

Finding Services Using Service Discovery



Outline



Service Discovery

Discovering Services with Spring Cloud

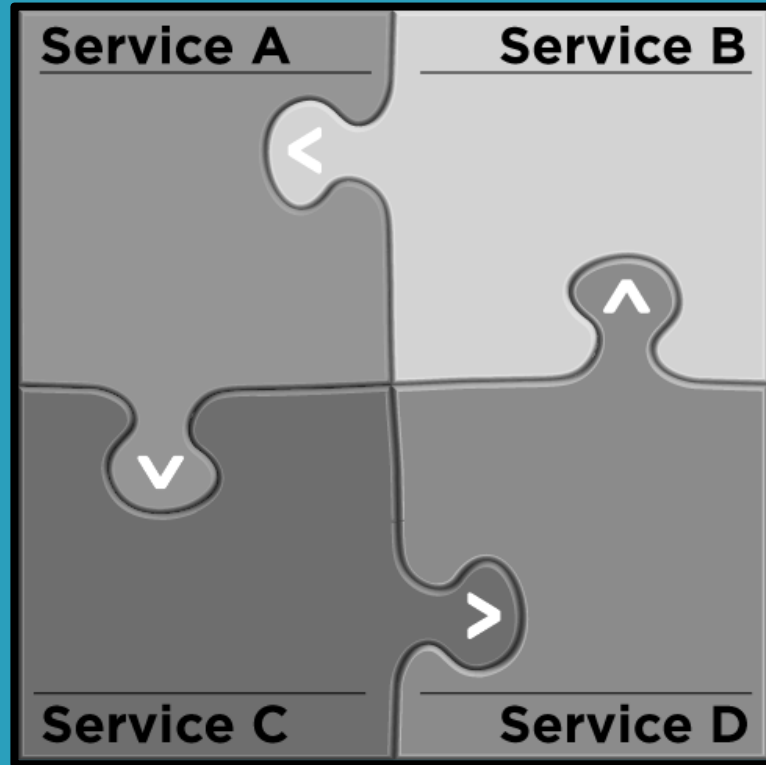
- Using Eureka client and server
- Configuration
- Health & High Availability
- Dashboard
- AWS Support



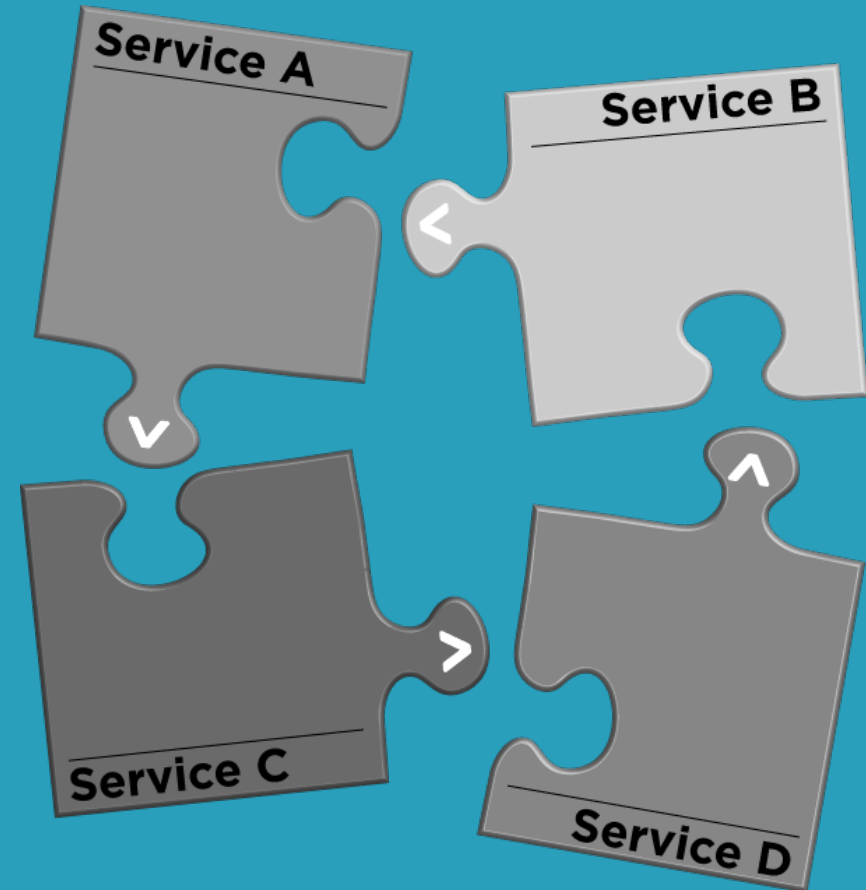
What is **Service
Discovery** and why do we
need it?



Changes in the Way We Develop Software



From single applications



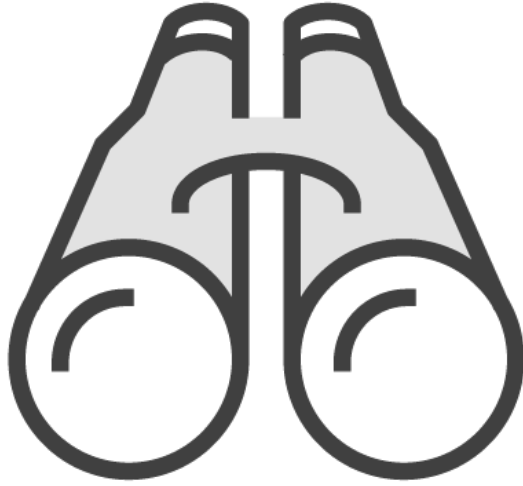
To individually deployable services



The Problem: How Does One Service Locate Another?



Application Service A



Locate?



