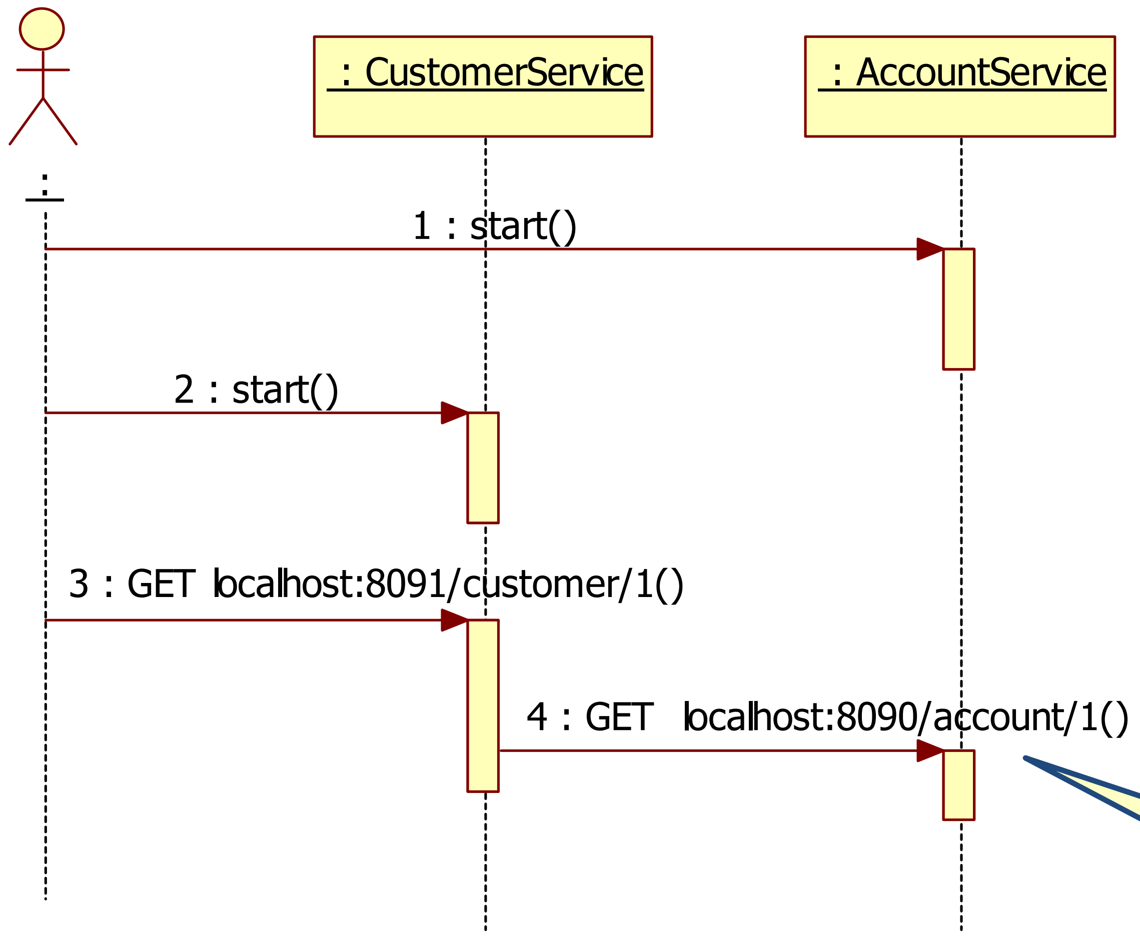
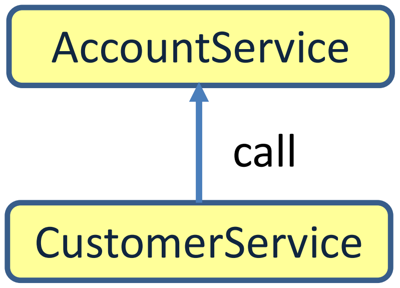
SERVICE REGISTRY: EUREKA

Service Registry

- Like the phone book for microservices
  - Services register themselves with their location and other meta-data
  - Clients can lookup other services
- Netflix Eureka

