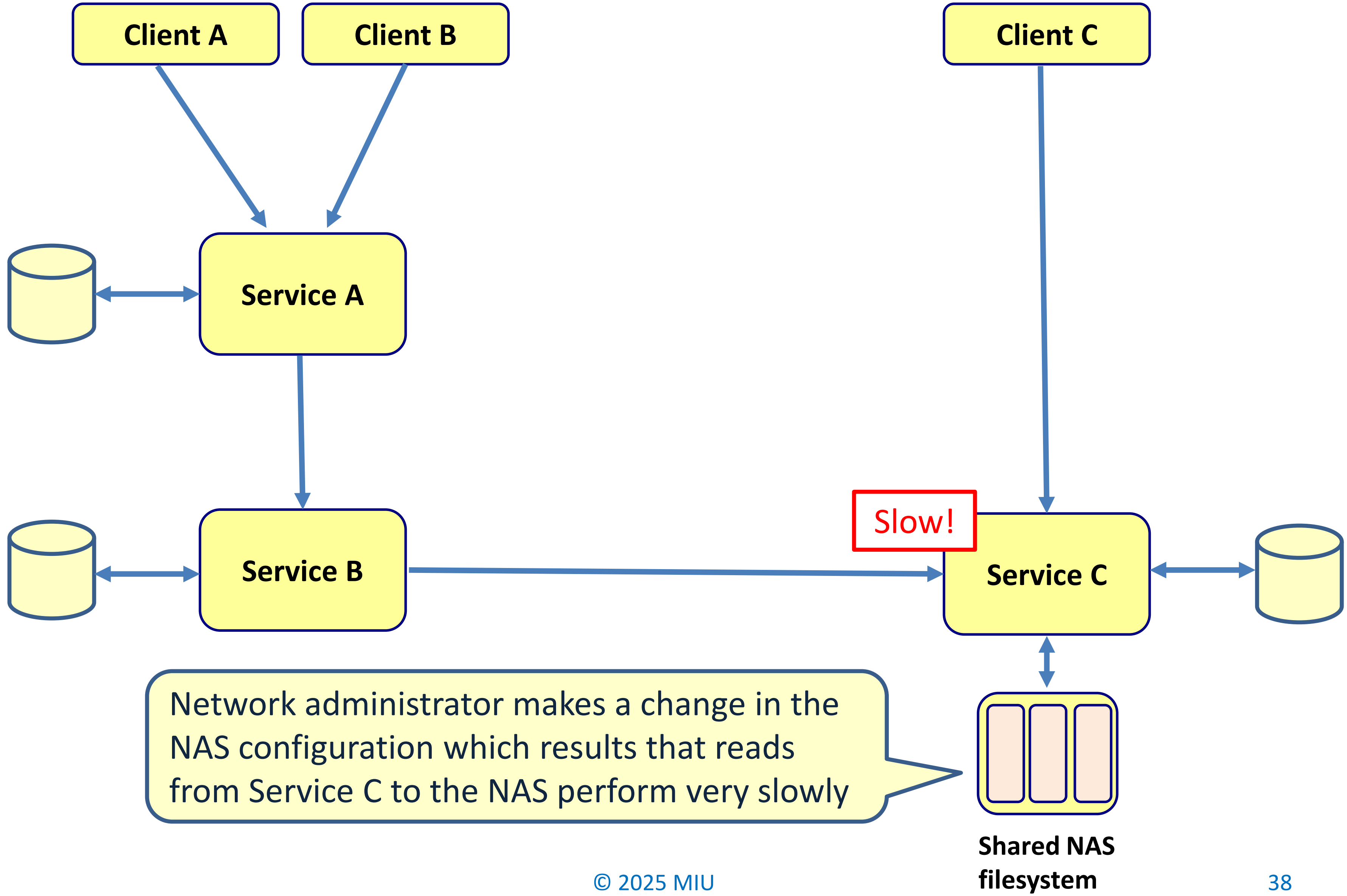
- Load balancing
- Failover



Example

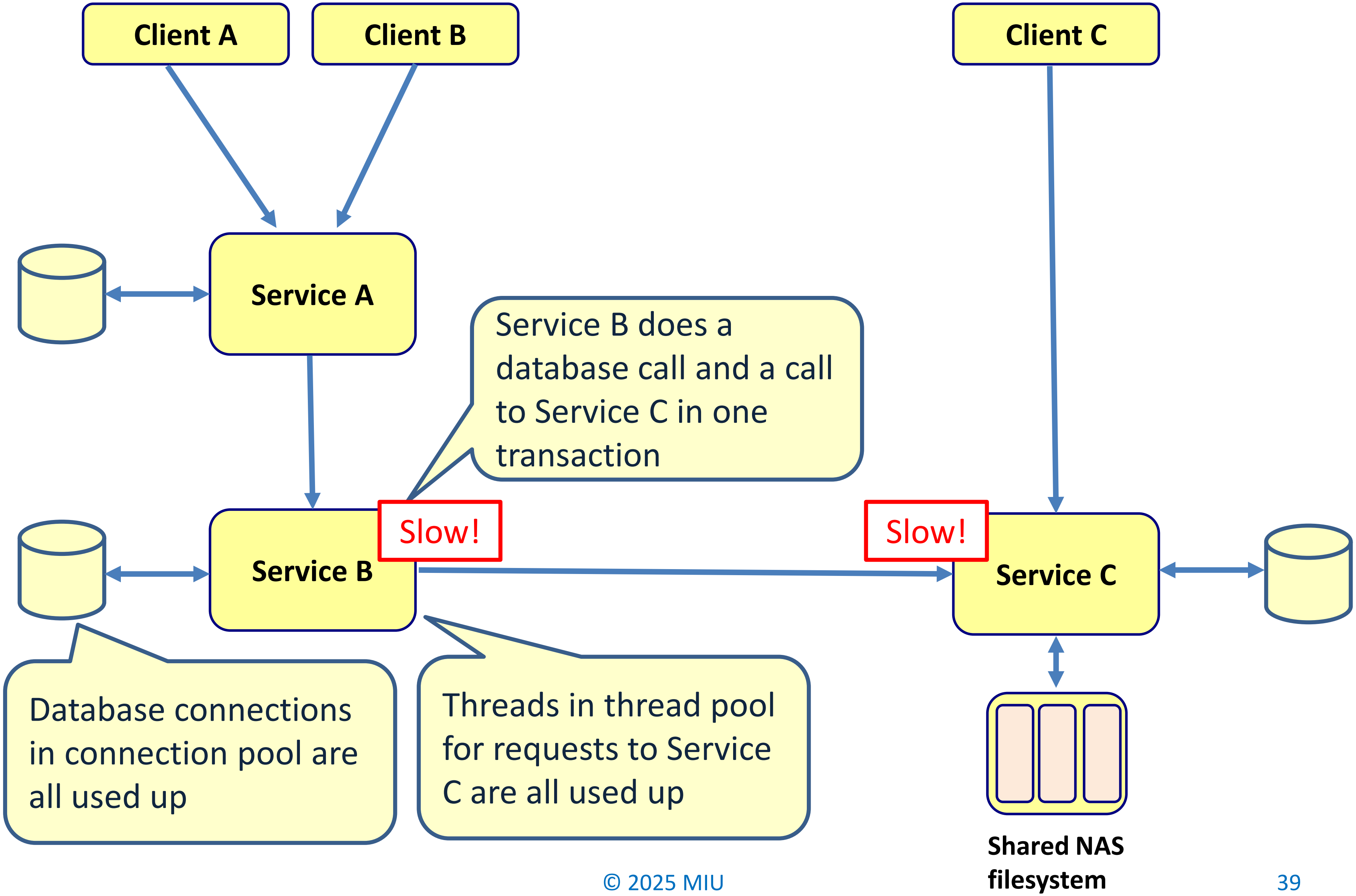

Client A
Client B
Client C
Service A
Service B
Service C
Shared NAS filesystem

Example



Network administrator makes a change in the NAS configuration which results that reads from Service C to the NAS perform very slowly

Example

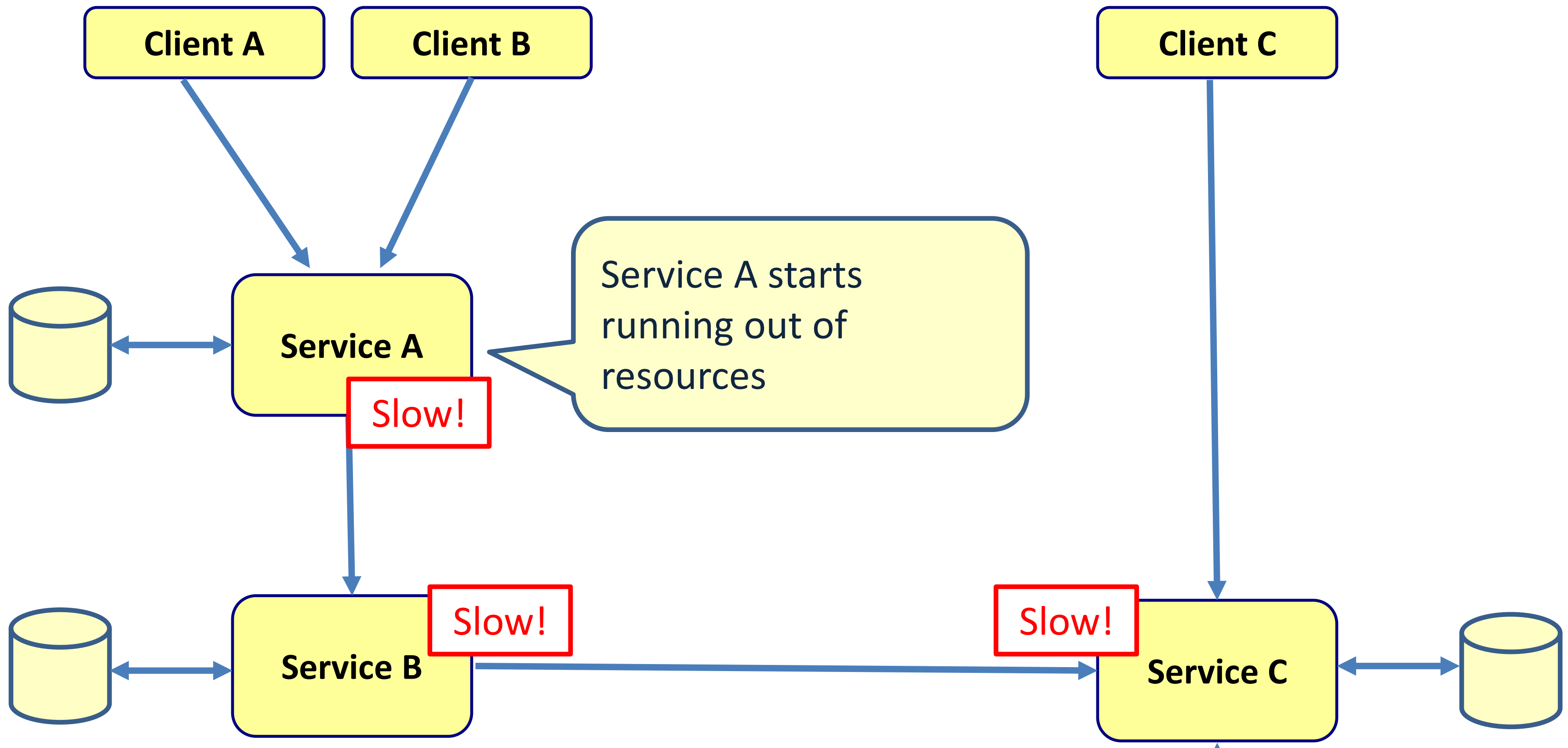


Service B does a database call and a call to Service C in one transaction

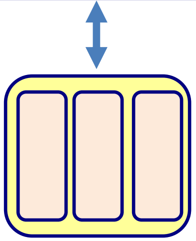
Database connections in connection pool are all used up