Application Service B

The Simple Approach: Via Configuration

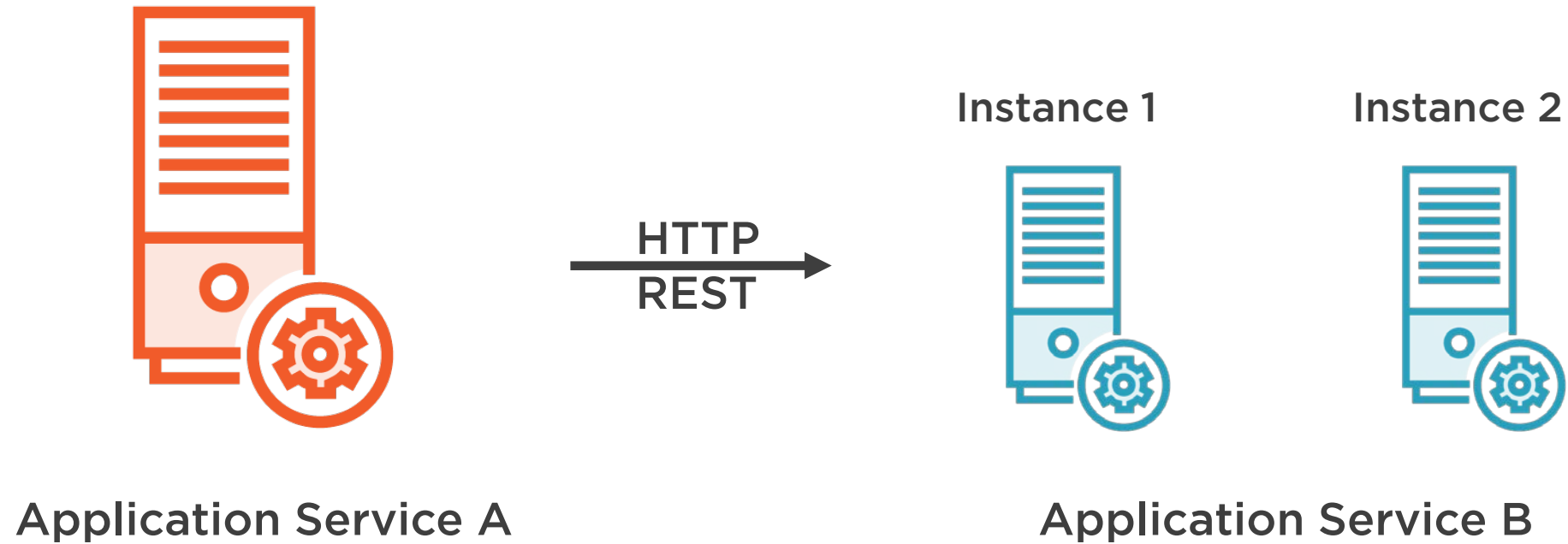


Application Service A

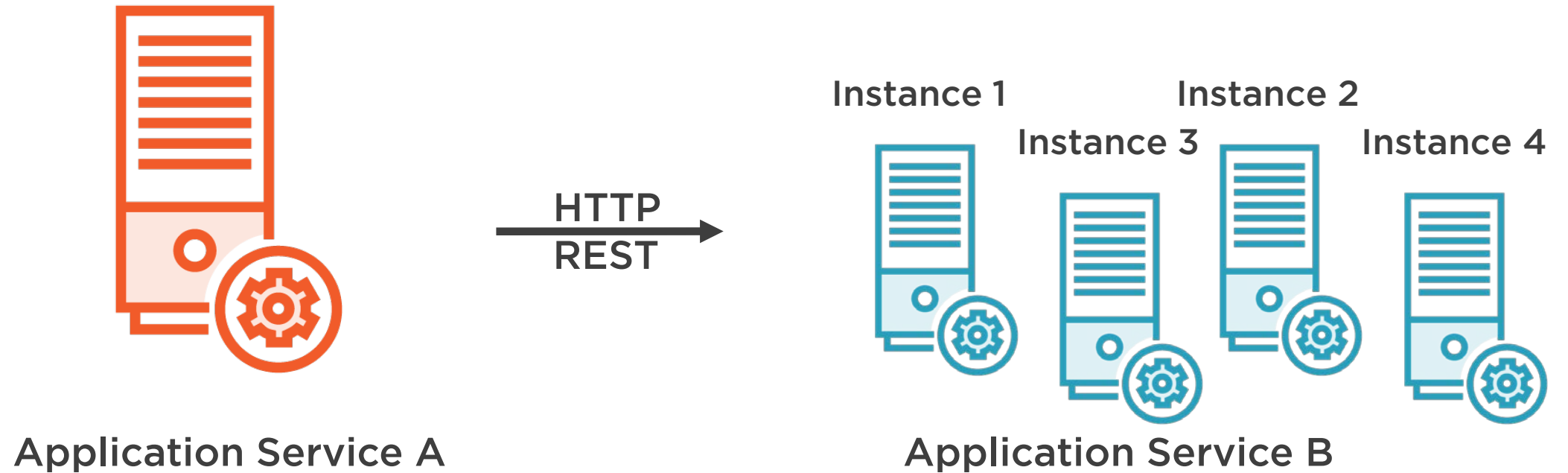
Configuration File
B: <http://1.2.3.4:6500/>