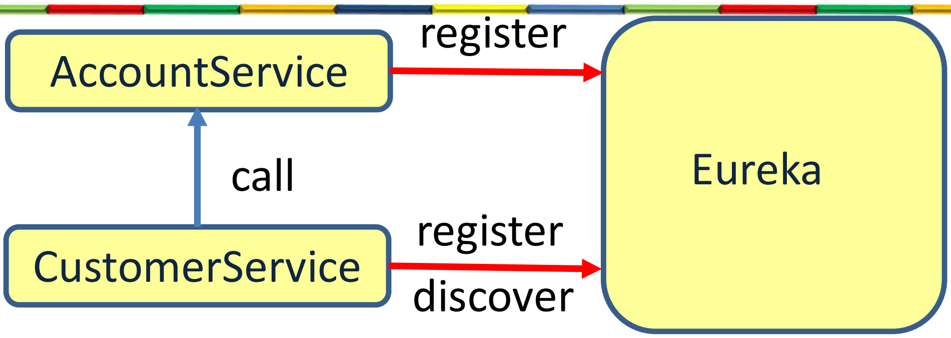
Without Eureka

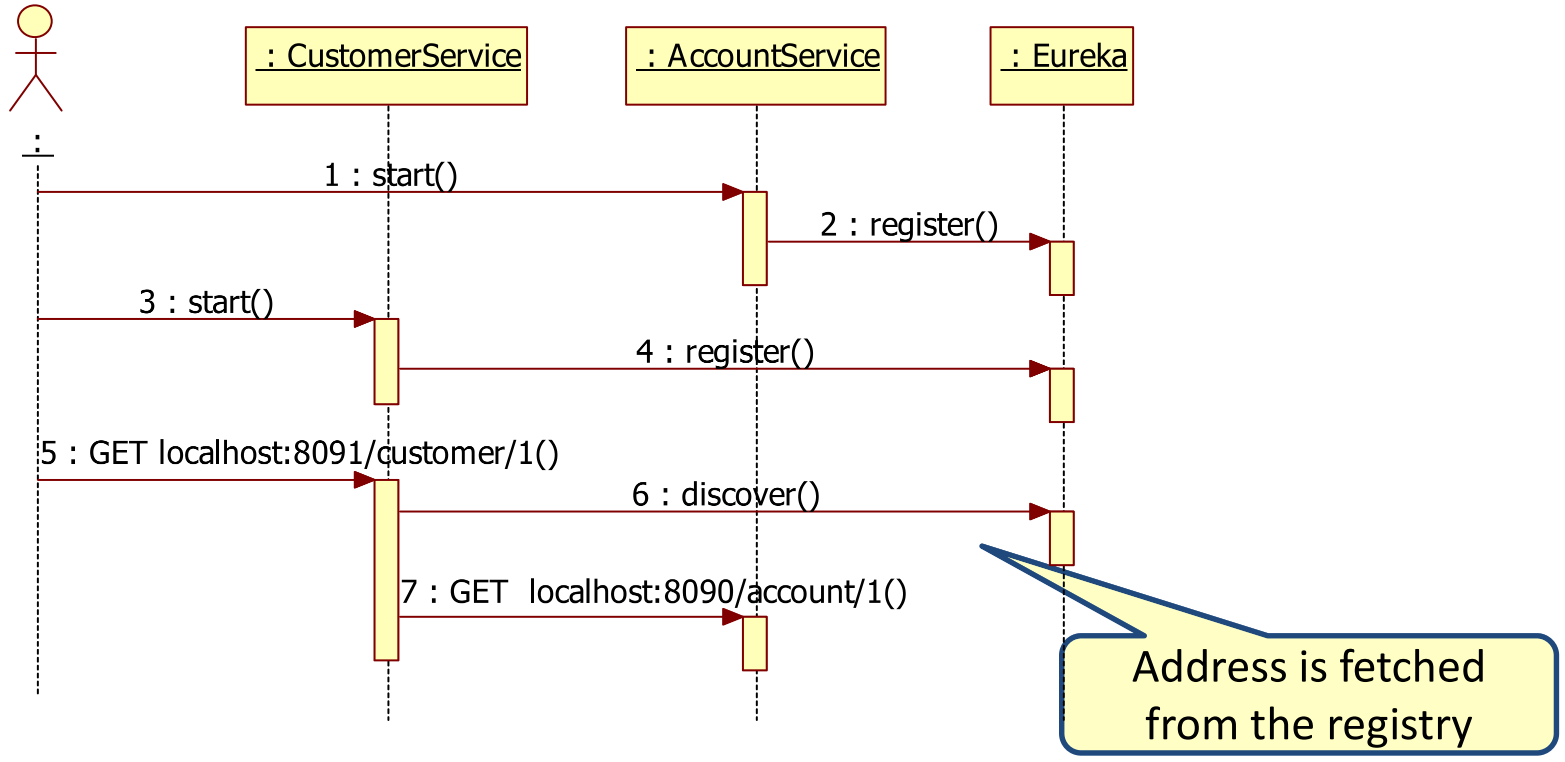

: CustomerService
: AccountService
:
1 : start()
2 : start()
3 : GET localhost:8091/customer/1()
4 : GET localhost:8090/account/1()



Address is hard coded

Using Eureka





Why service registry/discovery?

1. Loosely coupled services
   - Service consumers should not know the physical location of service instances.
     - We can easily scale up or scale down service instances
2. Increase application resilience
   - If a service instance becomes unhealthy or unavailable, the service discovery engine will remove that instance from the list of available services.

Eureka Server

```java
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

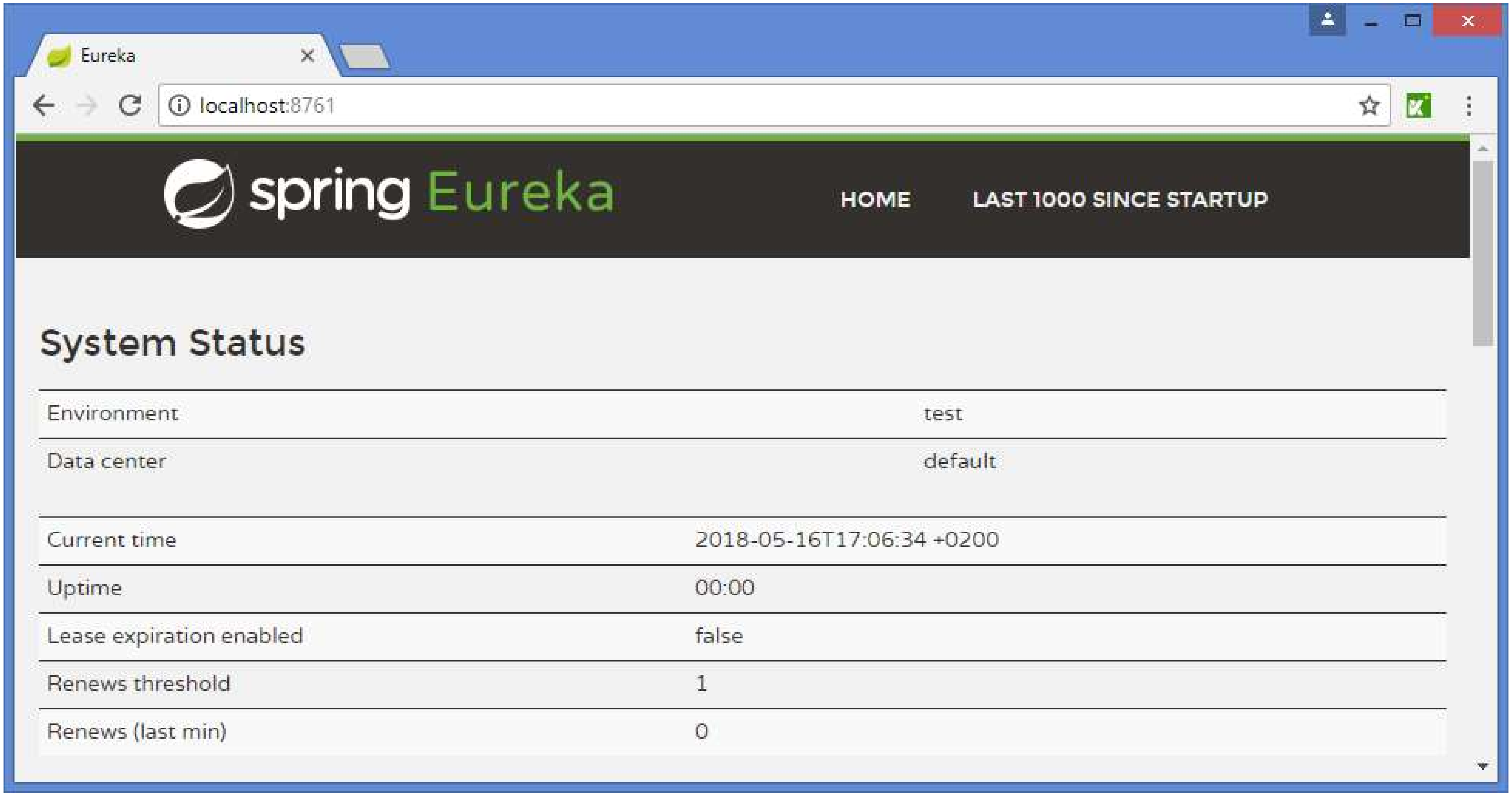
application.yml

```yaml
server:
  port: 8761

eureka:
  client:
    registerWithEureka: false   #telling the server not to register himself
    fetchRegistry: false
```

bootstrap.yml

```yaml
spring:
  application:
    name: Eureka Server
```

Running Eureka



System Status

Environment	test
Data center	default

Current time	2018-05-16T17:06:34 +0200
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

AccountService

```java
@SpringBootApplication
@EnableDiscoveryClient
public class AccountServiceApplication {