Threads in thread pool for requests to Service C are all used up

Example

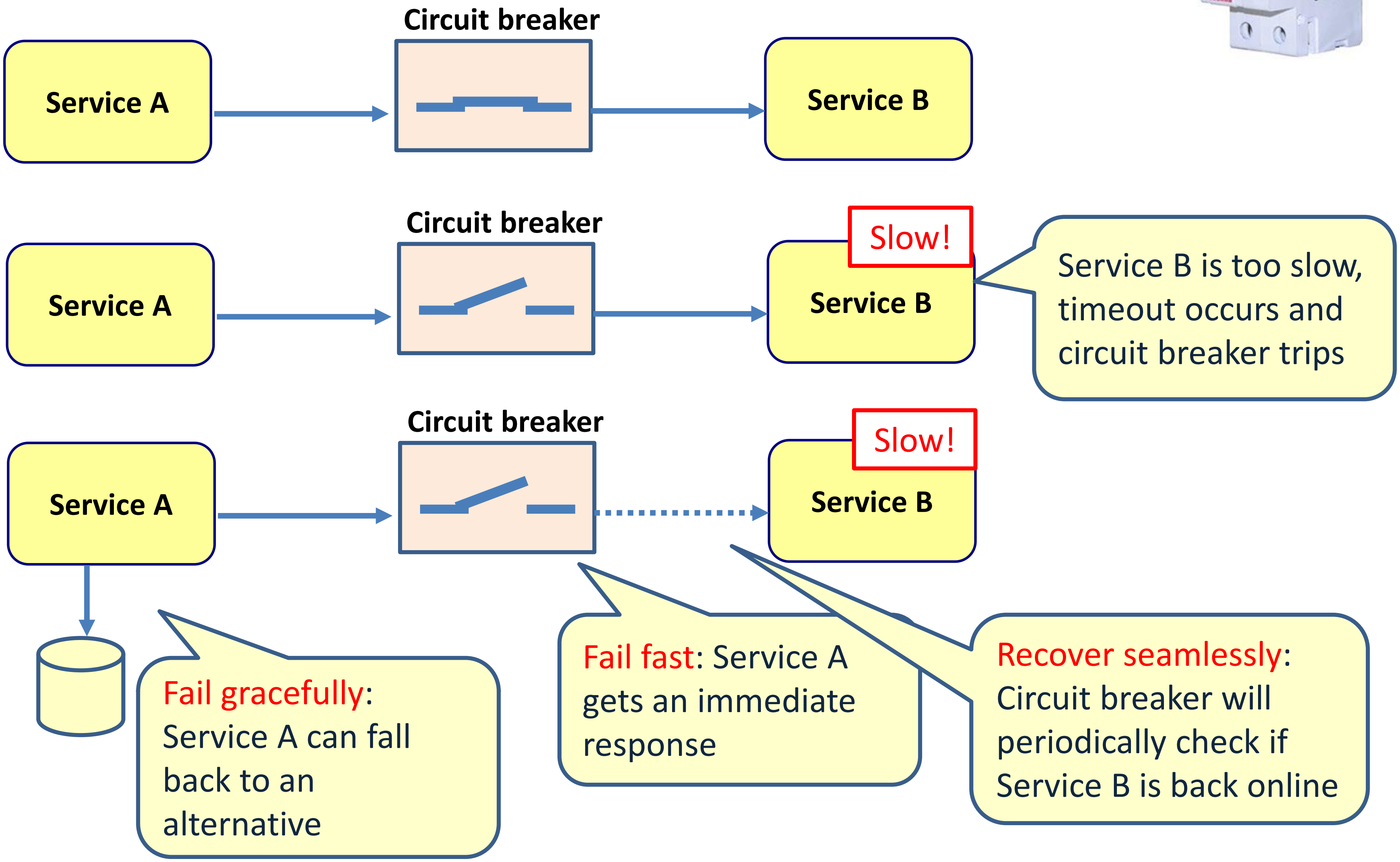


Service A starts running out of resources



Shared NAS filesystem

Circuit breaker



Service B is too slow, timeout occurs and circuit breaker trips

Fail gracefully: Service A can fall back to an alternative

Fail fast: Service A gets an immediate response

Recover seamlessly: Circuit breaker will periodically check if Service B is back online

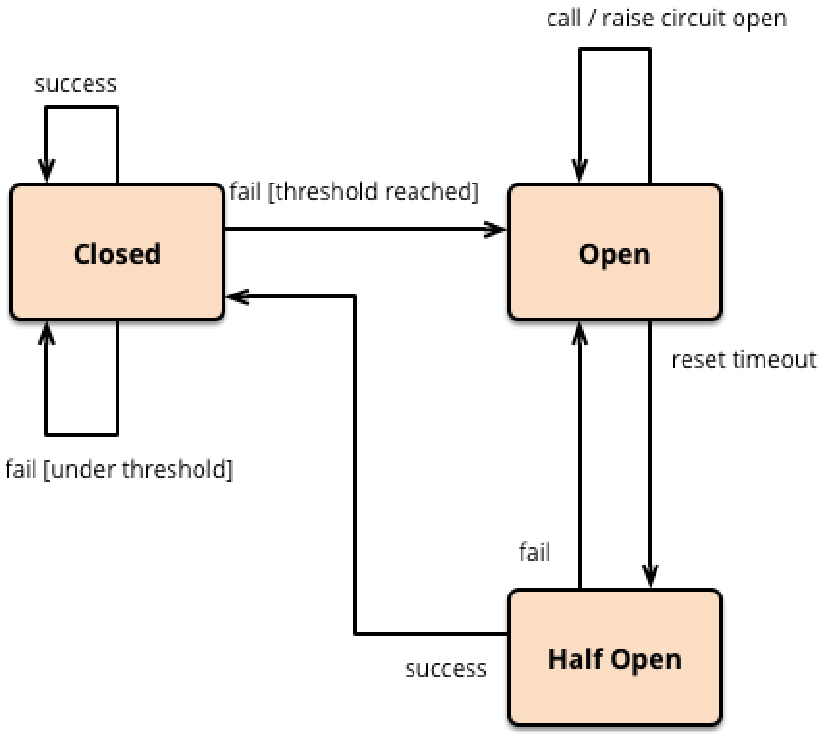
Main point

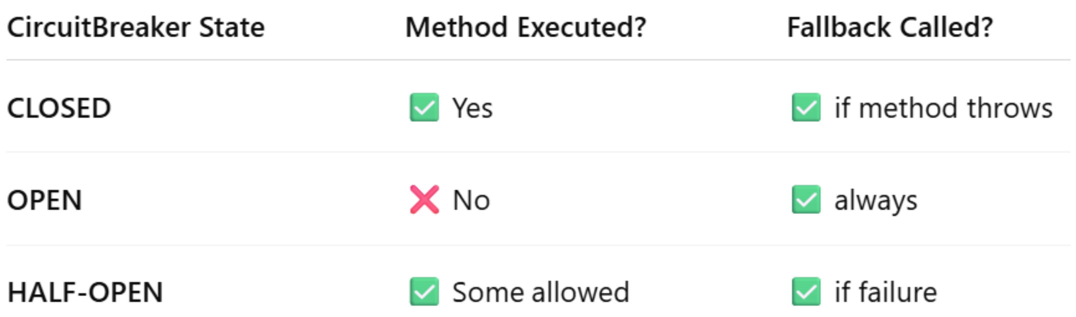
- A circuit breaker takes care that not the whole microservice architecture gets slow when one service becomes slow.

- Every relative part of creation is connected at the level of pure consciousness.

RESILIENCE: RESILIENCE4J

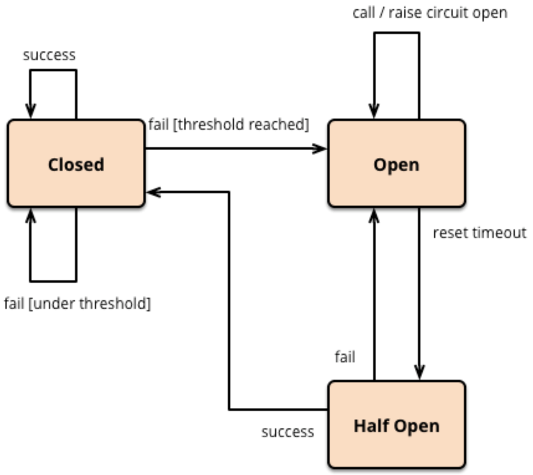
Resilience4J



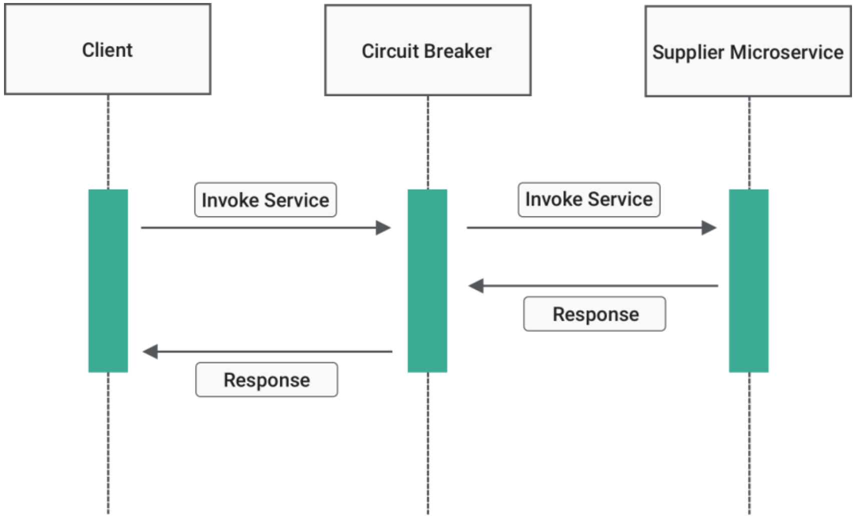


CircuitBreaker State	Method Executed?	Fallback Called?
CLOSED	✅ Yes	✅ if method throws
OPEN	❌ No	✅ always
HALF-OPEN	✅ Some allowed	✅ if failure

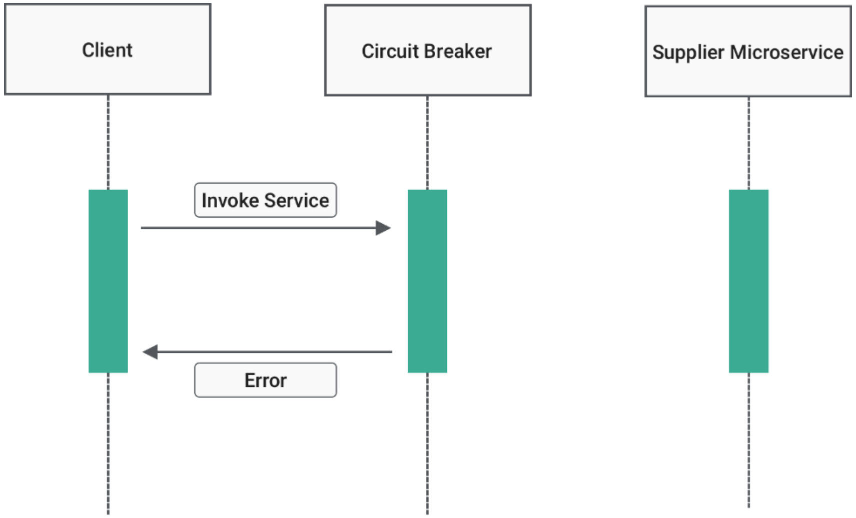
Resilience4J



Closed

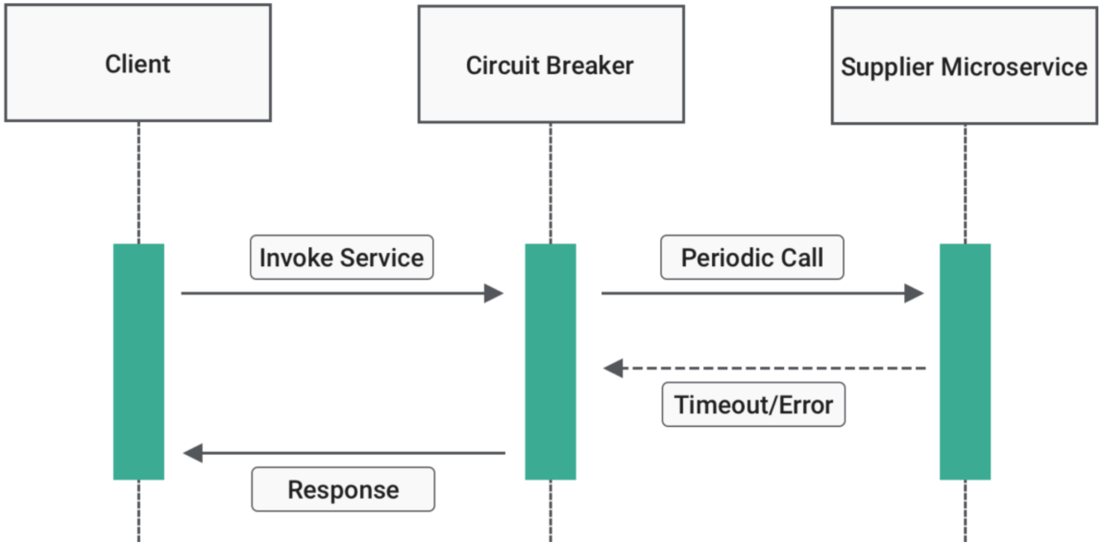


Open



Resilience4J

Half-Open



In the OPEN state and after a wait time duration has elapsed, it makes state transition from OPEN to HALF_OPEN and allows only a configurable number of calls. If the failure rate or slow call rate is greater than or equal to the configured threshold, the state changes back to OPEN. If the failure rate and slow call rate is below the threshold, the state changes back to CLOSED.