Application Service B

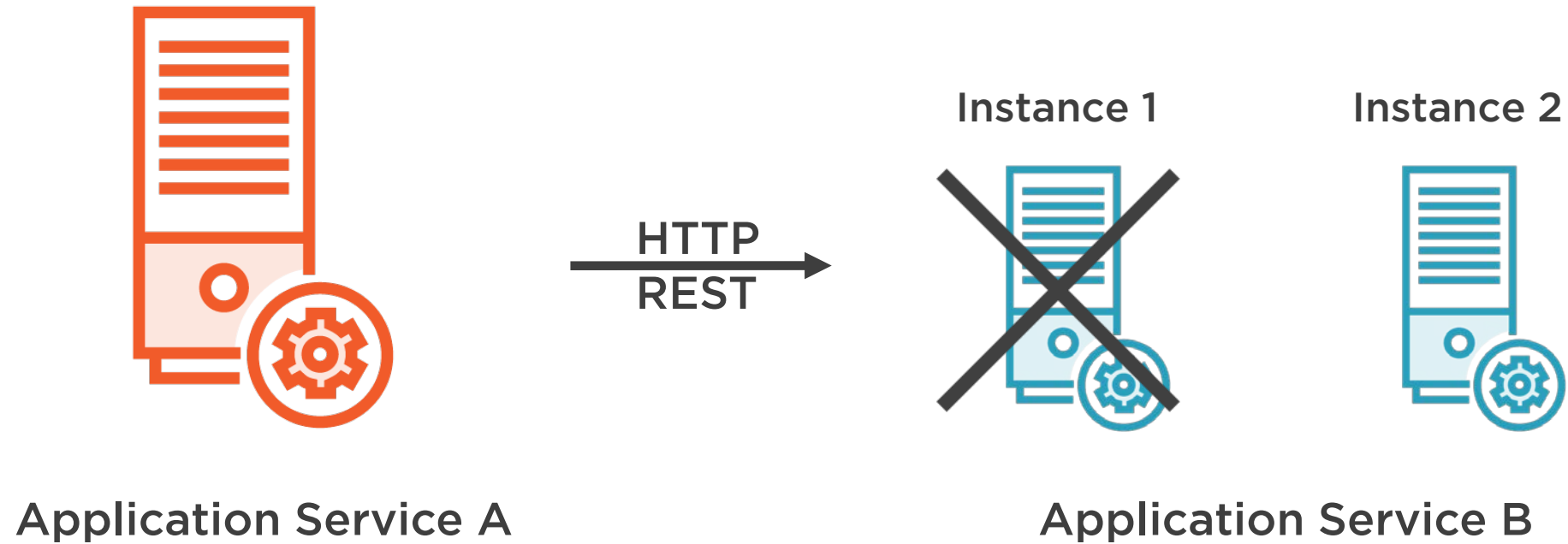
Multiple Instances



Instances Come and Go in Response to Demand



Instances Fail



The simple approach
is far **too static**
(frozen in time) for
the cloud!





Service discovery provides

- A way for a service to register itself
- A way for a service to deregister itself
- A way for a client to find other services
- A way to check the health of a service and remove unhealthy instances



Discovering Services With Spring Cloud

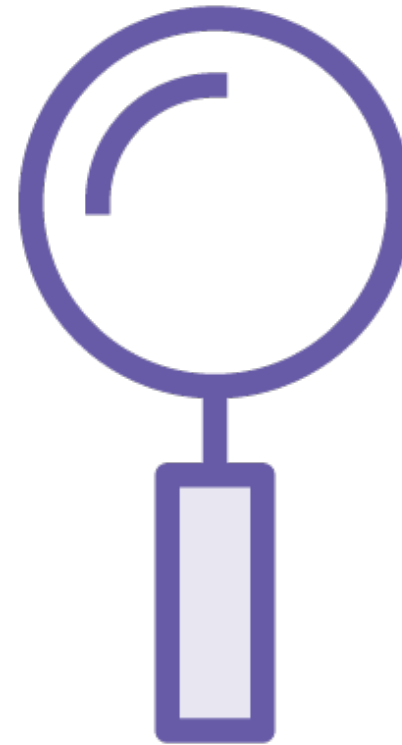


discover services with:

Spring Cloud Consul

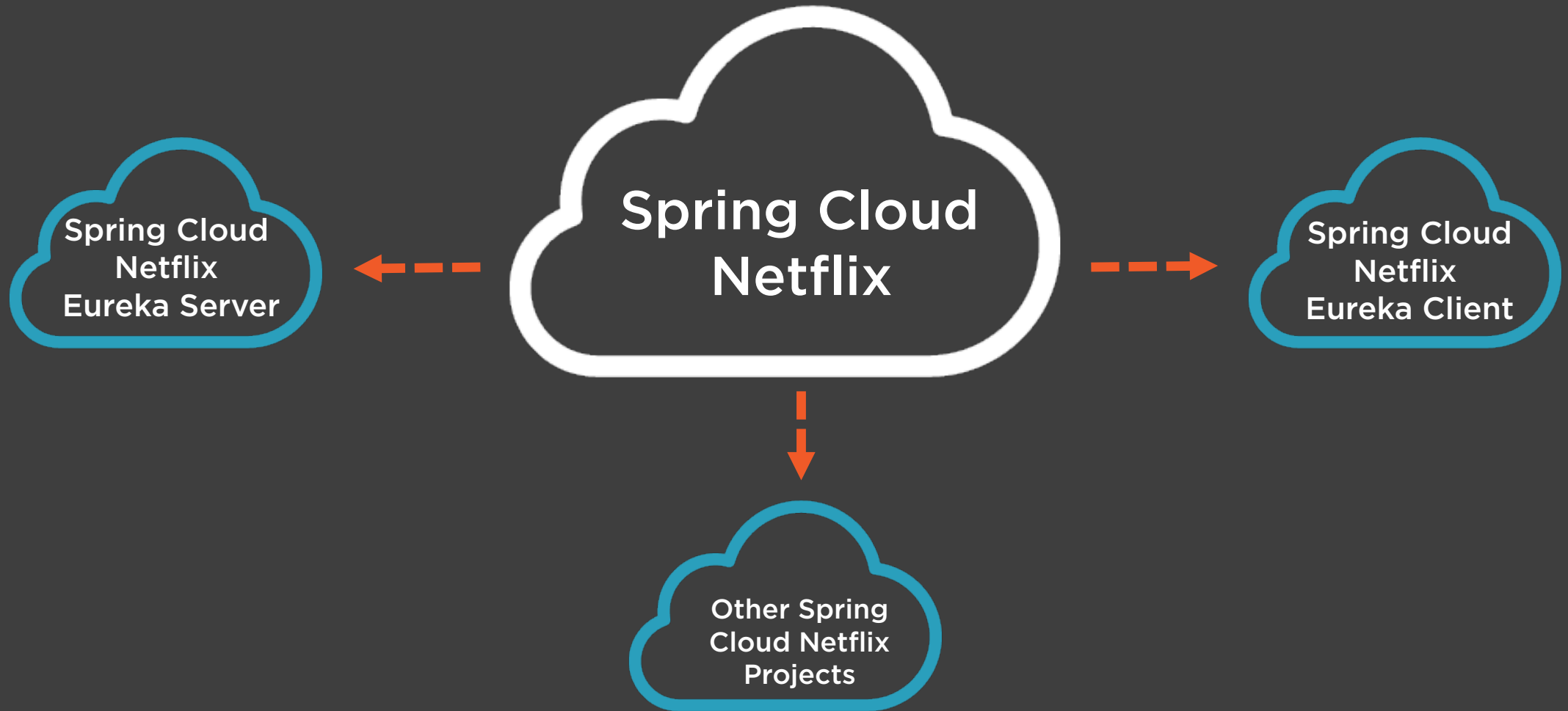
Spring Cloud Zookeeper

Spring Cloud Netflix



Netflix OSS + Spring + Spring Boot
=
Spring Cloud Netflix





Key Components in Service Discovery

Discovery Server



Service



Client



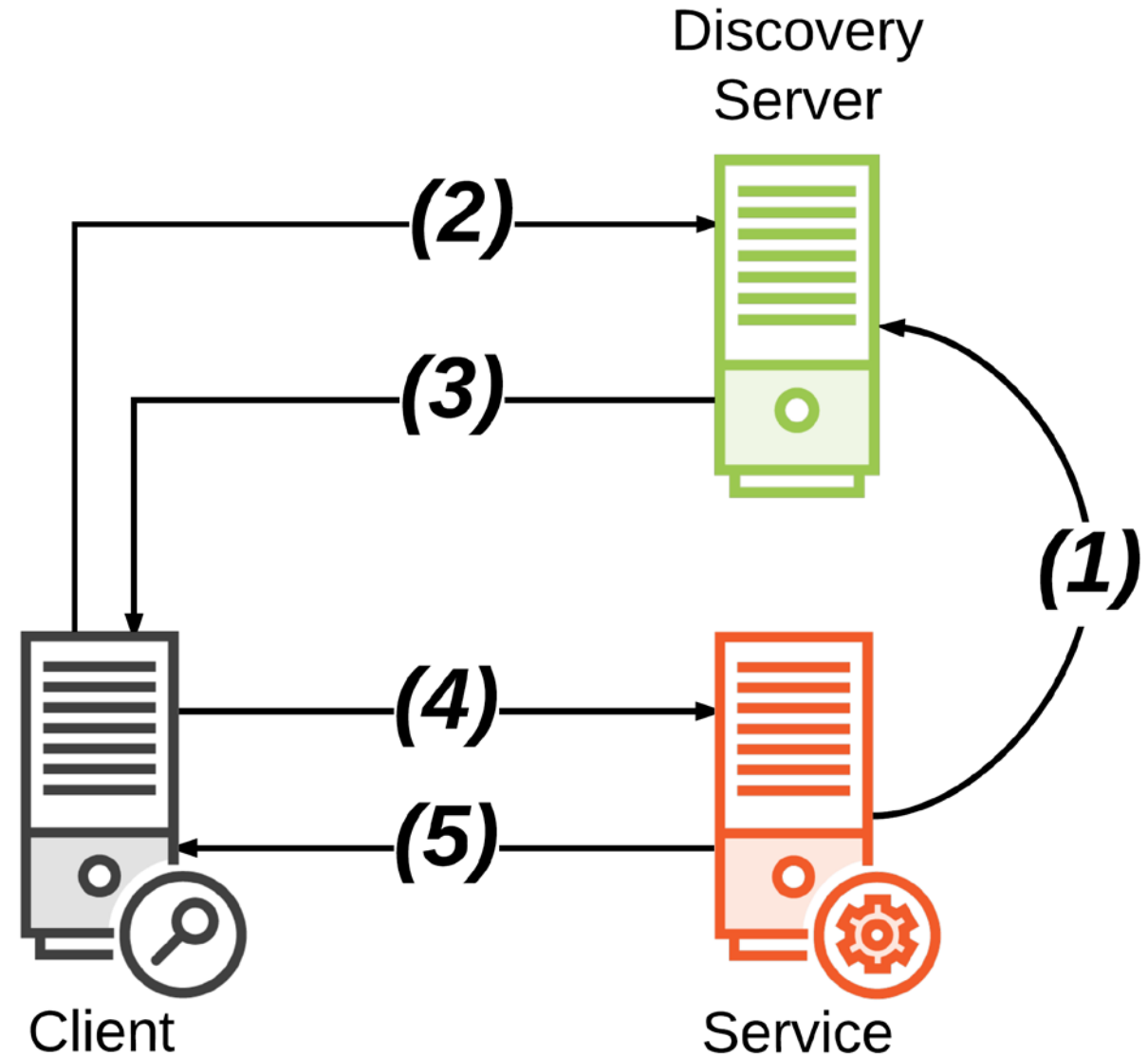
(1) Service registers location

(2) Client looks up service location

(3) Discovery server sends back location

(4) Client requests service at location

(5) Service sends response



Discovery



Server

An actively managed registry of service locations

Source of truth

One or more instances

Spring Cloud Project:

- Spring Cloud Eureka Server



Using Spring Cloud Eureka Server

pom.xml

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Camden.SR2</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```



Using Spring Cloud Eureka Server

pom.xml

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



Using Spring Cloud Eureka Server

application.properties

```
spring.application.name=discovery-server
```

OR

application.yml

```
spring:  
  application:  
    name: discovery-server
```



Using Spring Cloud Eureka Server

Application.java

```
@SpringBootApplication
@EnableEurekaServer
public class Application {

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



Demo



Creating and starting a discovery server



Application



Service

Provides some application functionality

The receiver of requests

A dependency of other service(s)

One or more instances

User of the discovery client

- Register
- Deregister

Using Spring Cloud Eureka Client in a Service

pom.xml

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Camden.SR2</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```



Using Spring Cloud Eureka Client in a Service

pom.xml

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



Using Spring Cloud Eureka Client in a Service

application.properties

```
spring.application.name=service  
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

application.yml

OR

```
spring:  
  application:  
    name: service  
eureka:  
  client:  
    service-url:  
      defaultZone: http://localhost:8761/eureka
```



Using Spring Cloud Eureka Client in a Service

Application.java

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {

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



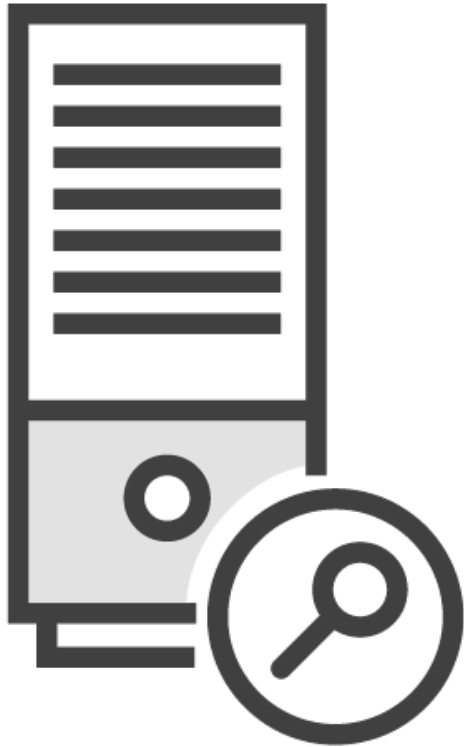
Demo



Creating a discoverable service



Application



Client

Calls another application service to implement its functionality

The issuer of requests

Depends on other service(s)