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

application.yml

```yaml
server:
  port: 8090

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

bootstrap.yml

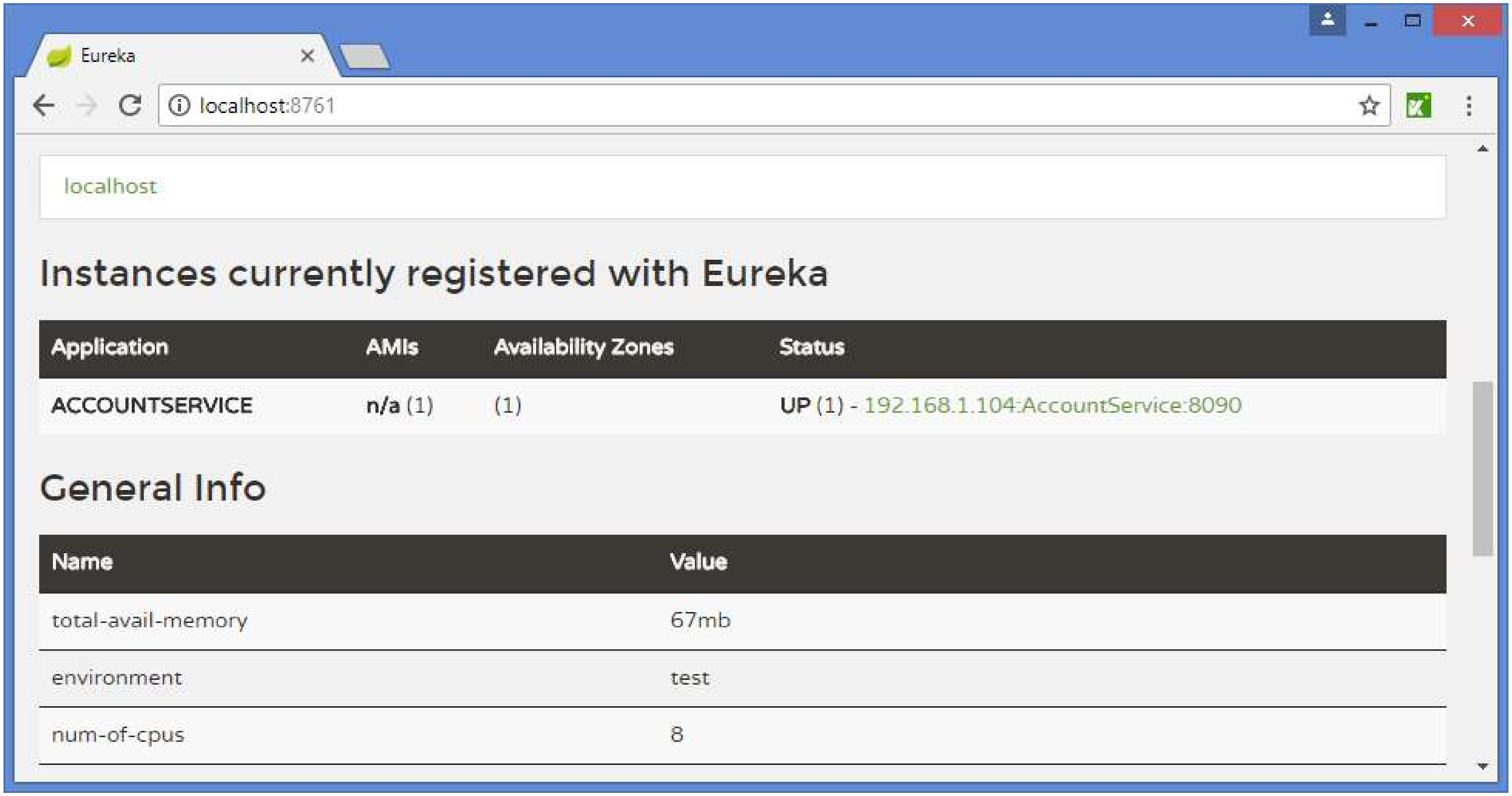
```yaml
spring:
  application:
    name: AccountService
```

AccountService

```java
@RestController
public class AccountController {
  @RequestMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    return new Account("1234", "1000.00");
  }
}
```

```java
public class Account {
  private String accountNumber;
  private String balance;
  ...
}
```

Running the AccountService



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 192.168.1.104:AccountService:8090

General Info

Name	Value
total-avail-memory	67mb
environment	test
num-of-cpus	8

CustomerService

```java
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class AccountServiceApplication {

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

Use Feign

application.yml

```yaml
server:
  port: 8091

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

bootstrap.yml

```yaml
spring:
  application:
    name: CustomerService
```

CustomerService: the controller

```
@RestController
public class CustomerController {
  @Autowired
  AccountFeignClient accountClient;

  @RequestMapping("/customer/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    Account account = accountClient.getName(customerId);
    return account;
  }

  @FeignClient("AccountService")
  interface AccountFeignClient {
    @RequestMapping("/account/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId);
  }
}
```

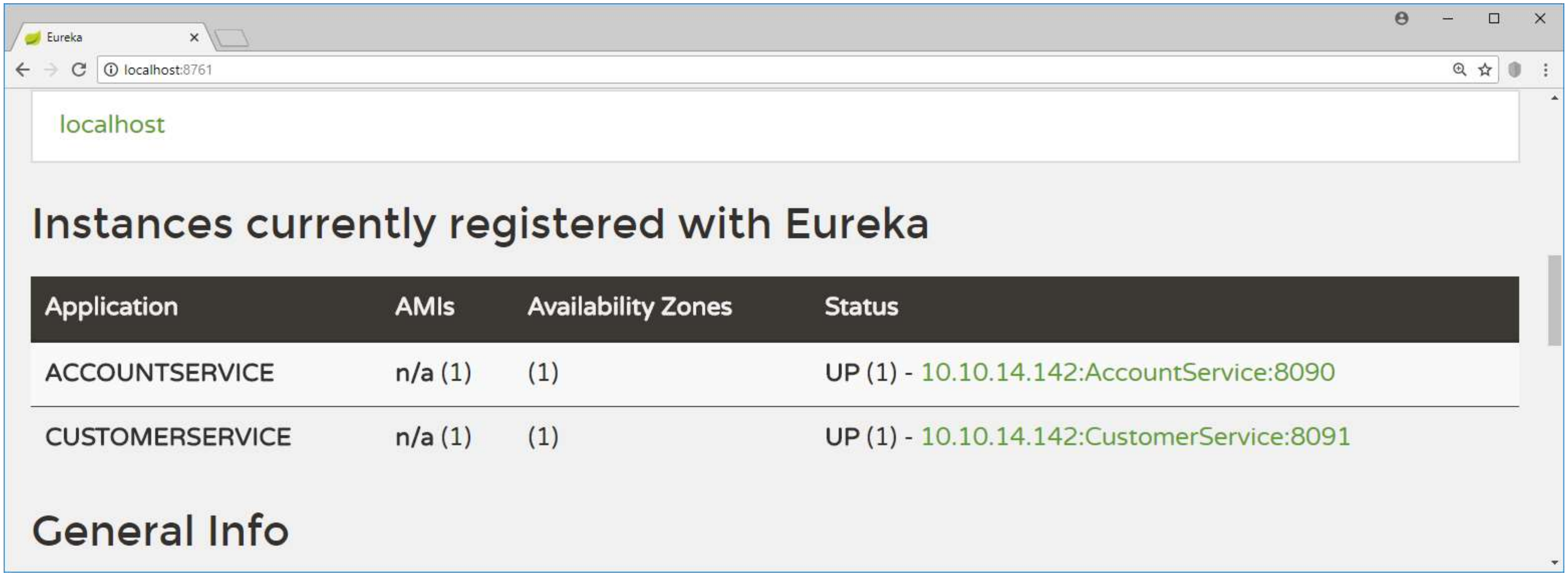
Name of the service

Use Feign to access the AccountService

application.yml