ServiceTwo

```
@RestController
public class ServiceTwoController {
   private static final Logger logger =
LoggerFactory.getLogger(ServiceTwoController.class.getName());

   @RequestMapping("/text")
   public String getText() {
      return "World";
   }
}
```

ServiceOne

```
@RestController
public class ServiceOneController {
   @Autowired
   private ServiceTwoClient serviceTwoClient;

   @CircuitBreaker(name = "demoCircuitBreaker", fallbackMethod = "fallbackMethod")
   @RequestMapping("/text")
   public String getText() {
      String service2Text = serviceTwoClient.getText();
      return "Hello "+ service2Text;
   }
   private String fallbackMethod(Throwable throwable) {
      return "Hello World from fallbackMethod";
   }

   @FeignClient("ServiceTwo")
   interface ServiceTwoClient {
      @RequestMapping("/text")
      public String getText();
   }
}
```

Fallback method

configuration

application.yml

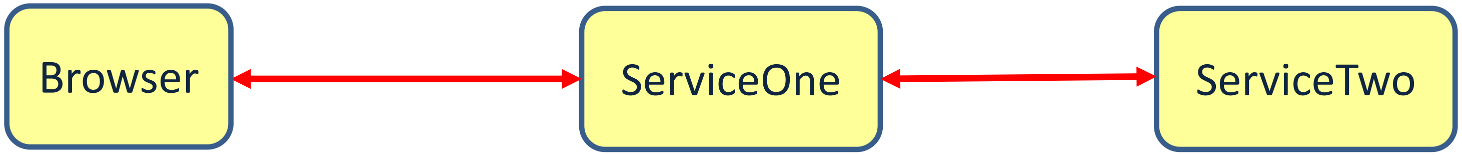
```yaml
server:
  port: 9093

spring:
  application:
    name: ServiceOne
  cloud:
    consul:
      host: localhost
      port: 8500
      discovery:
        enabled: true
        prefer-ip-address: true
        instance-id: ${spring.application.name}:${random.value}

resilience4j:
  circuitbreaker:
    instances:
      demoCircuitBreaker:
        slidingWindowSize: 5
        minimumNumberOfCalls: 3
        failureRateThreshold: 50
        waitDurationInOpenState: 5s
```

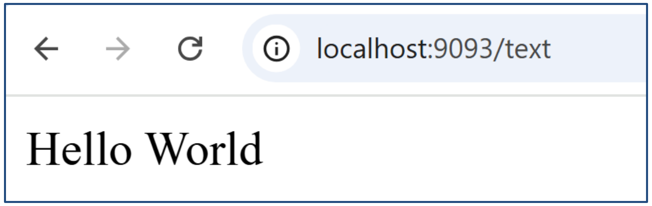
dependencies

pom.xml

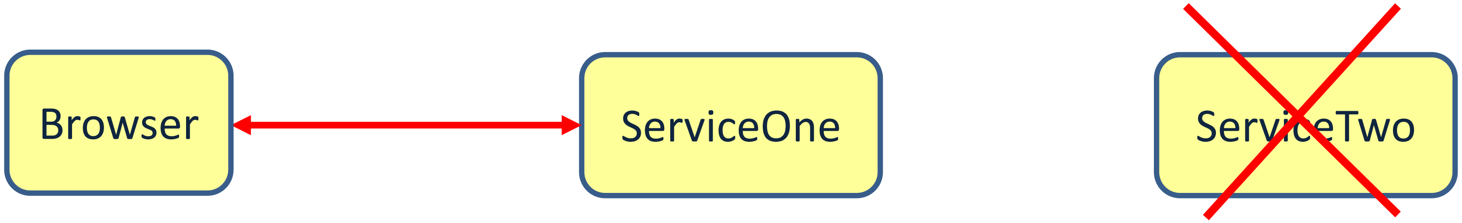
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-consul-discovery</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
  </dependency>
</dependencies>
```

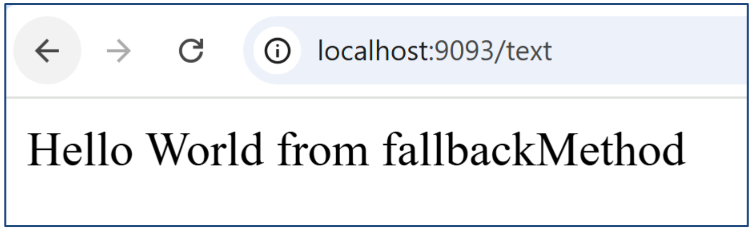
Using Resilience4J





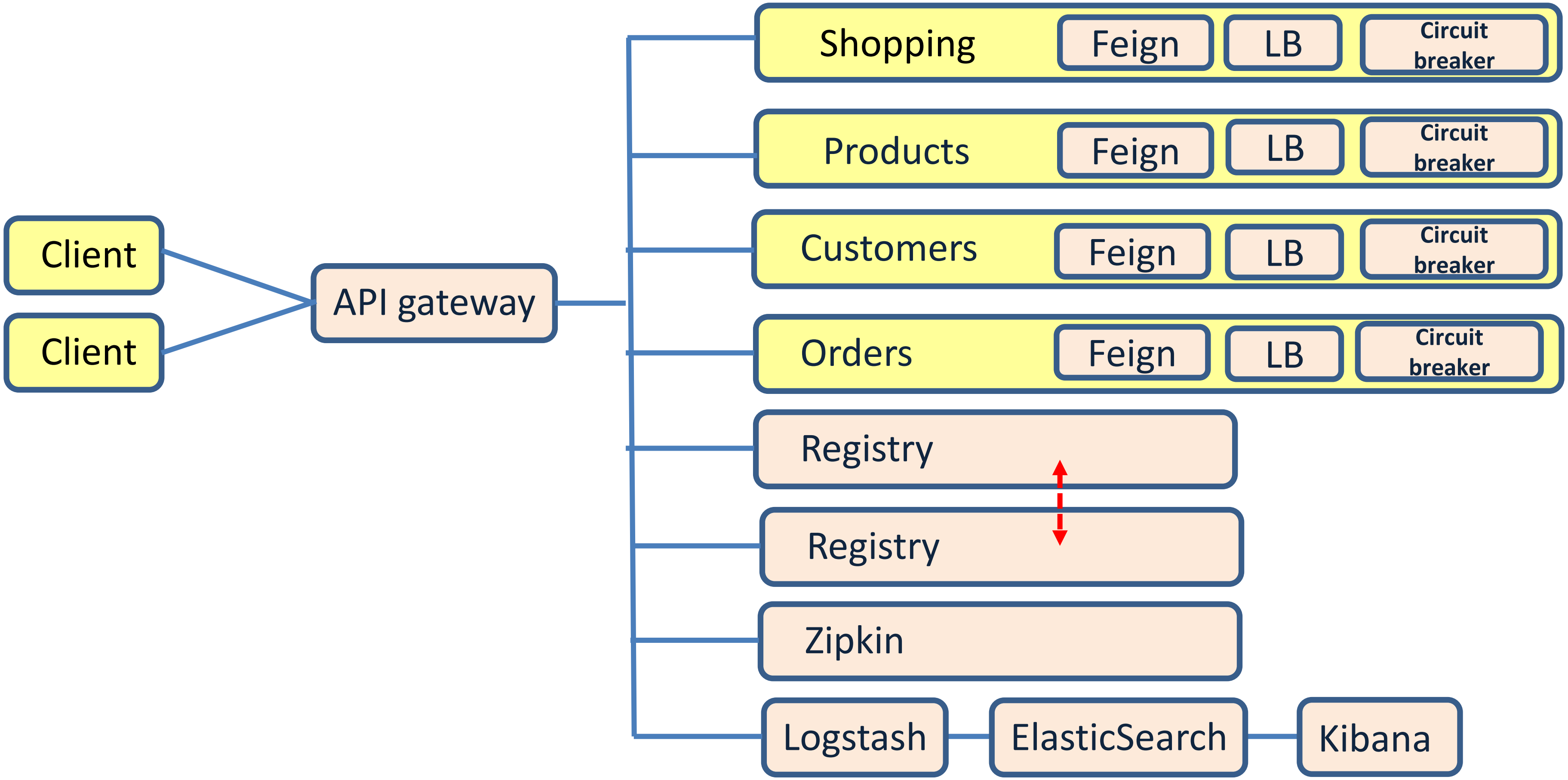
Hello World





Hello World from fallbackMethod

Implementing microservices



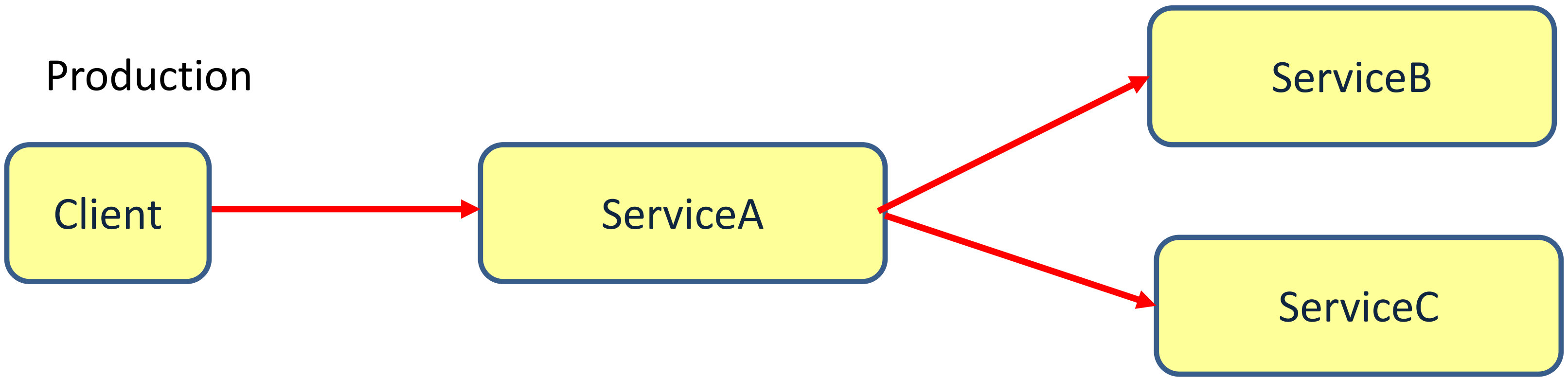
Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

SPRING CLOUD CONTRACT

How to test microservices

Production



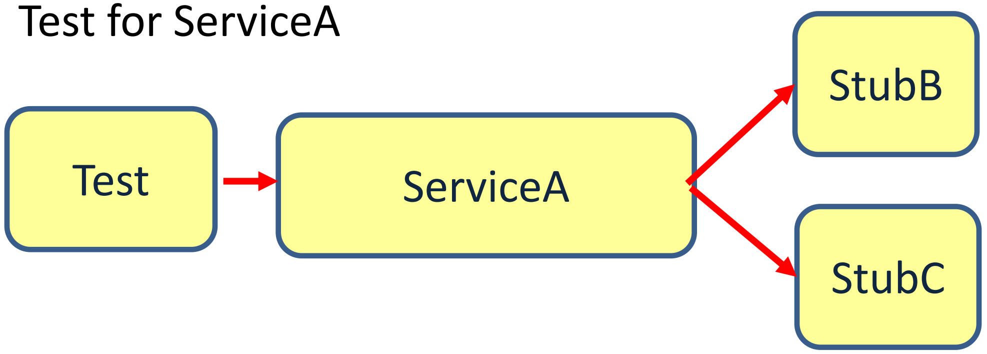
Client

ServiceA

ServiceB

ServiceC

Test for ServiceA



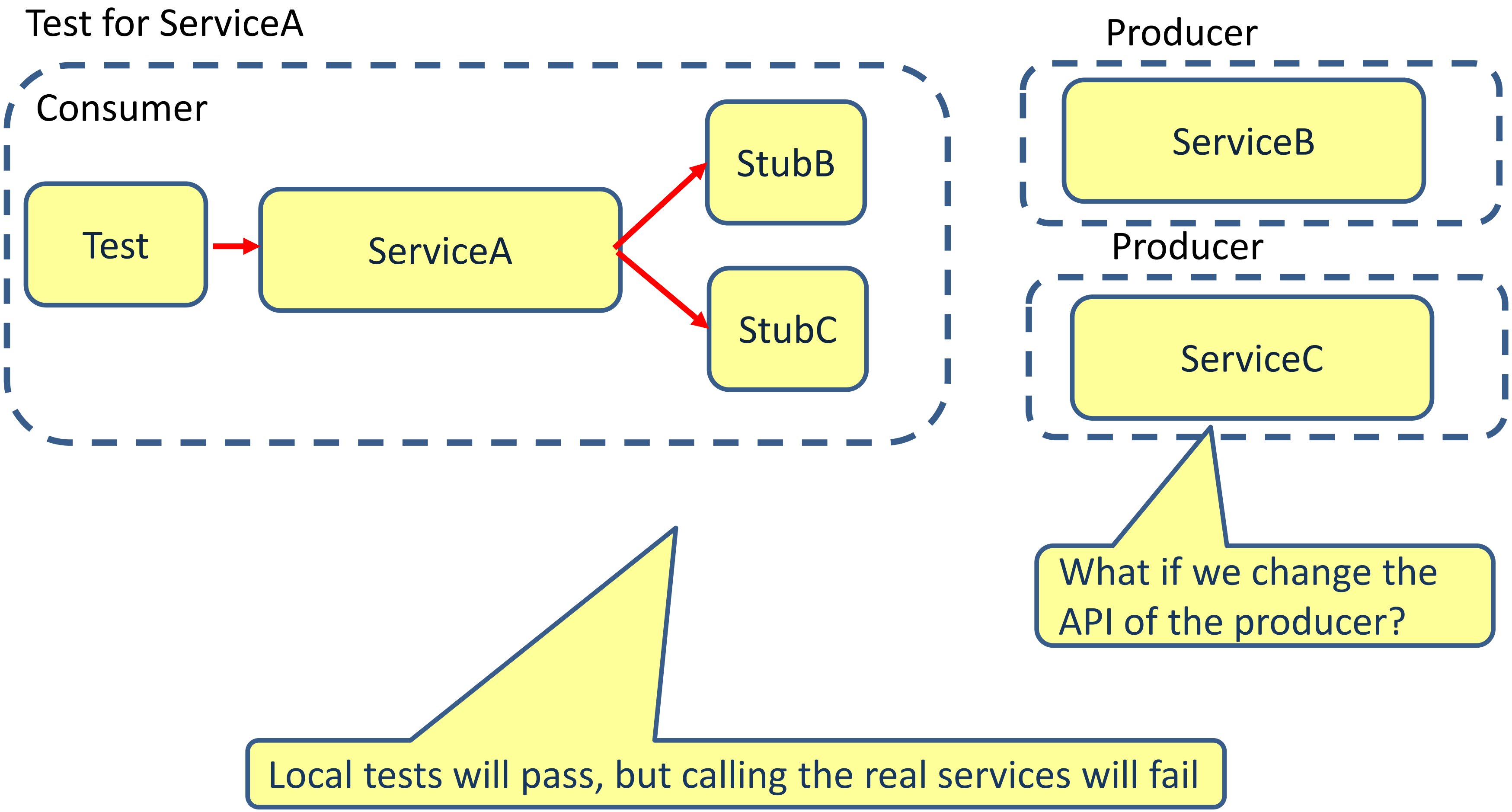
Test

ServiceA

StubB

StubC

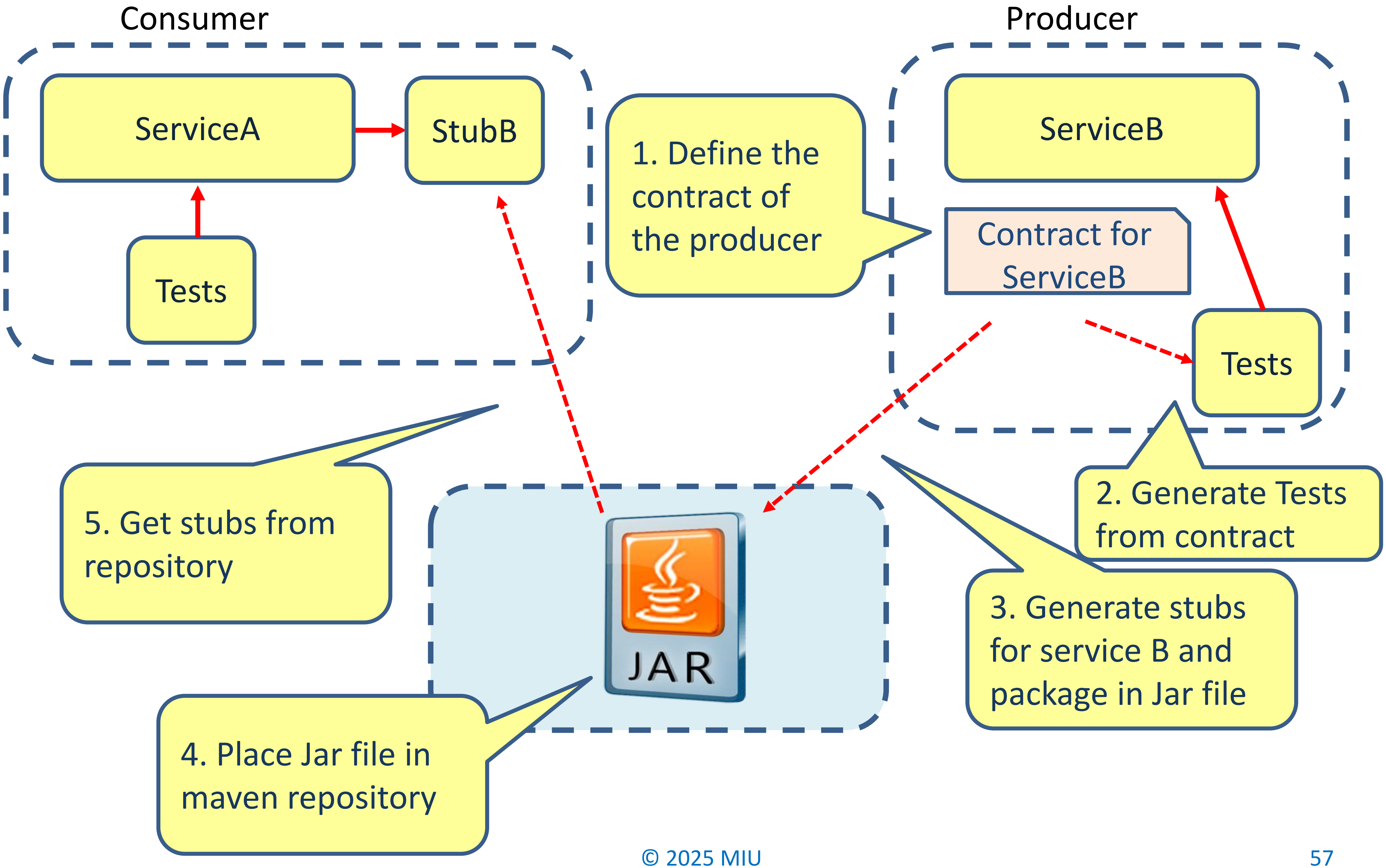
Stubs live at the consumer



What if we change the API of the producer?

Local tests will pass, but calling the real services will fail

Spring cloud contracts



1. Define the contract of the producer

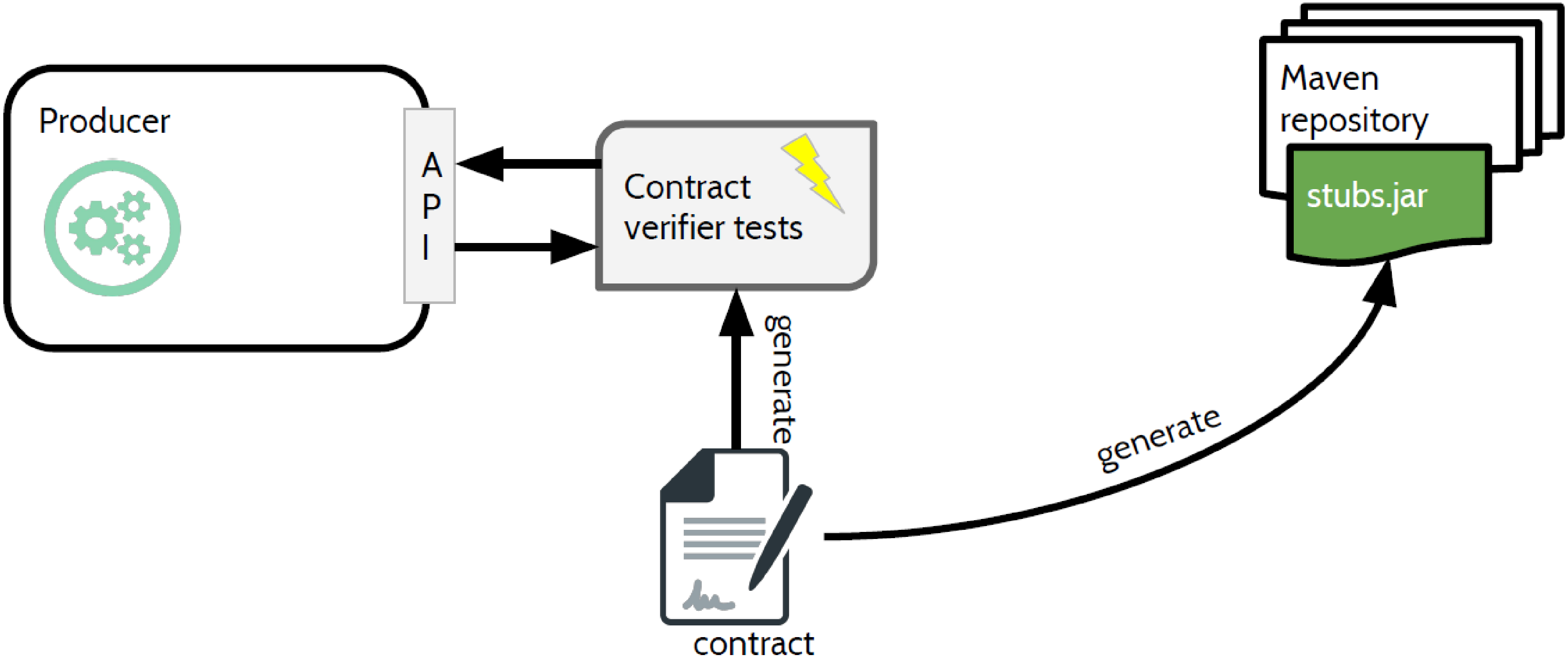
2. Generate Tests from contract

3. Generate stubs for service B and package in Jar file

4. Place Jar file in maven repository

5. Get stubs from repository

Producer


Producer
A
P
I
Contract verifier tests
Maven repository
stubs.jar
generate
generate
contract

Producer maven configuration

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-contract-verifier</artifactId>
  <scope>test</scope>
</dependency>


<plugin>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-contract-maven-plugin</artifactId>
  <version>2.2.2.RELEASE</version>
  <extensions>true</extensions>
  <configuration>
    <baseClassForTests>service.BaseTestClass</baseClassForTests>
    <testFramework>JUNIT5</testFramework>
  </configuration>
</plugin>
```