User of the discovery client

- Find service locations

Using Spring Cloud Eureka Client in an Application Client

pom.xml

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Camden.SR2</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```



Using Spring Cloud Eureka Client in an Application Client

pom.xml

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



Using Spring Cloud Eureka Client in an Application Client

application.properties

```
spring.application.name=client
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
eureka.client.register-with-eureka=false
```

application.yml

OR

```
spring:
  application:
    name: client
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
    register-with-eureka: false
```



Using Spring Cloud Eureka Client in an Application Client

Application.java

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {

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



Discovering Services as a Client: Two Options

@Inject
EurekaClient client

@Inject
DiscoveryClient client

* Spring DiscoveryClient



```
InstanceInfo instance =  
    eurekaClient.getNextServerFromEureka(  
        "service-id", false);  
  
String baseUrl = instance.getHomePageUrl();
```

Using the `EurekaClient`

`getNextServerFromEureka` – pick the next instance using round-robin

- 1st argument – virtual host name or service id of service to call
 - By default, apps use the `spring.application.name` as their virtual hostname when registering
- 2nd argument – whether or not this is a secure request



```
List<ServiceInstance> instances = client.getInstance("service-id");
```

```
String baseUrl = instances.get(0).getUri().toString();
```

Using the Spring `DiscoveryClient`

`getInstances` - return all instances of the given service id

- 1st argument - virtual host name or service id of service to call
 - By default, apps use the `spring.application.name` as their virtual hostname when registering



Demo



Creating a client that can discover services



Spring Cloud Eureka Dashboard





Enabled by default

- `eureka.dashboard.enabled=true`

Displays useful metadata and service status



Demo



Spring Cloud Eureka web dashboard



Configuring Spring Cloud Eureka



Areas of Configuration

1

`eureka.server.*`

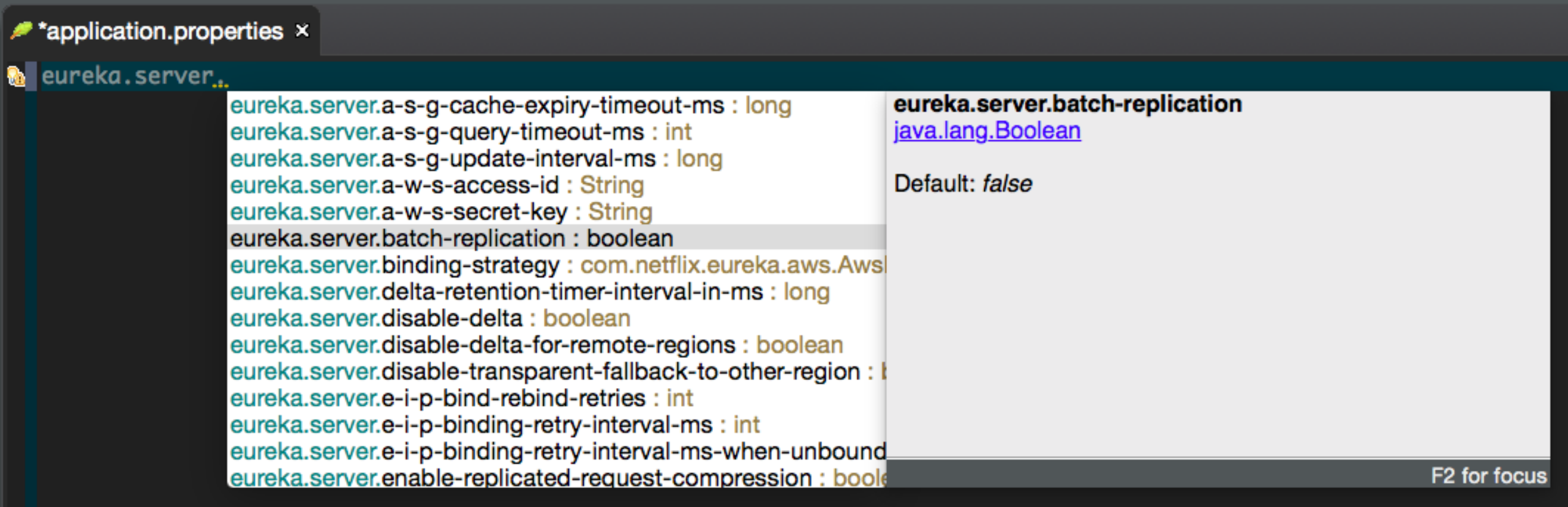
2

`eureka.client.*`

3

`eureka.instance.*`





The screenshot shows an IDE window titled '*application.properties' with a sub-tab 'eureka.server...'. The main text area displays a list of configuration properties for the Eureka server, including timeouts, intervals, access IDs, secrets, replication settings, binding strategies, and delta retention. A tooltip is visible for the 'eureka.server.batch-replication' property, showing its type as 'java.lang.Boolean' and its default value as 'false'.

```
eureka.server.a-s-g-cache-expiry-timeout-ms : long
eureka.server.a-s-g-query-timeout-ms : int
eureka.server.a-s-g-update-interval-ms : long
eureka.server.a-w-s-access-id : String
eureka.server.a-w-s-secret-key : String
eureka.server.batch-replication : boolean
eureka.server.binding-strategy : com.netflix.eureka.aws.Aws
eureka.server.delta-retention-timer-interval-in-ms : long
eureka.server.disable-delta : boolean
eureka.server.disable-delta-for-remote-regions : boolean
eureka.server.disable-transparent-fallback-to-other-region : b
eureka.server.e-i-p-bind-rebind-retries : int
eureka.server.e-i-p-binding-retry-interval-ms : int
eureka.server.e-i-p-binding-retry-interval-ms-when-unbound
eureka.server.enable-replicated-request-compression : boole
```