```
server:
  port: 8091
```

Running the CustomerService



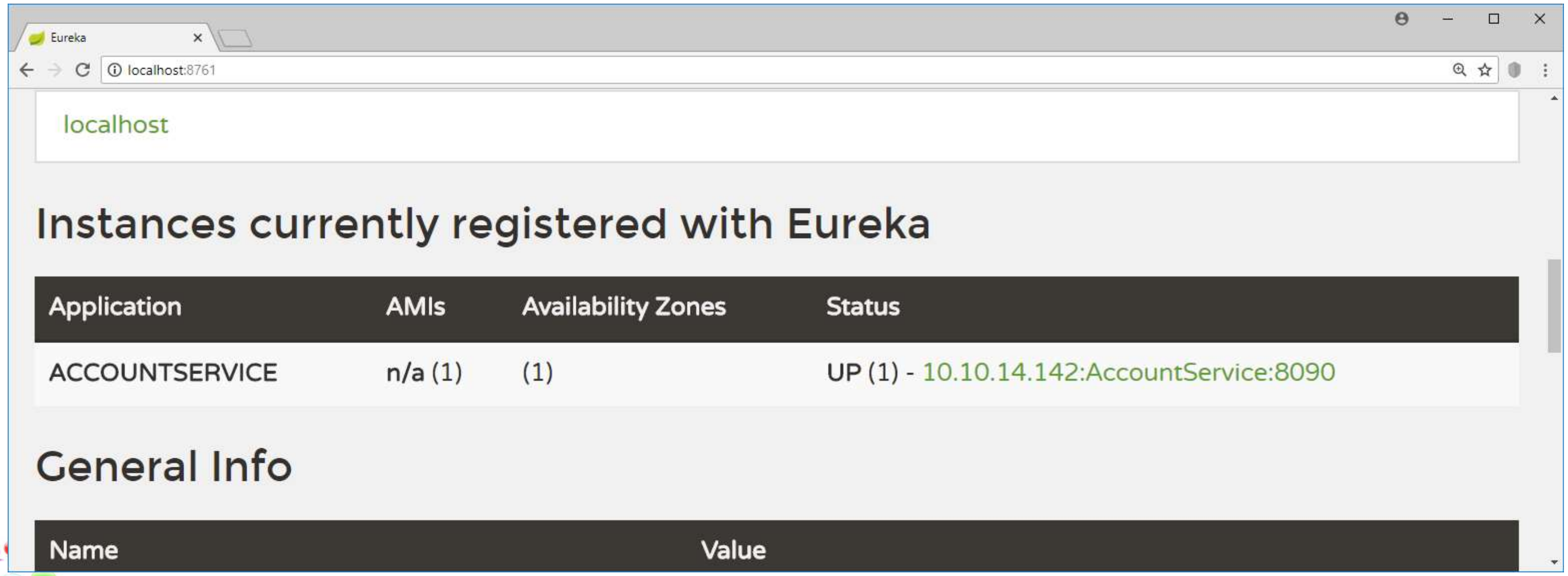
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:AccountService:8090
CUSTOMERSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:CustomerService:8091

General Info

Stopping the CustomerService

- Eureka monitors the health of registered services.
- If we stop the CustomerService, Eureka will notice that automatically



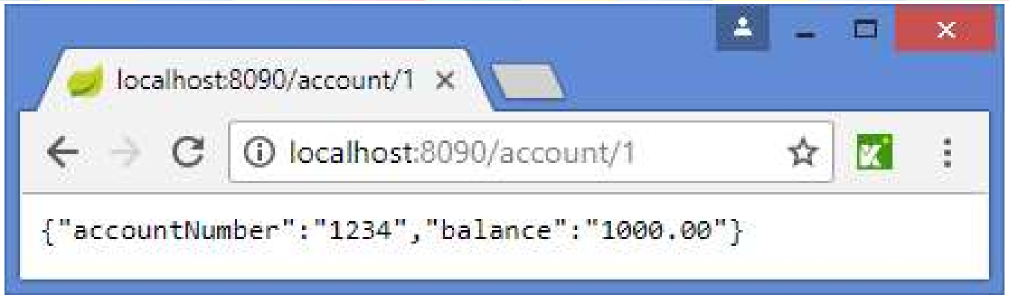
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:AccountService:8090

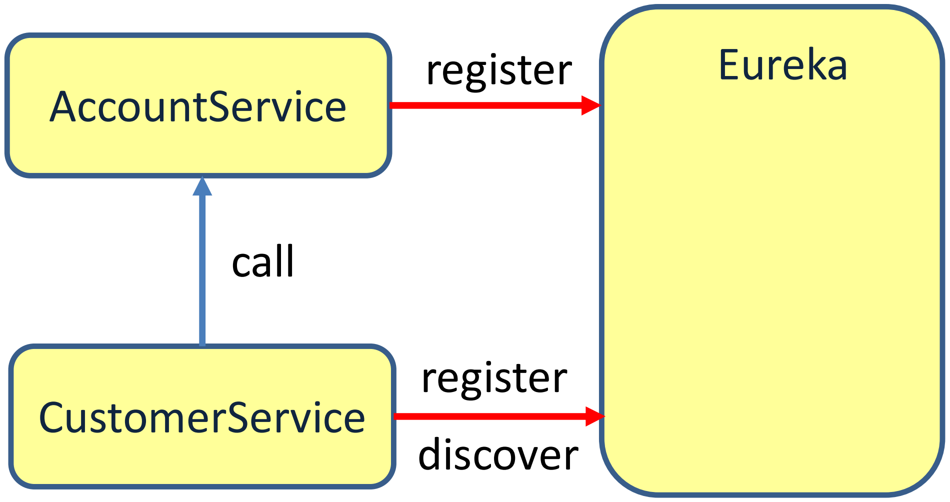
General Info

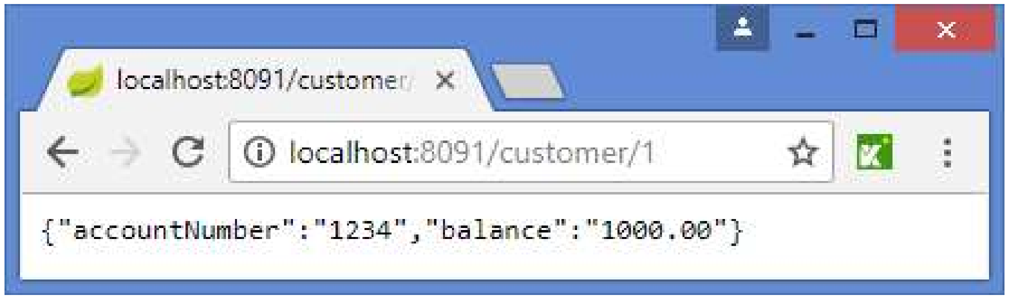
Name	Value

Using Eureka