Producer

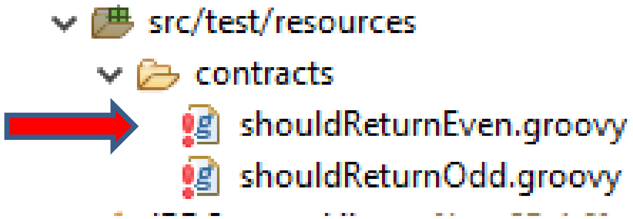
```java
@RestController
public class EvenOddController {

    @GetMapping("/validate")
    public String evenOrOdd(@RequestParam("number") Integer number) {
        return number % 2 == 0 ? "Even" : "Odd";
    }
}
```

```java
@SpringBootApplication
public class EvenoddServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(EvenoddServiceApplication.class, args);
  }
}
```

Producer contract 1

```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
  description "should return even when number input is even"
  request{
    method GET()
    url("/validate") {
      queryParameters {
        parameter("number", "2")
      }
    }
  }
  response {
    body("Even")
    status 200
  }
}
```

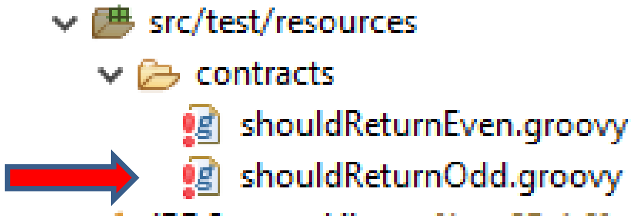
Contract in groovy



```
src/test/resources
  contracts
    shouldReturnEven.groovy
    shouldReturnOdd.groovy
```

Producer contract 2

```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
  description "should return odd when number input is odd"
  request {
    method GET()
    url("/validate") {
      queryParameters {
        parameter("number", "1")
      }
    }
  }
  response {
    body("Odd")
    status 200
  }
}
```



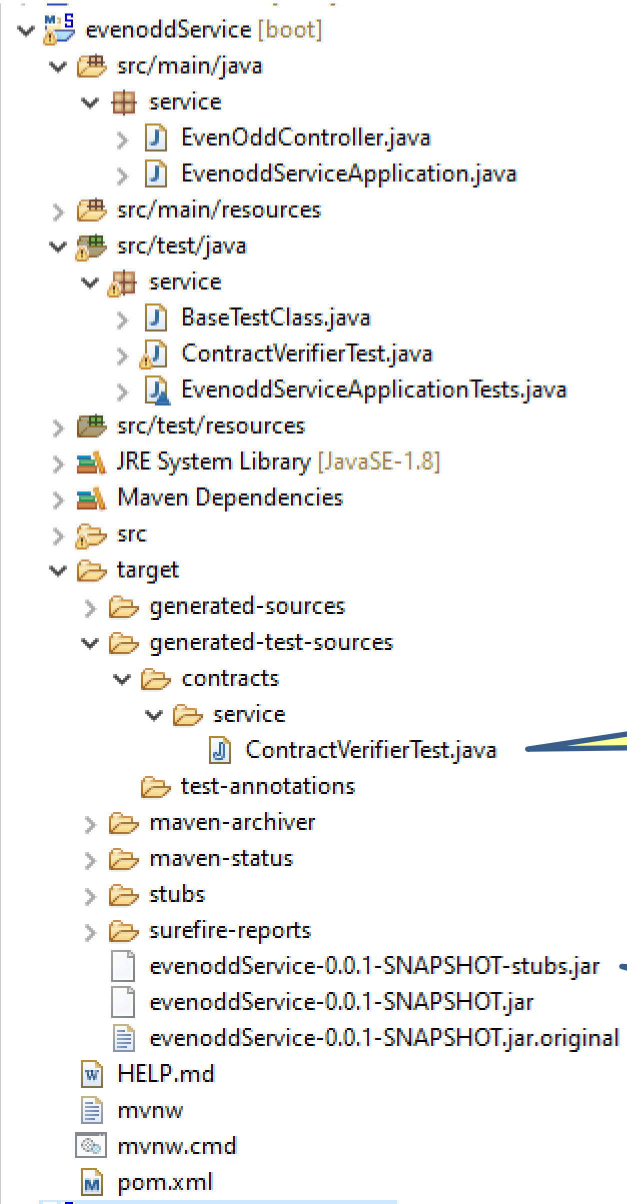
src/test/resources
contracts
shouldReturnEven.groovy
shouldReturnOdd.groovy

Producer: base test class

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@DirtiesContext
@AutoConfigureMessageVerifier
public class BaseTestClass {

    @Autowired
    private EvenOddController evenOddController;