eureka.server.batch-replication
[java.lang.Boolean](#)
Default: *false*

F2 for focus

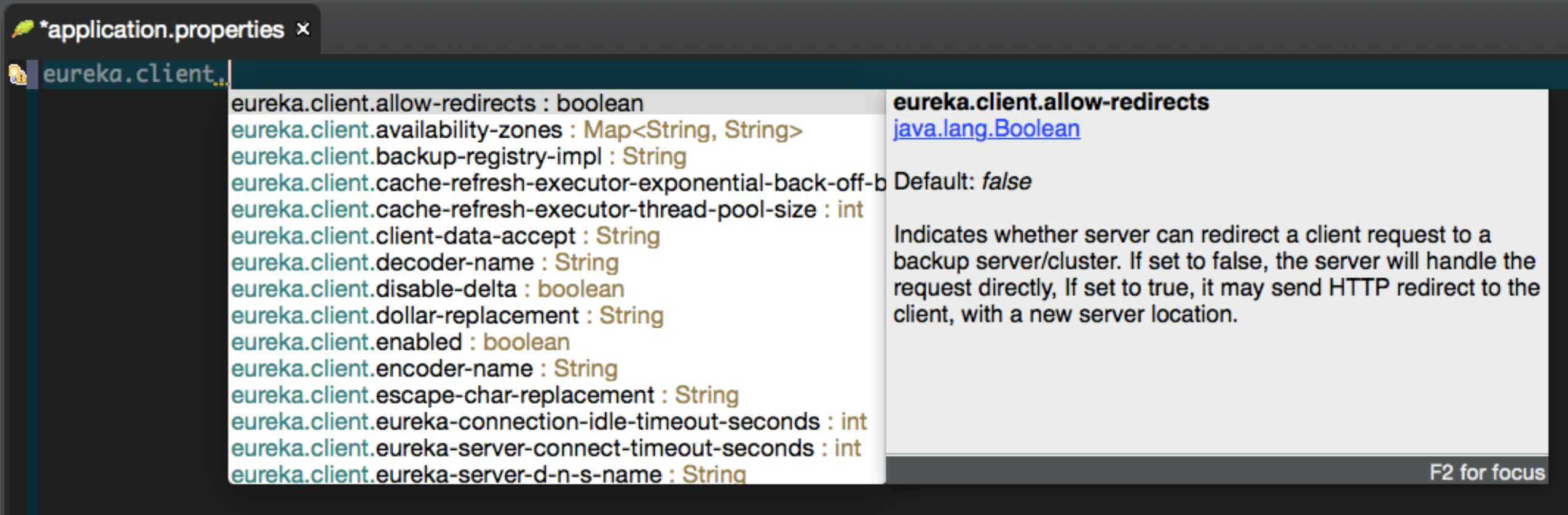
1. Eureka Server Configuration

Eureka Server – the discovery server; contains a registry of services that can be discovered

All configuration under the `eureka.server` prefix

EurekaServerConfigBean





The screenshot shows an IDE window titled '*application.properties' with a sub-tab 'eureka.client...'. The left pane displays a list of configuration properties for the eureka.client, including allow-redirects, availability-zones, backup-registry-impl, cache-refresh-executor-exponential-back-off-b, cache-refresh-executor-thread-pool-size, client-data-accept, decoder-name, disable-delta, dollar-replacement, enabled, encoder-name, escape-char-replacement, eureka-connection-idle-timeout-seconds, eureka-server-connect-timeout-seconds, and eureka-server-d-n-s-name. The right pane shows the details for the selected property, 'eureka.client.allow-redirects', which is of type 'java.lang.Boolean' with a default value of 'false'. A description states: 'Indicates whether server can redirect a client request to a backup server/cluster. If set to false, the server will handle the request directly, If set to true, it may send HTTP redirect to the client, with a new server location.' A footer bar indicates 'F2 for focus'.

```
eureka.client.allow-redirects : boolean
eureka.client.availability-zones : Map<String, String>
eureka.client.backup-registry-impl : String
eureka.client.cache-refresh-executor-exponential-back-off-b
eureka.client.cache-refresh-executor-thread-pool-size : int
eureka.client.client-data-accept : String
eureka.client.decoder-name : String
eureka.client.disable-delta : boolean
eureka.client.dollar-replacement : String
eureka.client.enabled : boolean
eureka.client.encoder-name : String
eureka.client.escape-char-replacement : String
eureka.client.eureka-connection-idle-timeout-seconds : int
eureka.client.eureka-server-connect-timeout-seconds : int
eureka.client.eureka-server-d-n-s-name : String
```

eureka.client.allow-redirects
[java.lang.Boolean](#)
Default: *false*
Indicates whether server can redirect a client request to a backup server/cluster. If set to false, the server will handle the request directly, If set to true, it may send HTTP redirect to the client, with a new server location.

F2 for focus

2. Eureka Client Configuration

Eureka Client - anything that can discover services

All configuration under the `eureka.client` prefix

EurekaClientConfigBean



*application.properties x

eureka.instance..

eureka.instance.a-s-g-name : String

eureka.instance.app-group-name : String

eureka.instance.appname : String

eureka.instance.data-center-info : com.netflix.appinfo.DataC

eureka.instance.default-address-resolution-order : String[]

eureka.instance.environment : org.springframework.core.env

eureka.instance.health-check-url : String

eureka.instance.health-check-url-path : String

eureka.instance.home-page-url : String

eureka.instance.home-page-url-path : String

eureka.instance.host-info : org.springframework.cloud.com

eureka.instance.hostname : String

eureka.instance.inet-utils : org.springframework.cloud.com

eureka.instance.initial-status : com.netflix.appinfo.InstanceIn

eureka.instance.instance-enabled-onit : boolean

eureka.instance.health-check-url-path

java.lang.String

Default: /health

Gets the relative health check URL path for this instance. The health check page URL is then constructed out of the hostname and the type of communication - secure or unsecure as specified in securePort and nonSecurePort. It is normally used for making educated decisions based on the health of the instance - for example, it can be used to determine whether to proceed deployments to an entire farm or stop the deployments without causing further damage.

F2 for focus

3. Eureka Instance Configuration

Eureka Instance - anything that registers itself with the Eureka Server to be discovered by others

All configuration under the `eureka.instance` prefix

EurekaInstanceConfigBean



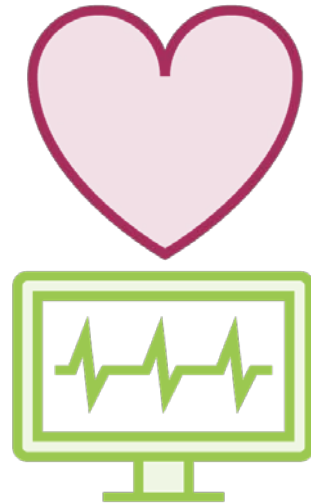
Spring Cloud Eureka: Health & High Availability



Eureka Server: Are My Services Healthy?