```
{"accountNumber":"1234","balance":"1000.00"}
```





```
{"accountNumber":"1234","balance":"1000.00"}
```

Registering with Eureka

- When a service registers with Eureka, Eureka will wait for 3 successive health checks over the course of 30 seconds before the service becomes available in Eureka

Eureka high availability

- Multiple Eureka servers can be configured as such that they replicate the contents of their registries.

application.yml

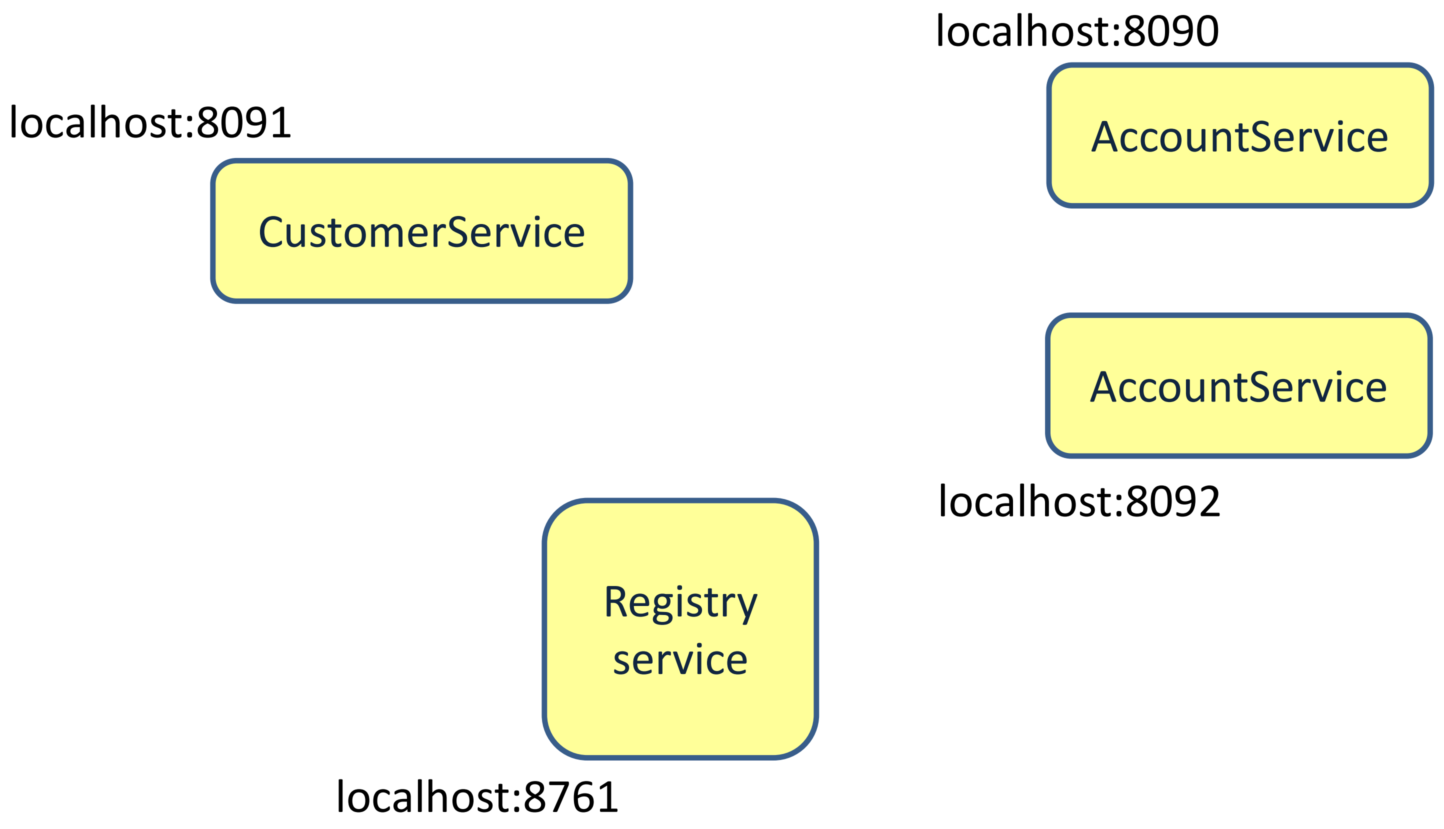
```
server:
  port: 8091

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

This can be a comma separated list of Eureka instances.
If the first instance does not respond, we try the next instance