    @BeforeEach
    public void setup() {
        StandaloneMockMvcBuilder standaloneMockMvcBuilder
          = MockMvcBuilders.standaloneSetup(evenOddController);
        RestAssuredMockMvc.standaloneSetup(standaloneMockMvcBuilder);
    }
}
```

This is the base class for all to be generated test classes

After running **maven install**



```
evenoddService [boot]
  src/main/java
    service
      EvenOddController.java
      EvenoddServiceApplication.java
  src/main/resources
  src/test/java
    service
      BaseTestClass.java
      ContractVerifierTest.java
      EvenoddServiceApplicationTests.java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
    generated-sources
    generated-test-sources
      contracts
        service
          ContractVerifierTest.java
      test-annotations
    maven-archiver
    maven-status
    stubs
    surefire-reports
    evenoddService-0.0.1-SNAPSHOT-stubs.jar
    evenoddService-0.0.1-SNAPSHOT.jar
    evenoddService-0.0.1-SNAPSHOT.jar.original
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Generated test class based on the contract

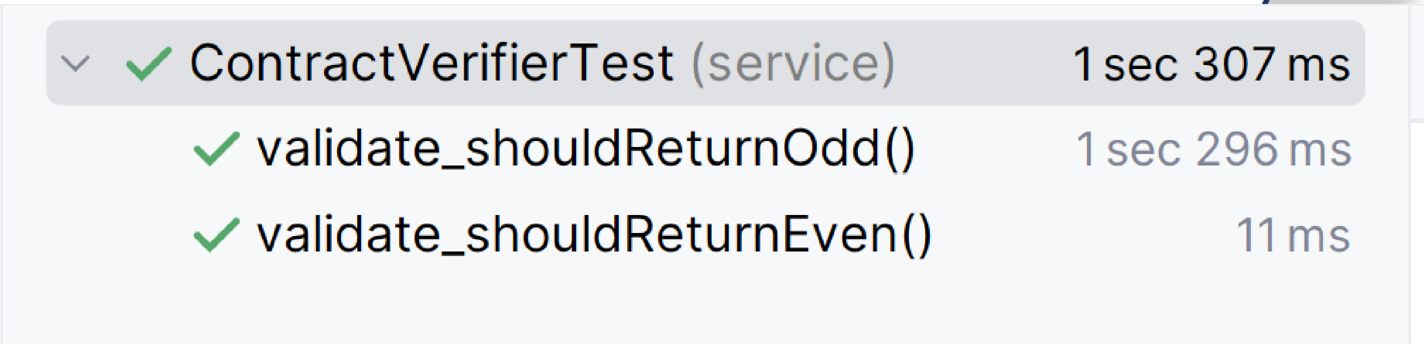
Generated stub classes to be used by the consumer. This jar will be placed in the local maven repository

Generated tests

```
@SuppressWarnings("rawtypes")
public class ContractVerifierTest extends BaseTestClass {

  @Test
  public void validate_shouldReturnEven() throws Exception {
    // given:
      MockMvcRequestSpecification request = given();

    // when:
      ResponseOptions response = given().spec(request)
          .queryParam("number","2")
          .get("/validate");
    // then:
      assertThat(response.statusCode()).isEqualTo(200);
    // and:
      String responseBody = response.getBody().asString();
      assertThat(responseBody).isEqualTo("Even");
  }
```

Generated tests

```java
@Test
public void validate_shouldReturnOdd() throws Exception {
  // given:
    MockMvcRequestSpecification request = given();

  // when:
    ResponseOptions response = given().spec(request)
        .queryParam("number","1")
        .get("/validate");
  // then:
    assertThat(response.statusCode()).isEqualTo(200);
  // and:
    String responseBody = response.getBody().asString();
    assertThat(responseBody).isEqualTo("Odd");
}

}
```

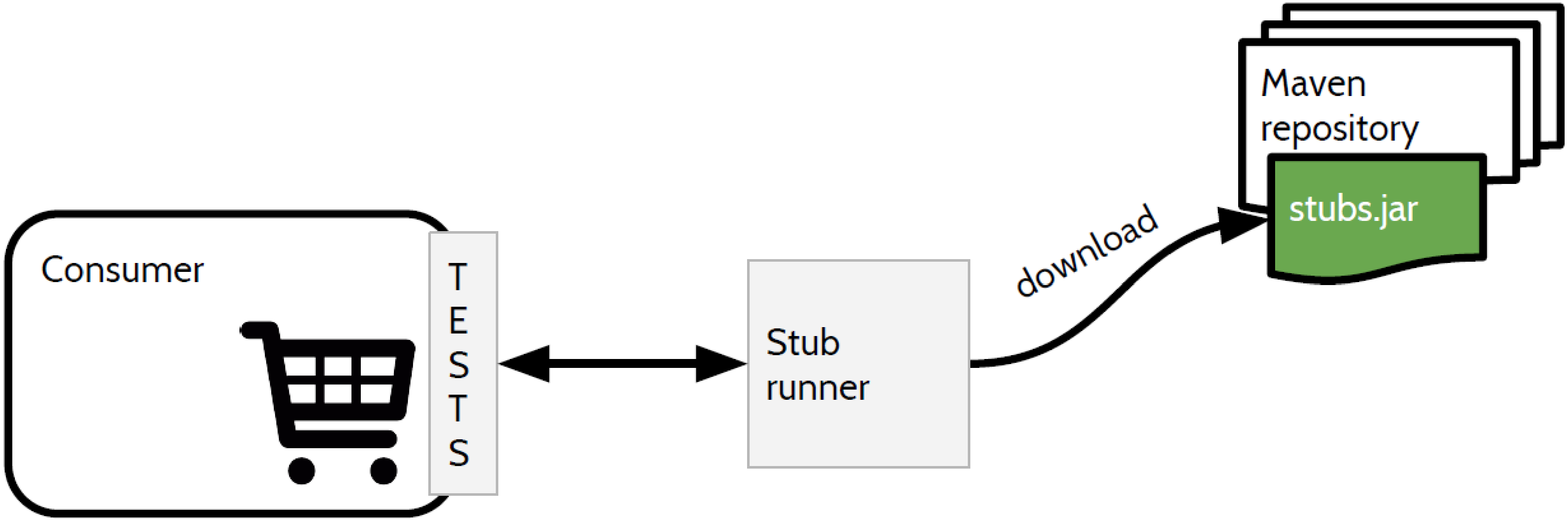


```
✓ ContractVerifierTest (service)      1 sec 307 ms
    ✓ validate_shouldReturnOdd()      1 sec 296 ms
    ✓ validate_shouldReturnEven()            11 ms
```

Spring cloud contract DSL

```
import org.springframework.cloud.contract.spec.Contract
Contract.make {
  description("GET employee with id=1")
  request {
    method 'GET'
    url '/employee/1'
  }
 response {
    status 200
    body("""
    {
    "id": "1",
    "fname": "Jane",
    "lname": "Doe",
    "salary": "123000.00",
    "gender": "M"
    }
    """)
    headers {
      contentType(applicationJson())
    }
  }
}
```

Consumer



```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-contract-stub-runner</artifactId>
  <version>2.2.2.RELEASE</version>
  <scope>test</scope>
</dependency>
```

Consumer

```
@RestController
public class MathController {

    private RestTemplate restTemplate = new RestTemplate();

    @GetMapping("/calculate")
    public String checkOddAndEven(@RequestParam("number") Integer number) {
        HttpHeaders httpHeaders = new HttpHeaders();
        httpHeaders.add("Content-Type", "application/json");

        ResponseEntity<String> responseEntity = restTemplate.exchange(
          "http://localhost:8090/validate?number=" + number,
          HttpMethod.GET,
          new HttpEntity<>(httpHeaders),
          String.class);

        return responseEntity.getBody();
    }
}
```

Consumer

```java
@SpringBootApplication
public class MathServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(MathServiceApplication.class, args);
  }

}
```

Consumer test

Get the stubs from the local repository

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
        ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void given_WhenPassEvenNumberInQueryParam_ThenReturnEven() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=2")
          .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk())
          .andExpect(content().string("Even"));
    }

    @Test
    public void given_WhenPassOddNumberInQueryParam_ThenReturnOdd() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=1")
          .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk())
          .andExpect(content().string("Odd"));
    }
}
```

Consumer test

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
        ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {
```

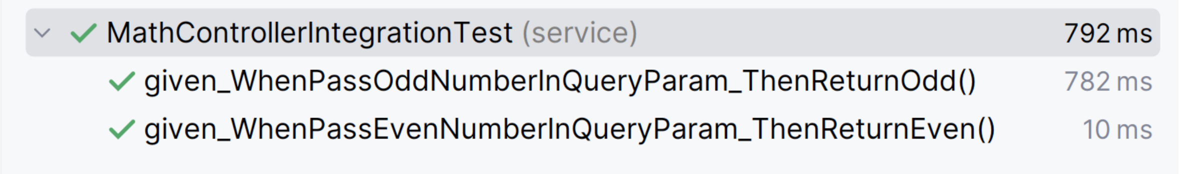
Group id

Artifact id

Version + means latest version

stubs

Port number to run the stubs on



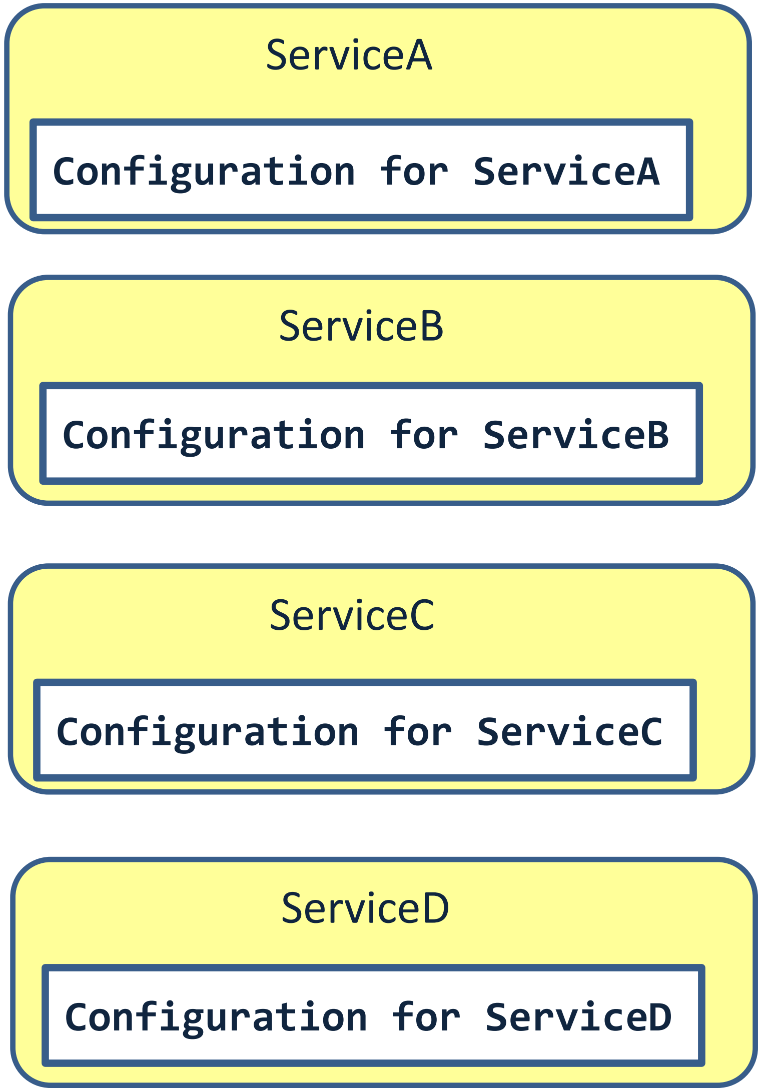
```
✓ MathControllerIntegrationTest (service)                          792 ms
    ✓ given_WhenPassOddNumberInQueryParam_ThenReturnOdd()          782 ms
    ✓ given_WhenPassEvenNumberInQueryParam_ThenReturnEven()         10 ms
```

Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

CENTRALIZED CONFIGURATION SERVICE

Local configuration



ServiceA

Configuration for ServiceA

ServiceB

Configuration for ServiceB

ServiceC

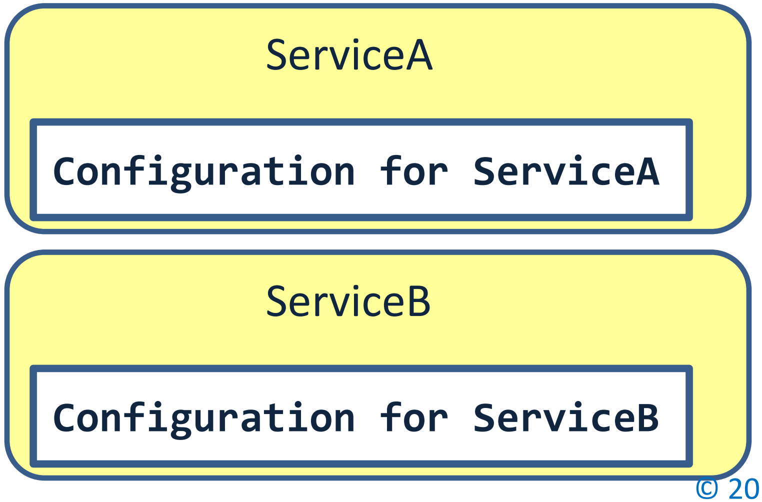
Configuration for ServiceC

ServiceD

Configuration for ServiceD

Local configuration challenges

- When we change the configuration we need to rebuild and redeploy the application
- Configuration may contain sensitive information
- Some of the properties are the same among services: lots of duplication