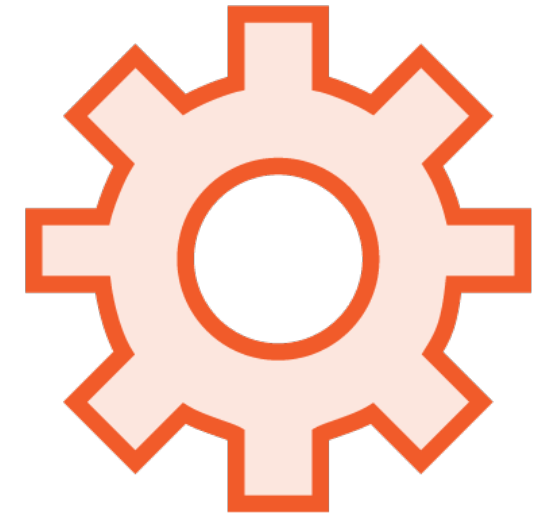
Regularly
checks the
status of
services



Clients send
heartbeats
every 30 sec
(default)



Services
removed after
90 secs of no
heartbeats
(default)



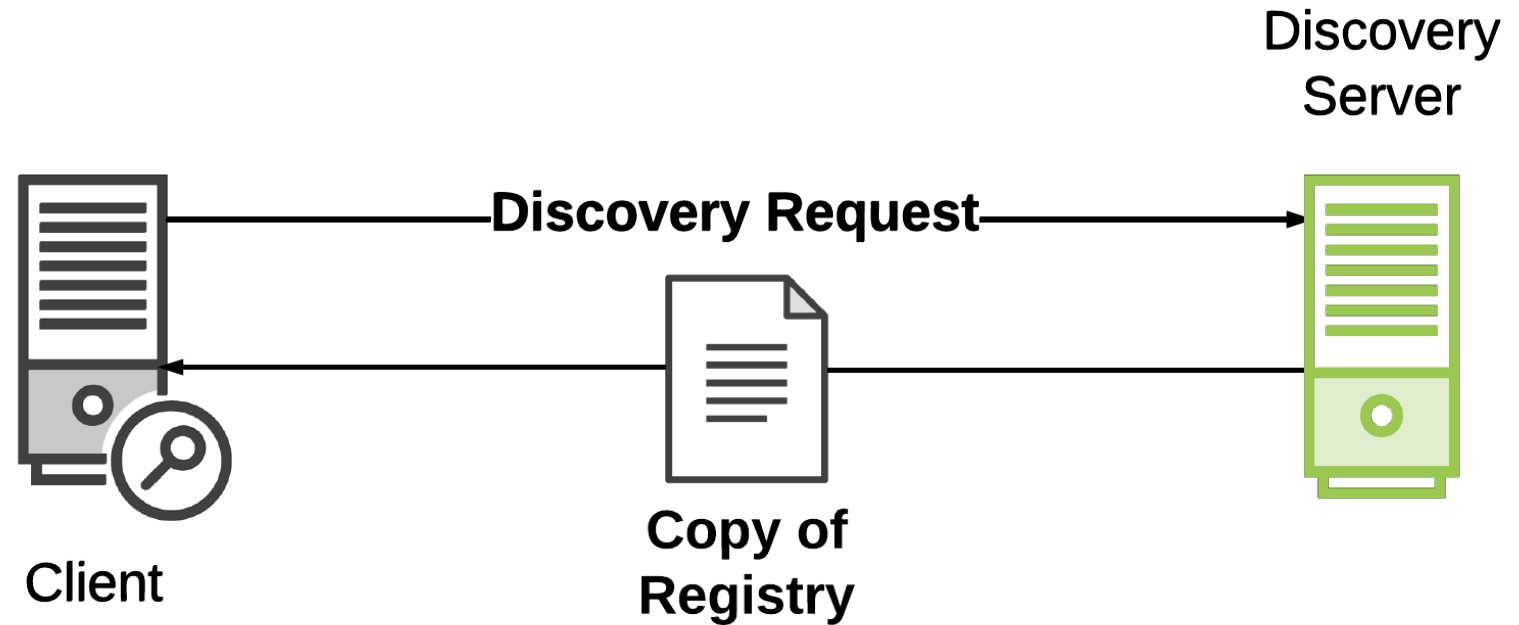
Can customize
configuration to use
/health endpoint
`eureka.client.
healthcheck.enabled`



The registry is distributed
(cached locally on every
client)

Clients *can* operate
without discovery server

Fetches deltas to update
registry



Spring Cloud Eureka AWS Support



Spring Cloud Eureka is AWS Aware

AWS-specific instance data

Multi-zone aware

Multi-region aware

Elastic IP Binding



Configuring Spring Cloud Eureka for AWS

```
@Configuration
public class AppConfig {

    @Bean
    public EurekaInstanceConfigBean eurekaInstanceConfig(
        InetUtilsProperties properties)
    {
        EurekaInstanceConfigBean bean = new
            EurekaInstanceConfigBean(new InetUtils(properties));
        AmazonInfo info = AmazonInfo.Builder.newBuilder().autoBuild("eureka");
        bean.setDataCenterInfo(info);
        return bean;
    }
}
```



Filter by tags and attributes or search by keyword

1 to 2 of 2

<input type="checkbox"/>	Name ▾	Instance ID ▴	Instance Type ▾	Availability Zone ▾	Instance State ▾	Status Checks ▾	Alarm Status
<input checked="" type="checkbox"/>	spring-cloud-discovery-server-1	i-782b7576	t2.micro	us-east-1b	● running	✓ 2/2 checks ...	None
<input type="checkbox"/>	spring-cloud-discovery-server-2	i-b9bcba41	t2.micro	us-east-1e	● running	✓ 2/2 checks ...	None

eureka.client.availability-zones.us-east-1=us-east-1b,us-east-1e

Availability Zones Configuration in `application.properties`

`eureka.client.availability-zones.[region]=[az1],[az2],[az3]`



Allocate New Address

Actions

Refresh

Settings

Help

Filter

VPC addresses

Search Elastic IPs...

<< 1 to 2 of 2 Elastic IPs >>

<input type="checkbox"/>	Address	Allocation ID	Instance ID	Network Interface ID	Scope	Private Address
<input type="checkbox"/>	34.192.167.121	eipalloc-bdf9e782	i-782b7576	eni-2d3231d2	vpc	172.31.52.243
<input type="checkbox"/>	34.193.24.166	eipalloc-59e3fd66	i-b9bcba41	eni-f1e0c418	vpc	172.31.33.117

eureka.client.service-url.us-east-1b=
http://ec2-34-192-167-121.compute-1.amazonaws.com:8761/eureka/

eureka.client.service-url.us-east-1e=
http://ec2-34-193-24-166.compute-1.amazonaws.com:8761/eureka/

Service URL Configuration in `application.properties`

`eureka.client.service-url.[zone]=http://[eip-dns]/eureka`

- * Use EIP DNS name. Do not use IP (as of version Eureka 1.4)



Eureka Dashboard: AWS Multi-zone Discovery Servers

DS Replicas

ec2-34-192-167-121.compute-1.amazonaws.com

ec2-34-193-24-166.compute-1.amazonaws.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
DISCOVERY-SERVER-1	ami-40d28157 (1)	us-east-1b (1)	UP (1) - i-782b7576
DISCOVERY-SERVER-2	ami-40d28157 (1)	us-east-1e (1)	UP (1) - i-b9bcba41



Eureka Dashboard: AWS Instance Data

Instance Info

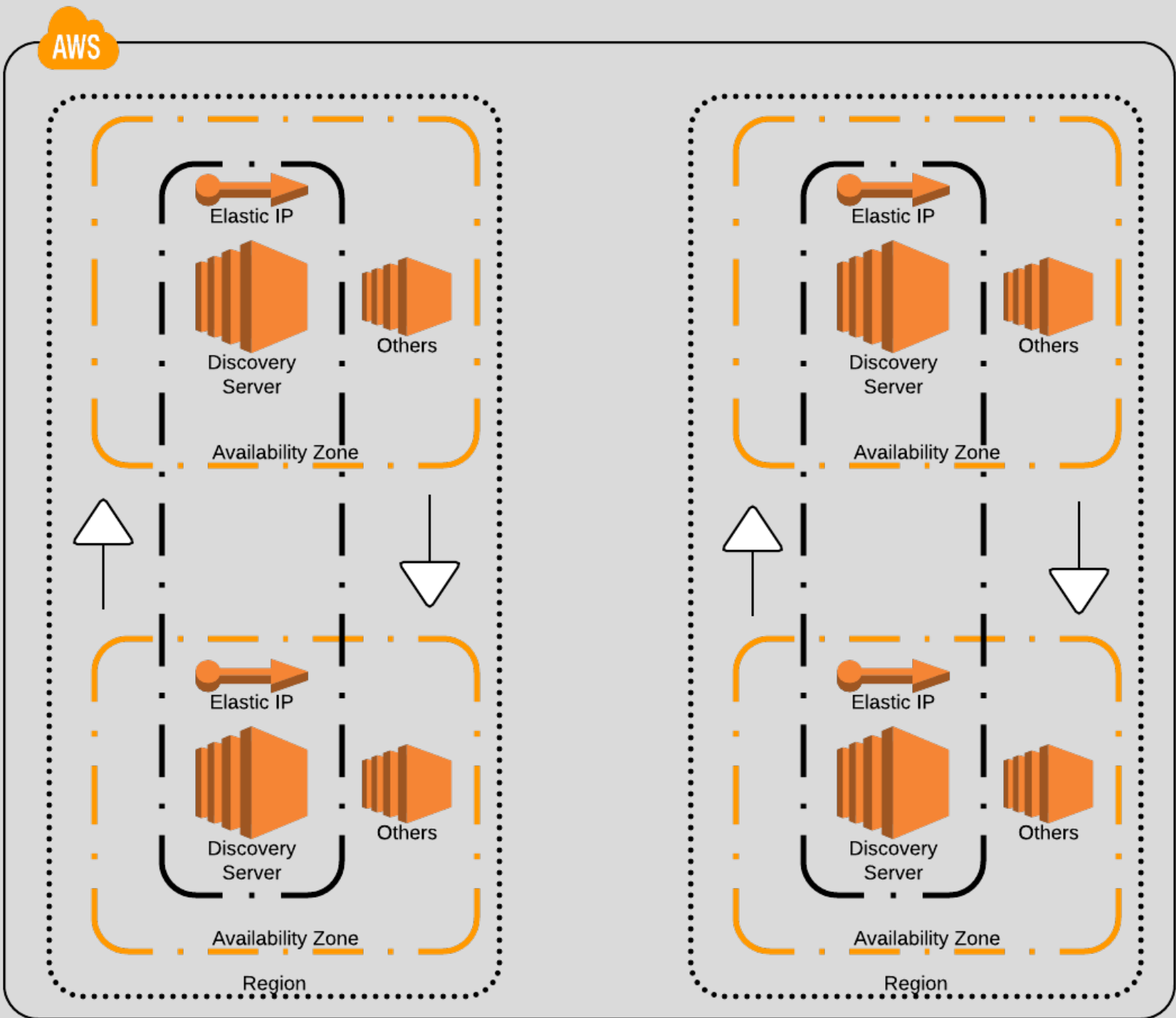
Name	Value
public-ipv4	34.192.167.121
public-hostname	ec2-34-192-167-121.compute-1.amazonaws.com
instance-id	i-782b7576
instance-type	t2.micro
ami-id	ami-40d28157
ipAddr	172.31.52.243
status	UP
availability-zone	us-east-1b



The diagram illustrates the concept of Availability Zones in AWS. It shows two identical regions, each containing two availability zones. Each availability zone contains a discovery server and other resources, all connected to a common Elastic IP. The diagram demonstrates how resources are distributed across multiple availability zones for high availability.

The diagram illustrates the concept of Availability Zones in AWS. It shows two identical regions, each containing two availability zones. Each availability zone contains a discovery server and other resources, all connected to a common Elastic IP. The diagram demonstrates how resources are distributed across multiple availability zones for high availability.

The diagram illustrates the concept of Availability Zones in AWS. It shows two identical regions, each containing two availability zones. Each availability zone contains a discovery server and other resources, all connected to a common Elastic IP. The diagram demonstrates how resources are distributed across multiple availability zones for high availability.



Summary



What is service discovery and why is it important?

Using the Spring Cloud Eureka Client & Server

Spring Cloud Eureka Dashboard

Health & High Availability

Configuring Spring Cloud Eureka

AWS Support