NETFLIX
RIBBON

LOAD BALANCING: RIBBON

Running 2 AccountServices using profiles



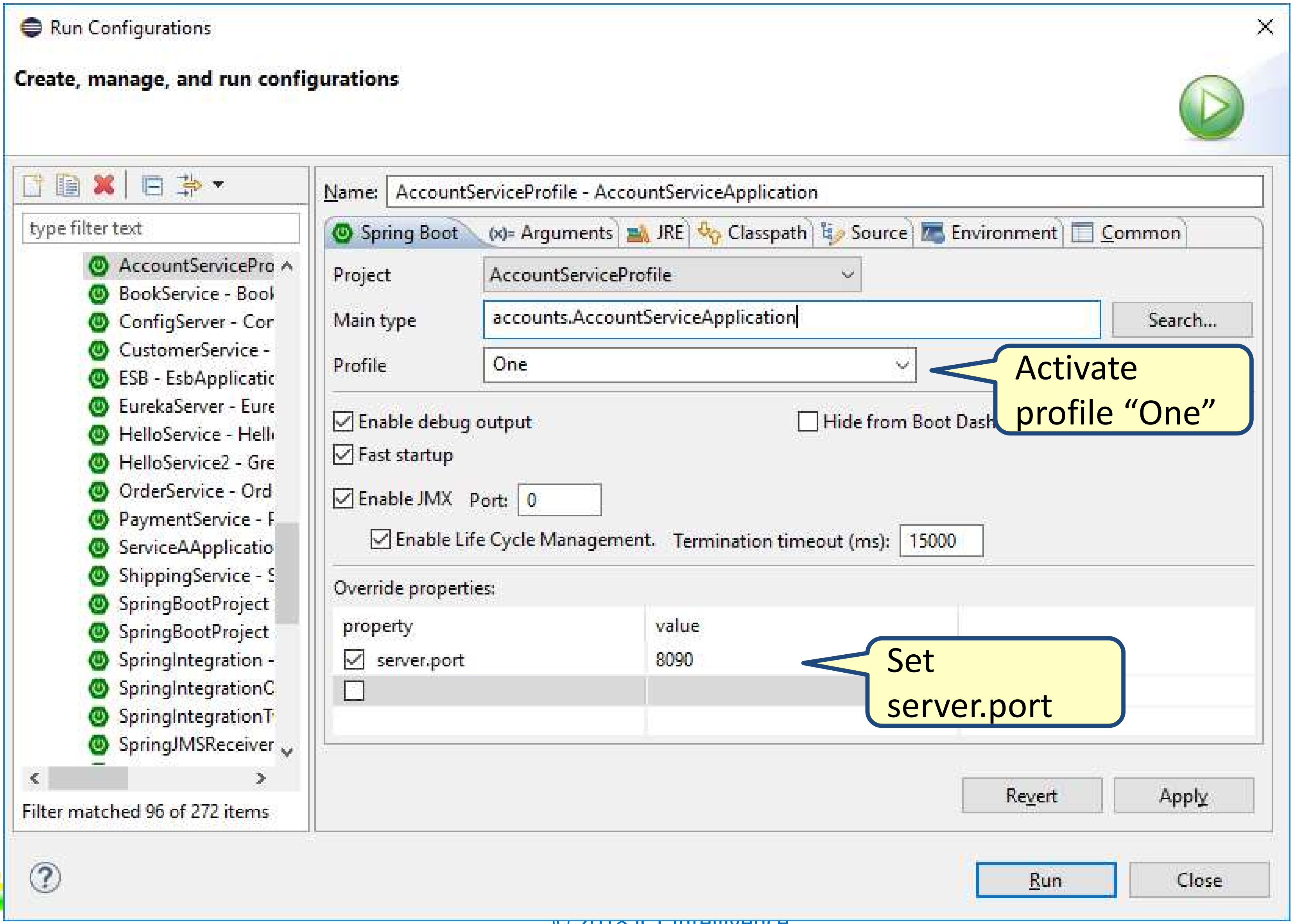
Spring Profiles

Define a profile

```
@RestController
@Profile("One")
public class AccountController1 {
  @GetMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    System.out.println("getName() on AccountController1 is called");
    return new Account("1234", "1000.00");
  }
}
```

```
@RestController
@Profile("Two")
public class AccountController2 {
  @GetMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    System.out.println("getName() on AccountController2 is called");
    return new Account("1234", "1000.00");
  }
}
```

Start the first instance



Run Configurations

Create, manage, and run configurations

type filter text

AccountServicePro
BookService - Book
ConfigServer - Cor
CustomerService -
ESB - EsbApplicatio
EurekaServer - Eure
HelloService - Hell
HelloService2 - Gre
OrderService - Ord
PaymentService - F
ServiceAApplicatio
ShippingService - S
SpringBootProject
SpringBootProject
SpringIntegration -
SpringIntegrationC
SpringIntegrationT
SpringJMSReceiver

Filter matched 96 of 272 items

Name: AccountServiceProfile - AccountServiceApplication

Spring Boot | Arguments | JRE | Classpath | Source | Environment | Common

Project: AccountServiceProfile

Main type: accounts.AccountServiceApplication

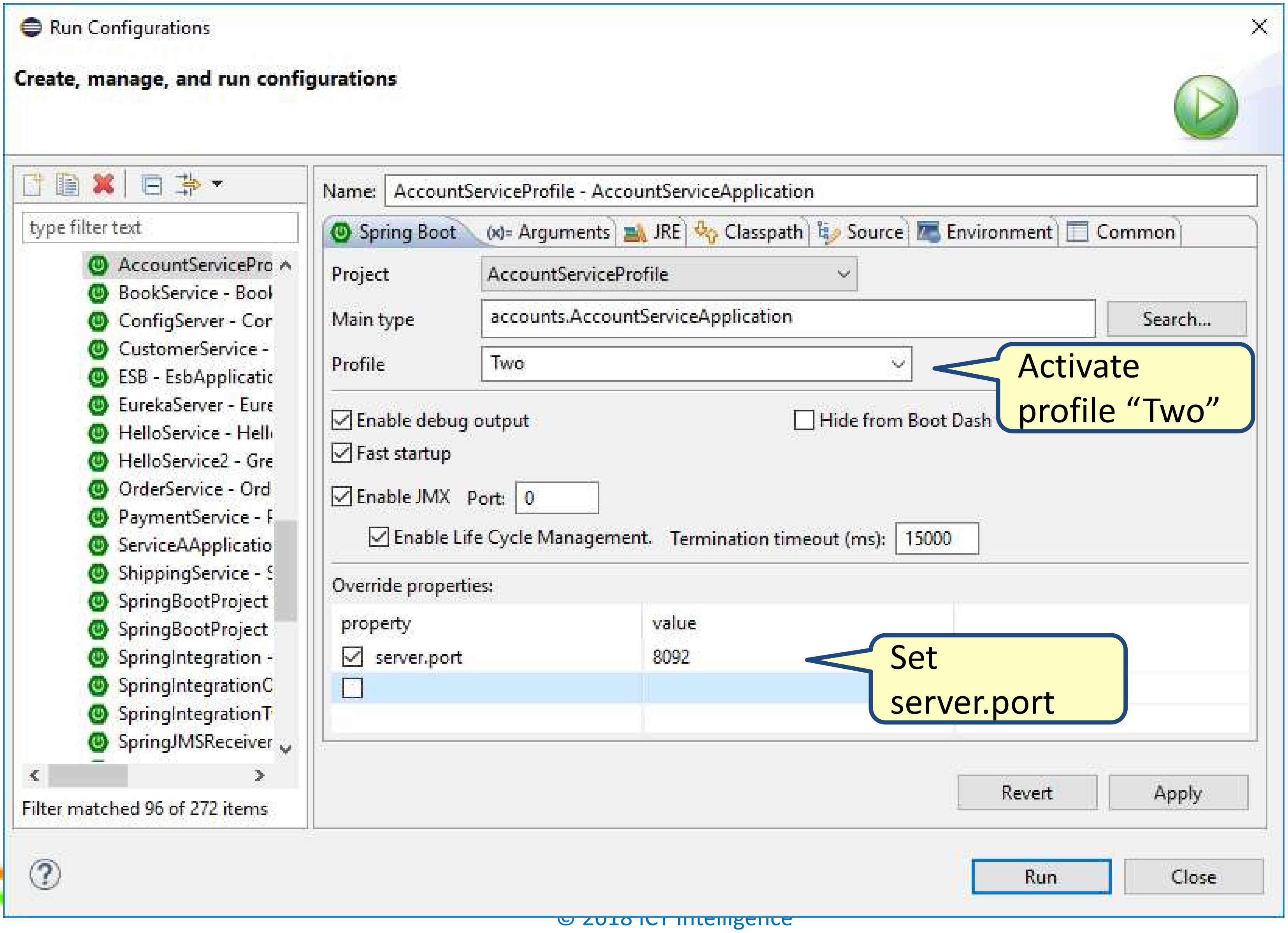
Profile: One

[✓] Enable debug output
[ ] Hide from Boot Dash
[✓] Fast startup
[✓] Enable JMX Port: 0
[✓] Enable Life Cycle Management. Termination timeout (ms): 15000

Override properties:

property	value
[✓] server.port	8090
[ ]	

Start the second instance



Run Configurations

Create, manage, and run configurations

type filter text

- AccountServicePro
- BookService - Book
- ConfigServer - Cor
- CustomerService -
- ESB - EsbApplicatic
- EurekaServer - Eure
- HelloService - Hell
- HelloService2 - Gre
- OrderService - Ord
- PaymentService - F
- ServiceAApplicatio
- ShippingService - S
- SpringBootProject
- SpringBootProject
- SpringIntegration -
- SpringIntegrationC
- SpringIntegrationT
- SpringJMSReceiver

Filter matched 96 of 272 items

Name: AccountServiceProfile - AccountServiceApplication

Spring Boot | Arguments | JRE | Classpath | Source | Environment | Common

Project: AccountServiceProfile

Main type: accounts.AccountServiceApplication

Profile: Two

[✓] Enable debug output
[ ] Hide from Boot Dash
[✓] Fast startup
[✓] Enable JMX Port: 0
[✓] Enable Life Cycle Management. Termination timeout (ms): 15000

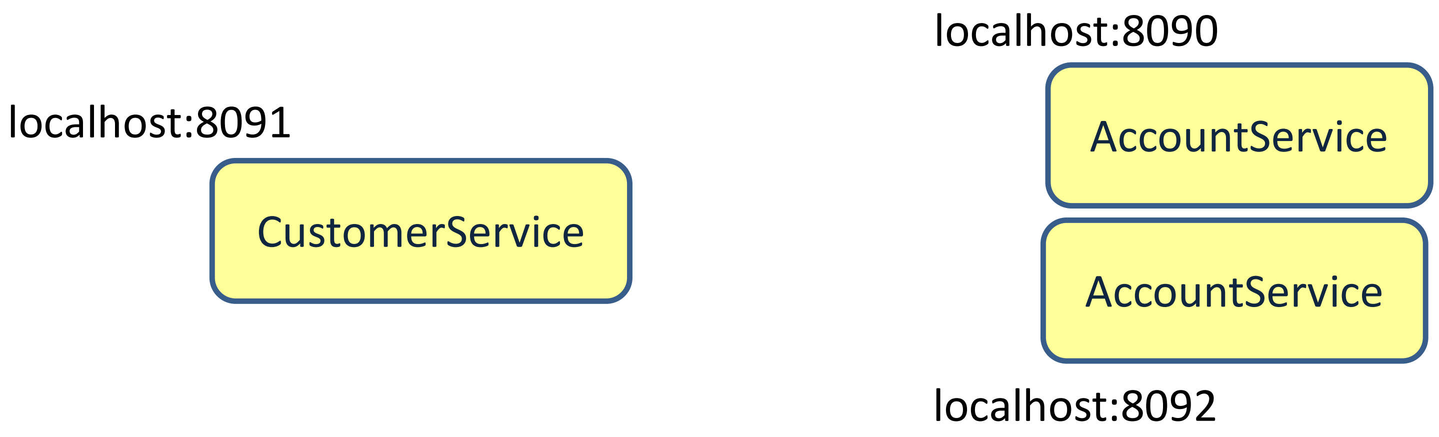
Override properties:

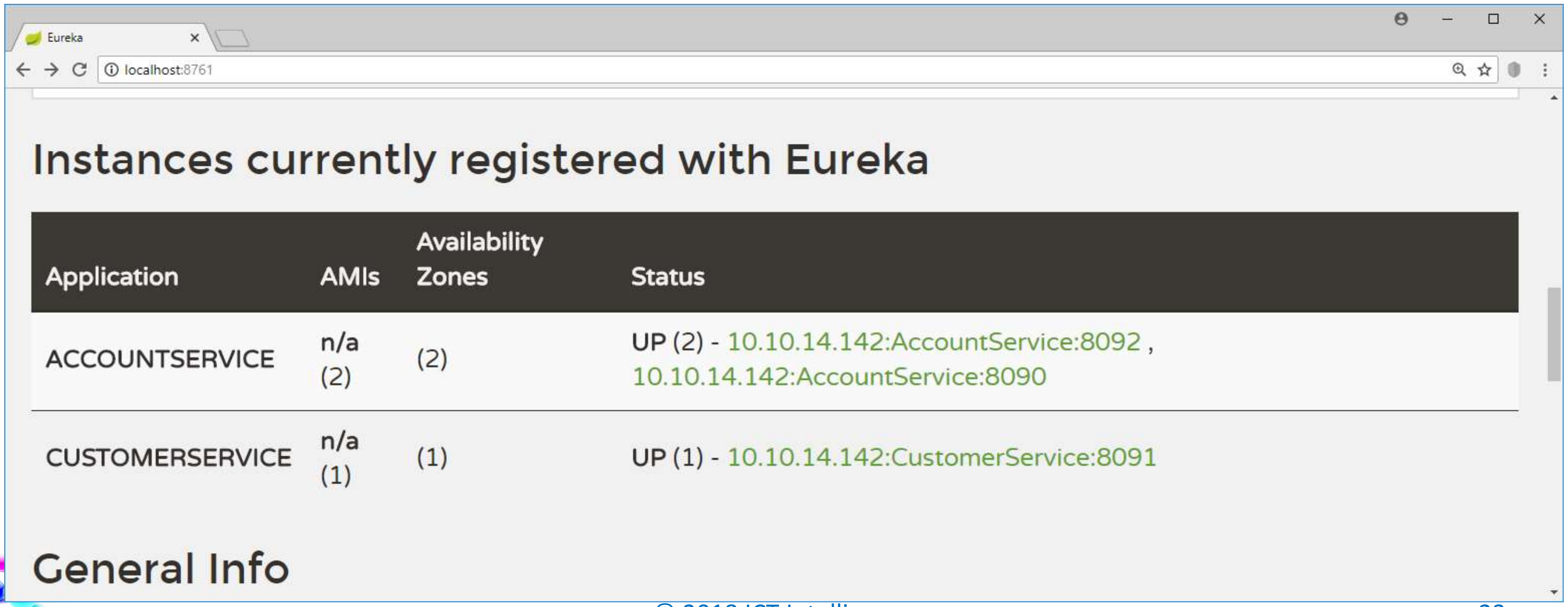
property	value
[✓] server.port	8092
[ ]	

Activate profile "Two"

Set server.port

2 instances of AccountService



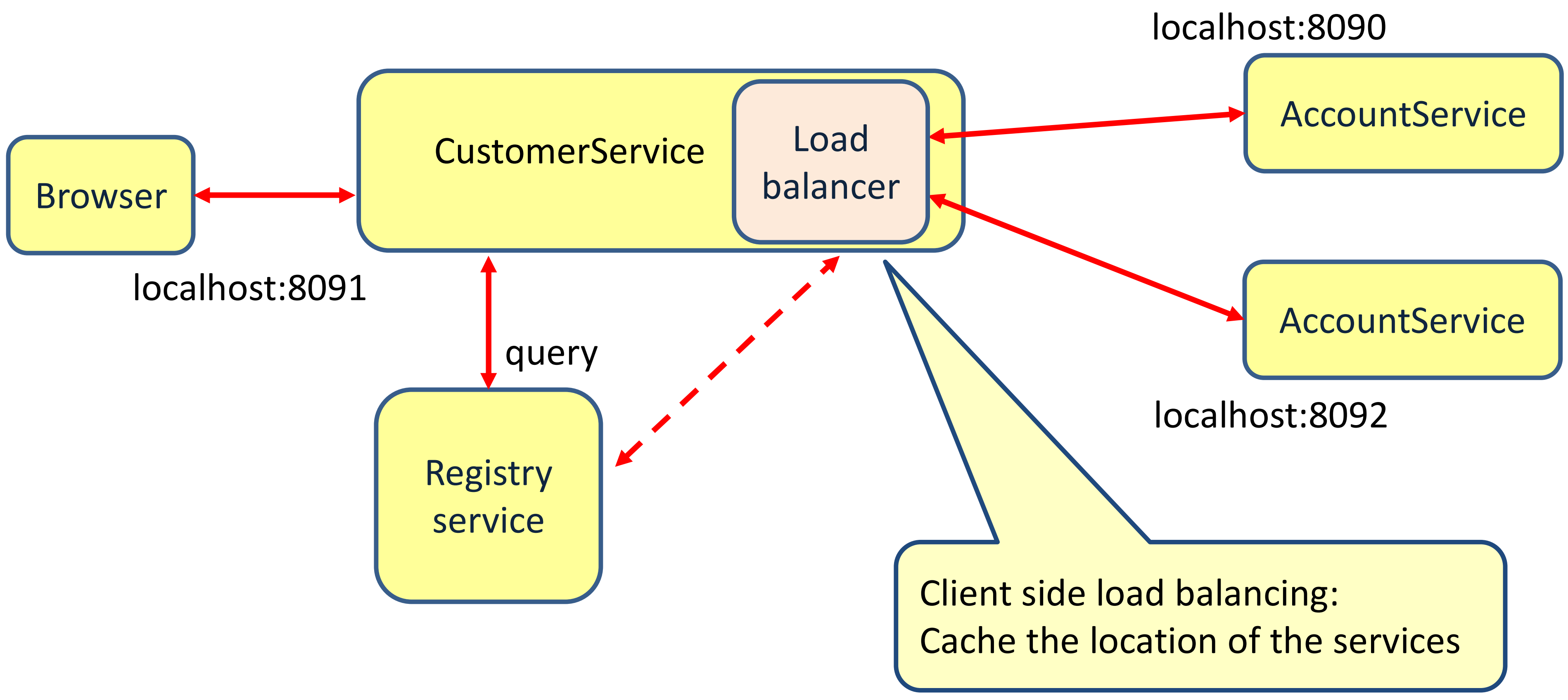


Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (2)	(2)	UP (2) - 10.10.14.142:AccountService:8092 , 10.10.14.142:AccountService:8090
CUSTOMERSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:CustomerService:8091

General Info

Load balancer



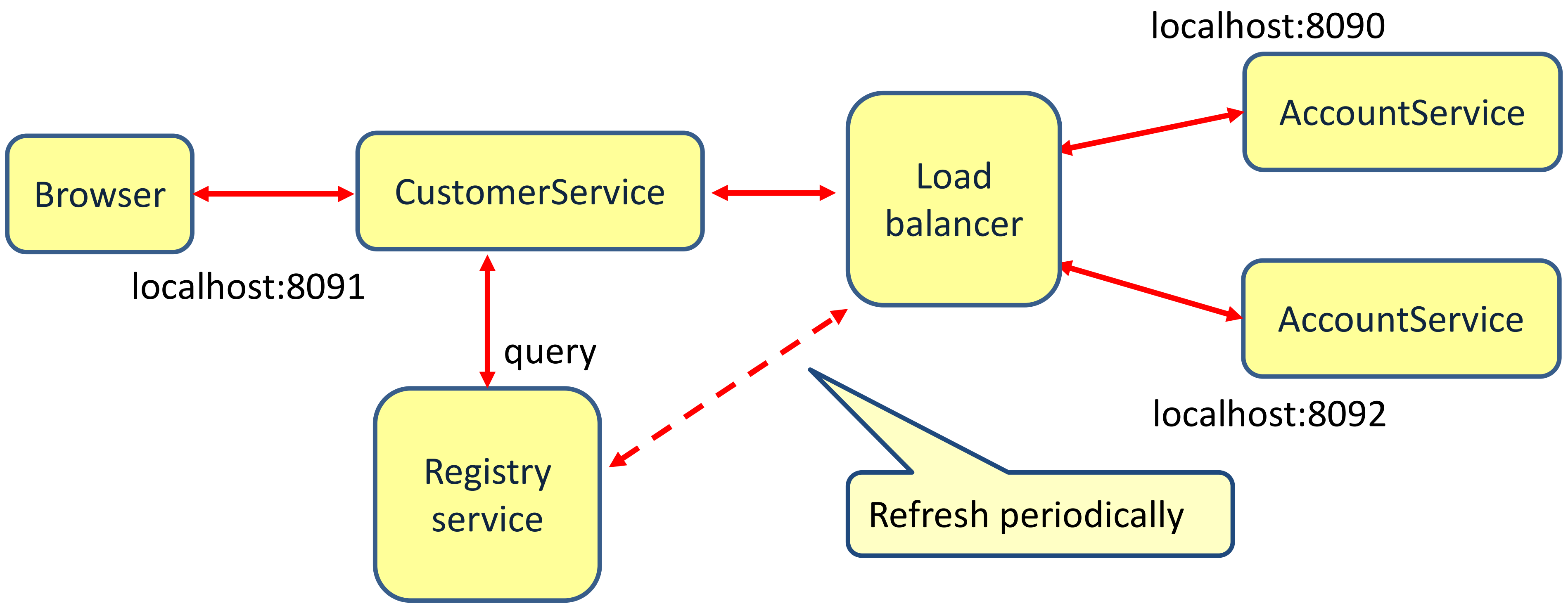
CustomerService calls AccountService

```
@RestController
public class CustomerController {
  @Autowired
  AccountFeignClient accountClient;

  @RequestMapping("/customer/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    Account account = accountClient.getName(customerId);
    return account;
  }

  @FeignClient("AccountService")
  @RibbonClient(name="AccountService")
  interface AccountFeignClient {
    @RequestMapping("/account/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId);
   }
}
```

Use Feign to call another service

Use Ribbon for load balancing

Load balancer



- The load balancer will use Round Robin by default.
- If you stop one instance of AccountService, automatically the other instance will be used.
- If you start the second instance again, it will use Round Robin again.