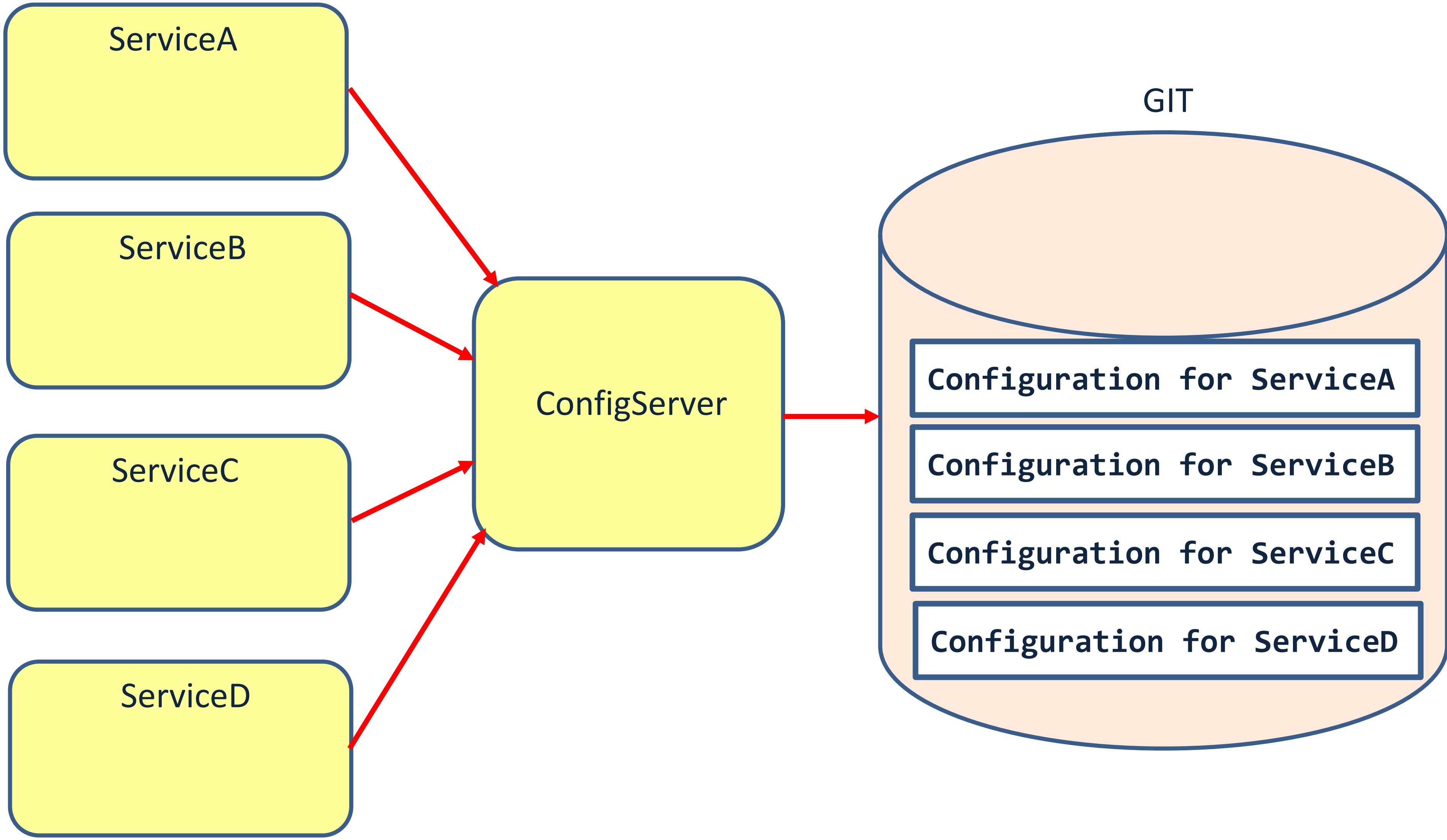
ServiceA

`Configuration for ServiceA`

ServiceB

`Configuration for ServiceB`

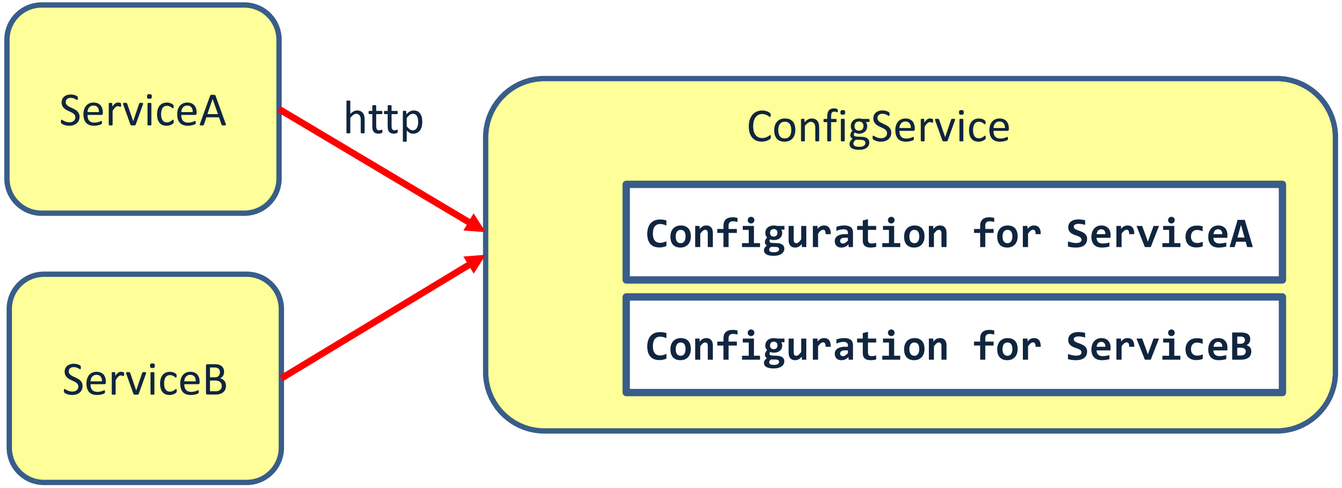
Spring cloud config server


ServiceA
ServiceB
ServiceC
ServiceD
ConfigServer
GIT
Configuration for ServiceA
Configuration for ServiceB
Configuration for ServiceC
Configuration for ServiceD

Spring cloud config

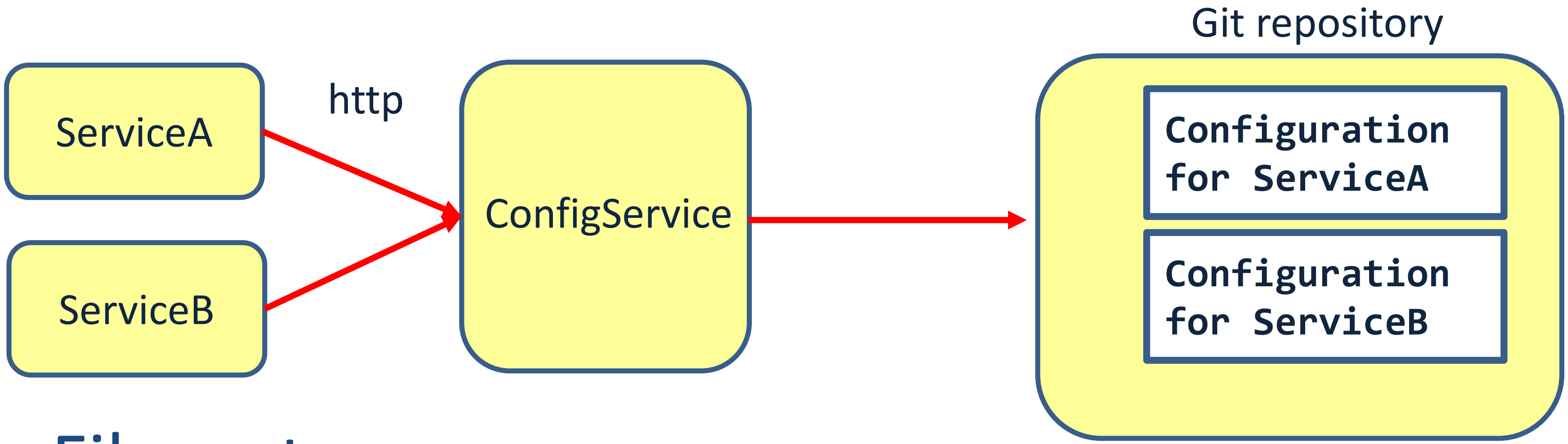
- HTTP access to centralized configuration

Centralized configuration



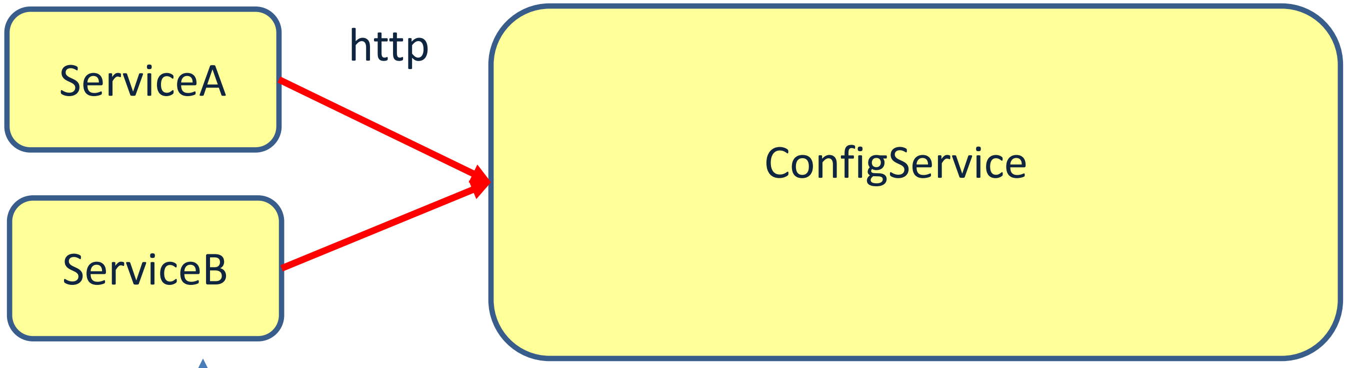
Where to store the config files?

- Git based (Github)



- File system
- HashiCorp Vault
- Database
- Cloud Secrets/Config Stores

Spring cloud config example



pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

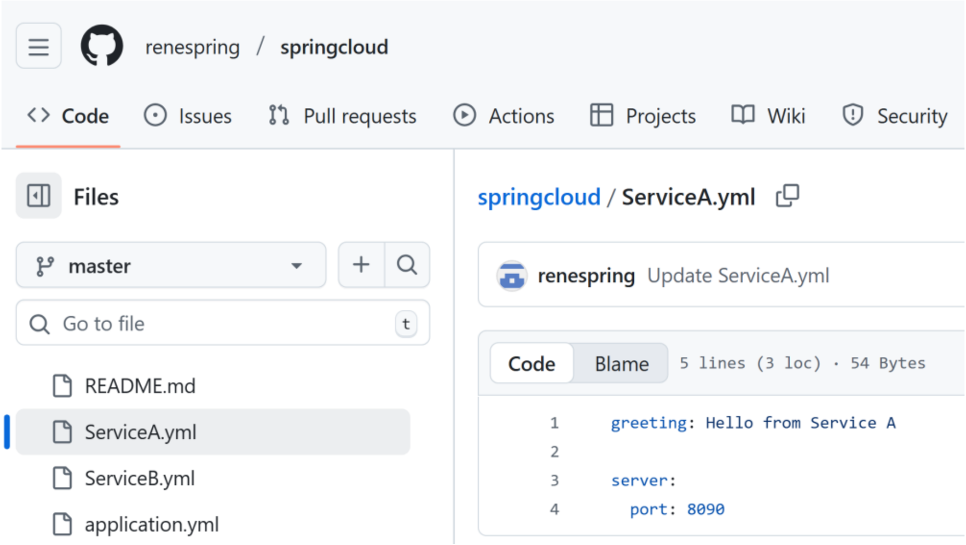
Configuration server

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(ConfigServiceApplication.class, args);
  }
}
```

application.yml

```yaml
server:
  port: 8888
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/renespring/springcloud.git
```

Config files in github



renespring / springcloud

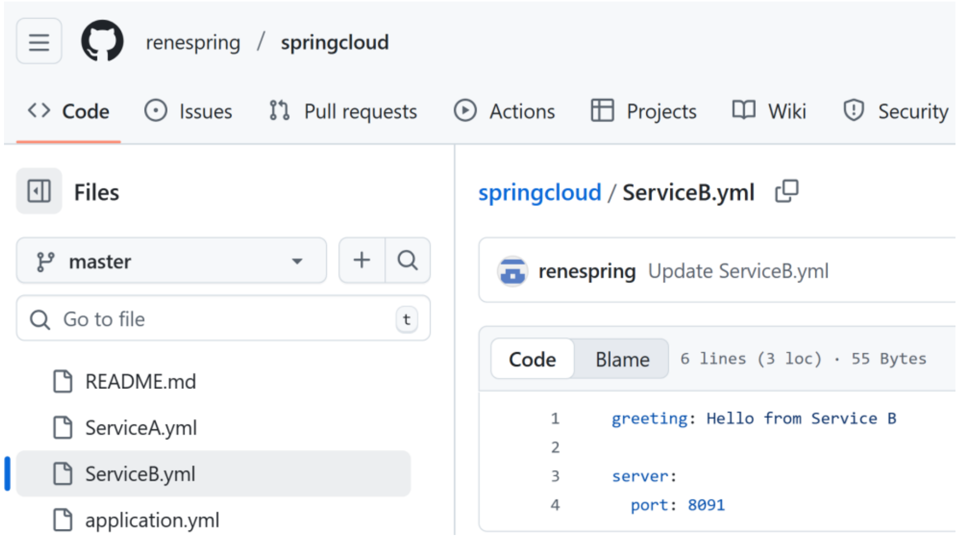
- README.md
- ServiceA.yml
- ServiceB.yml
- application.yml

springcloud / ServiceA.yml

renespring Update ServiceA.yml

5 lines (3 loc) · 54 Bytes

```yaml
greeting: Hello from Service A

server:
  port: 8090
```



renespring / springcloud

- README.md
- ServiceA.yml
- ServiceB.yml
- application.yml

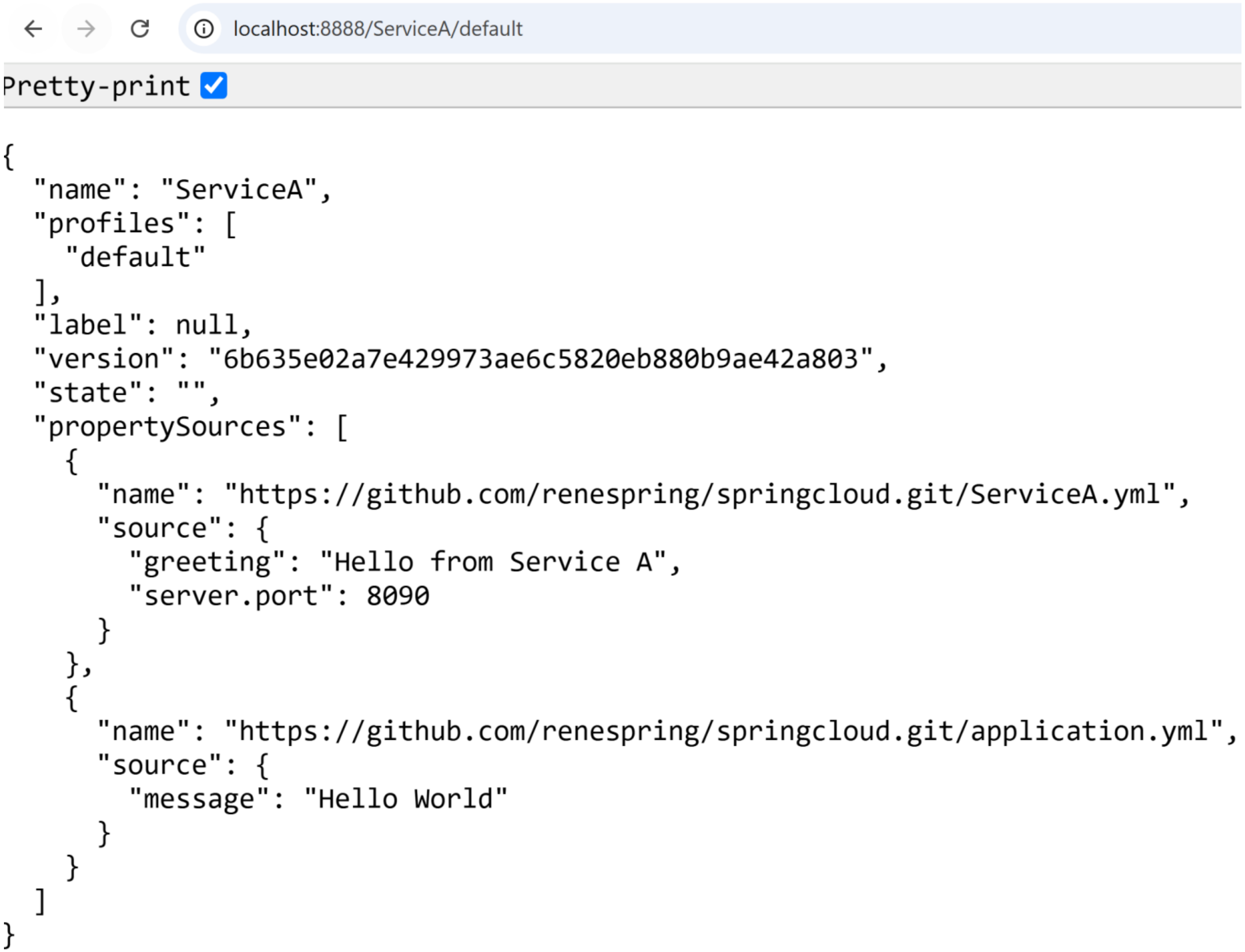
springcloud / ServiceB.yml

renespring Update ServiceB.yml

6 lines (3 loc) · 55 Bytes

```yaml
greeting: Hello from Service B

server:
  port: 8091
```

Configuration server



localhost:8888/ServiceA/default

Pretty-print ☑

```
{
  "name": "ServiceA",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": "6b635e02a7e429973ae6c5820eb880b9ae42a803",
  "state": "",
  "propertySources": [
    {
      "name": "https://github.com/renespring/springcloud.git/ServiceA.yml",
      "source": {
        "greeting": "Hello from Service A",
        "server.port": 8090
      }
    },
    {
      "name": "https://github.com/renespring/springcloud.git/application.yml",
      "source": {
        "message": "Hello World"
      }
    }
  ]
}
```

Configuration client: ServiceA

```java
@SpringBootApplication
public class ServiceAApplication {

  public static void main(String[] args) {
    SpringApplication.run(ServiceAApplication.class, args);
  }
}
```

```java
@RestController
public class ServiceAController {
  @Value("${greeting}")
  private String message;

  @RequestMapping("/")
  public String getName() {
    return message;
  }
}
```

application.yml

```yaml
spring:
  application:
    name: ServiceA
  config:
    import: configserver:http://localhost:8888
```

Configuration client: ServiceB

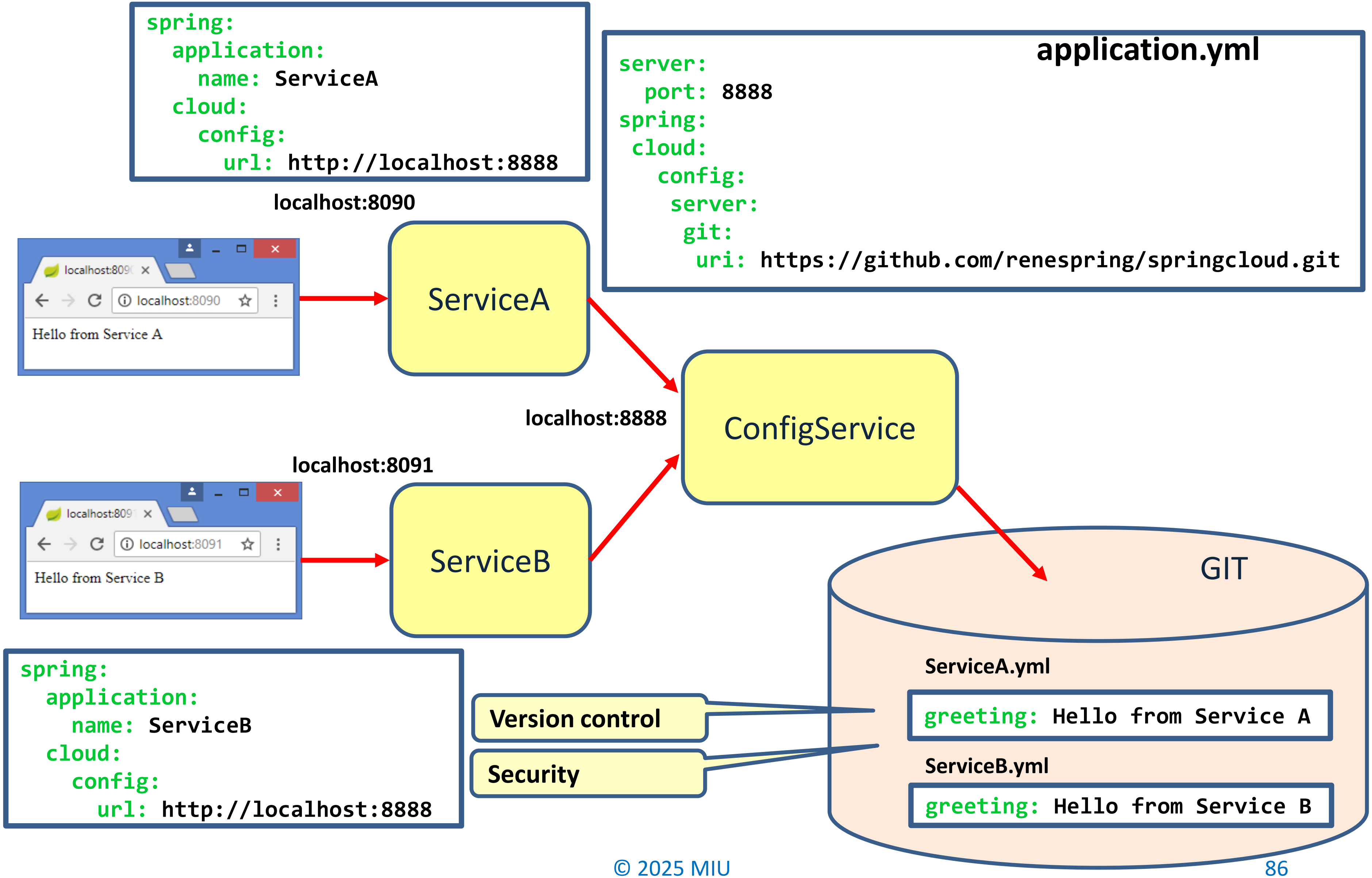
```java
@SpringBootApplication
public class ServiceBApplication {

  public static void main(String[] args) {
    SpringApplication.run(ServiceBApplication.class, args);
  }
}
```

```java
@RestController
public class ServiceBController {
  @Value("${greeting}")
  private String message;

  @RequestMapping("/")
  public String getName() {
    return message;
  }
}
```

application.yml

```yaml
spring:
  application:
    name: ServiceB
  config:
    import: configserver:http://localhost:8888
```

Use of the Config Server



```
spring:
  application:
    name: ServiceA
  cloud:
    config:
      url: http://localhost:8888
```

application.yml

```
server:
  port: 8888
spring:
 cloud:
   config:
    server:
     git:
      uri: https://github.com/renespring/springcloud.git
```

localhost:8090

localhost:8090

Hello from Service A

localhost:8091

localhost:8091

Hello from Service B

ServiceA.yml

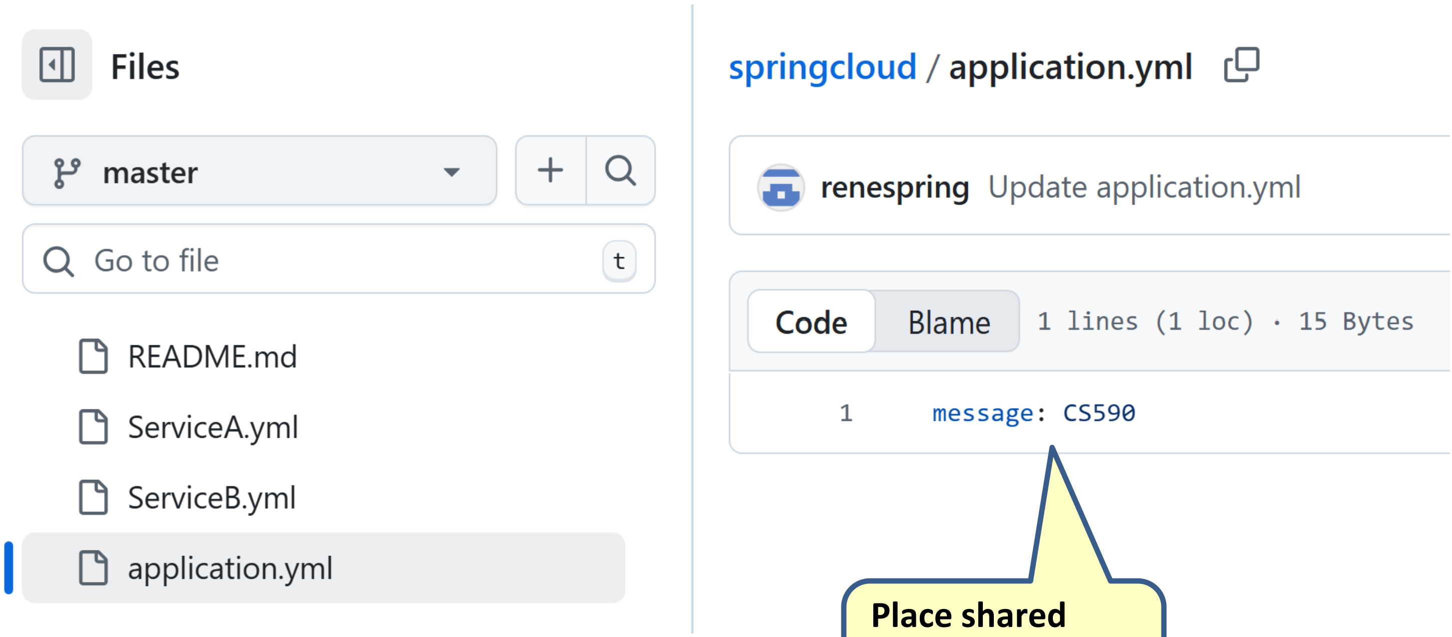
```
greeting: Hello from Service A
```

ServiceB.yml

```
greeting: Hello from Service B
```

```
spring:
  application:
    name: ServiceB
  cloud:
    config:
      url: http://localhost:8888
```

Shared configuration



- README.md
- ServiceA.yml
- ServiceB.yml
- application.yml

springcloud / application.yml

renespring Update application.yml

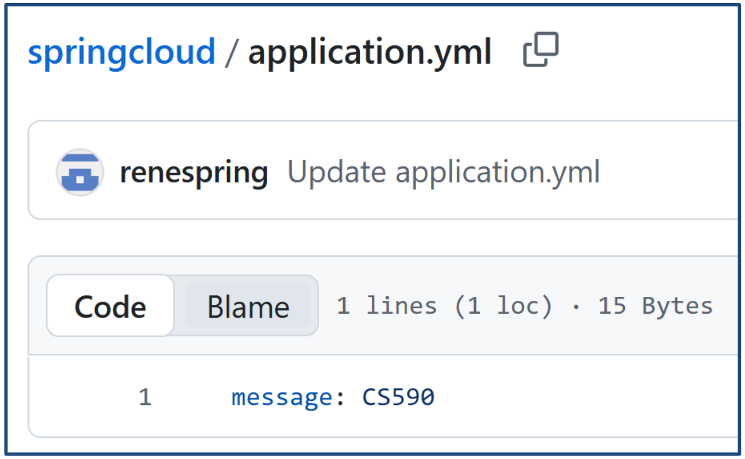
1 lines (1 loc) · 15 Bytes

```yaml
message: CS590
```

Place shared configuration in application.yml

Shared configuration

```
@RestController
public class ServiceAController {
    @Value("${greeting}")
    private String greeting;

    @Value("${message}")
    private String message;

    @RequestMapping("/")
    public String getName() {
        return greeting +" from "+ message;
    }
}
```

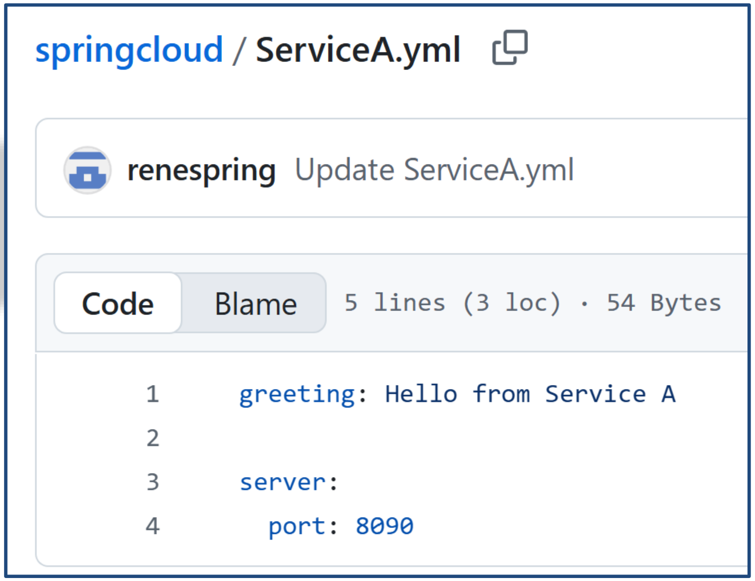


springcloud / application.yml

renespring Update application.yml

1 lines (1 loc) · 15 Bytes

```
message: CS590
```

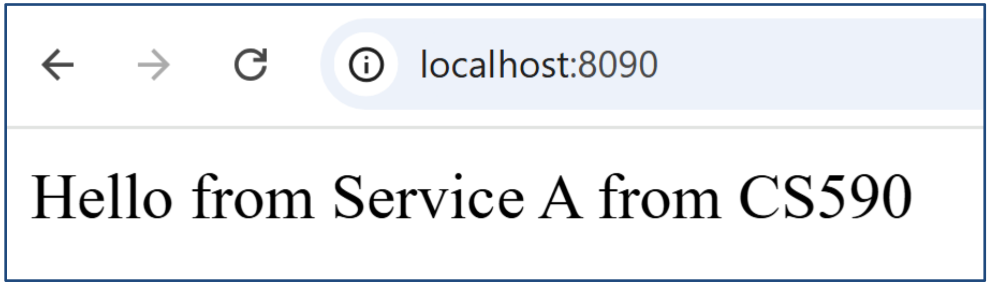


springcloud / ServiceA.yml

renespring Update ServiceA.yml

5 lines (3 loc) · 54 Bytes

```
greeting: Hello from Service A

server:
  port: 8090
```



localhost:8090

Hello from Service A from CS590

Refreshing configuration

- Use @RefreshScope and “/actuator/refresh” event

@RefreshScope

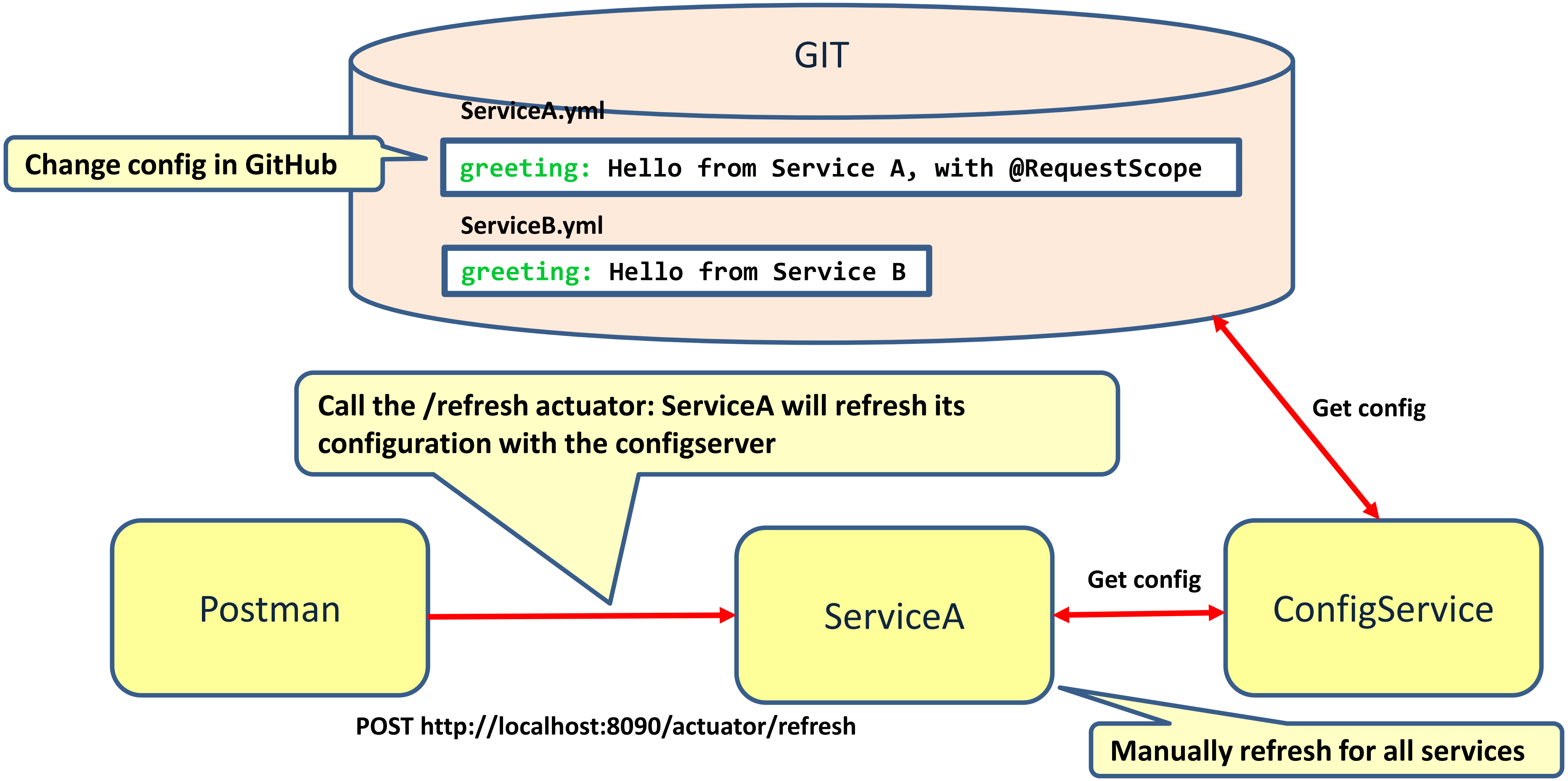
```java
@RestController
@RefreshScope
public class ServiceAController {
  @Value("${greeting}")
  private String message;

  @RequestMapping("/")
  public String getName() {
    return message;
  }
}
```

```yaml
server:
 port: 8888

spring:
 cloud:
  config:
   server:
    git:
     uri:
https://github.com/renespring/springcloud.git


management:
 endpoints:
  web:
   exposure:
    include: refresh
```

Expose the /refresh actuator

Refreshing configuration

- Use @RefreshScope and “/actuator/refresh” event



```
greeting: Hello from Service A, with @RequestScope
```

```
greeting: Hello from Service B
```

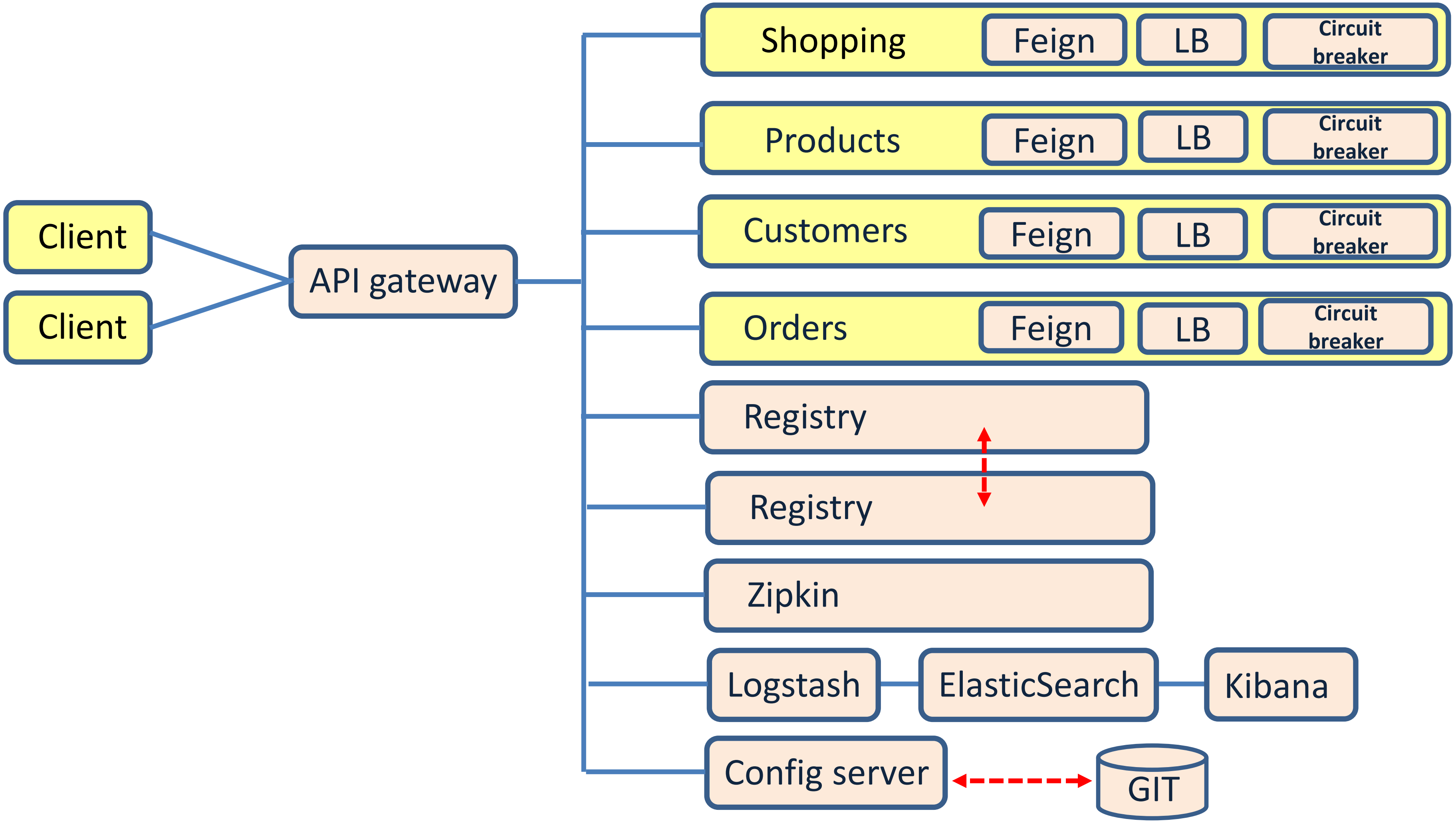
Change config in GitHub

Call the /refresh actuator: ServiceA will refresh its configuration with the configserver

POST http://localhost:8090/actuator/refresh

Manually refresh for all services

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Keep data in sync	
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK