Spring Cloud Discovery

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

Overview

Locating Services at Runtime Using Service Discovery

- Service Discovery
- Role of service discovery in micro services
- Problem with the status quo

Content

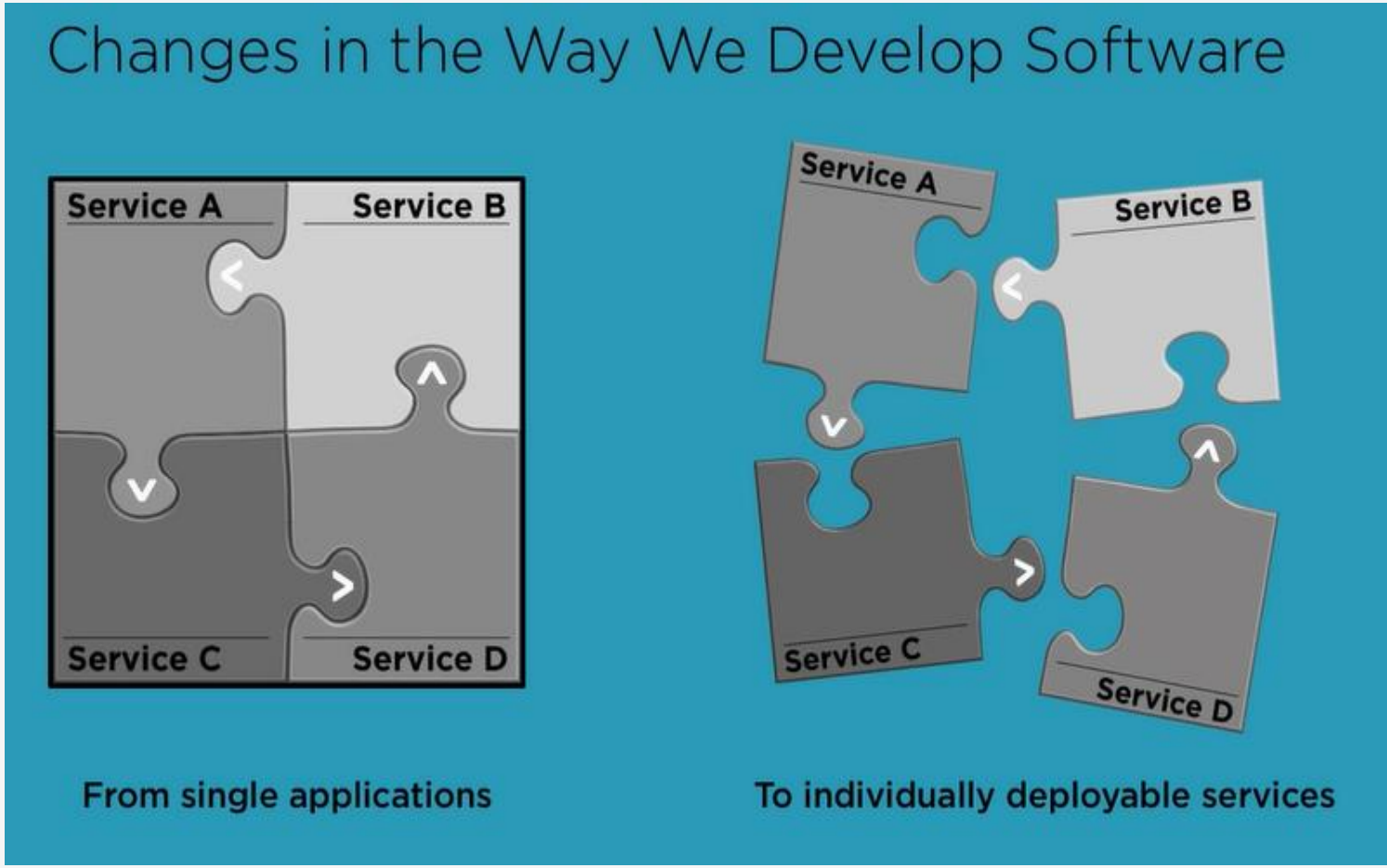
Service Discovery
by Pratap Kumar

- Describing Spring Cloud Eureka
- Creating a Eureka Server
- Registering services with Eureka (Eureka Client)
- Discovering Services with Eureka (Eureka Client)
- Configuring health information
- Reviewing the high availability setup
- Options for advanced configuration
- Eureka Dashboard
- Summary

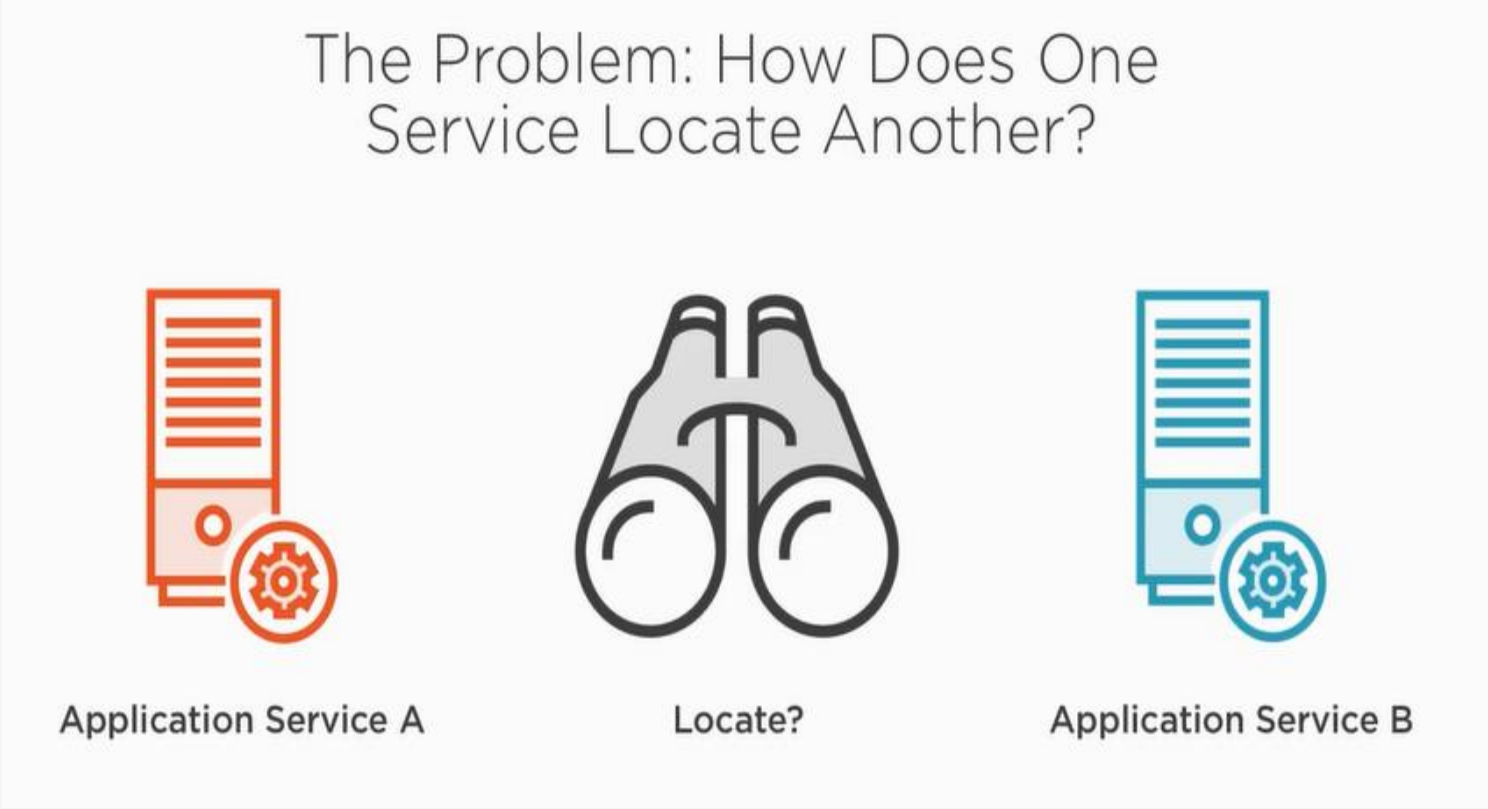
Service Discovery?

Service Discovery
by Pratap Kumar

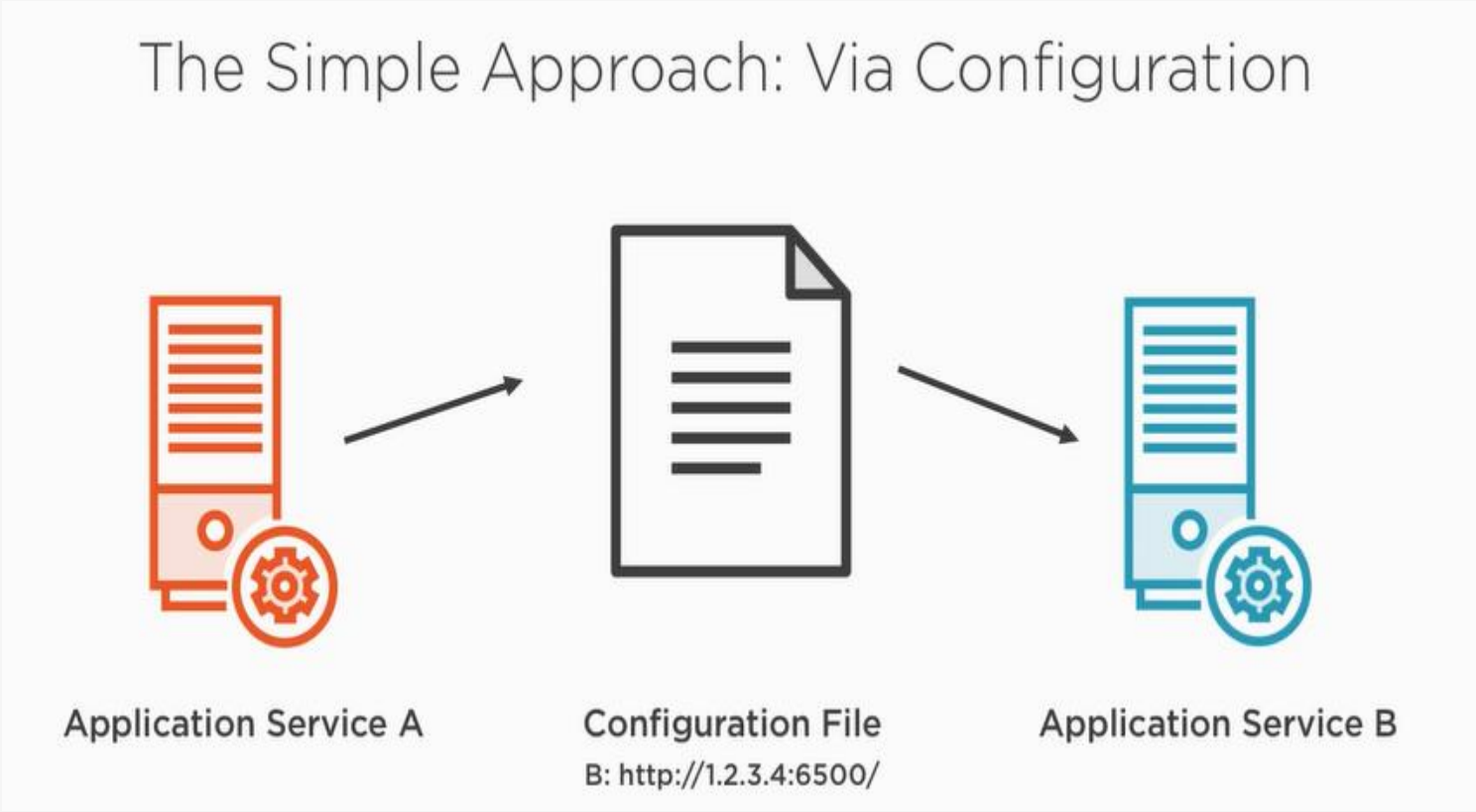
- **What is Service Discovery and why do we need it ?**



Service Discovery?

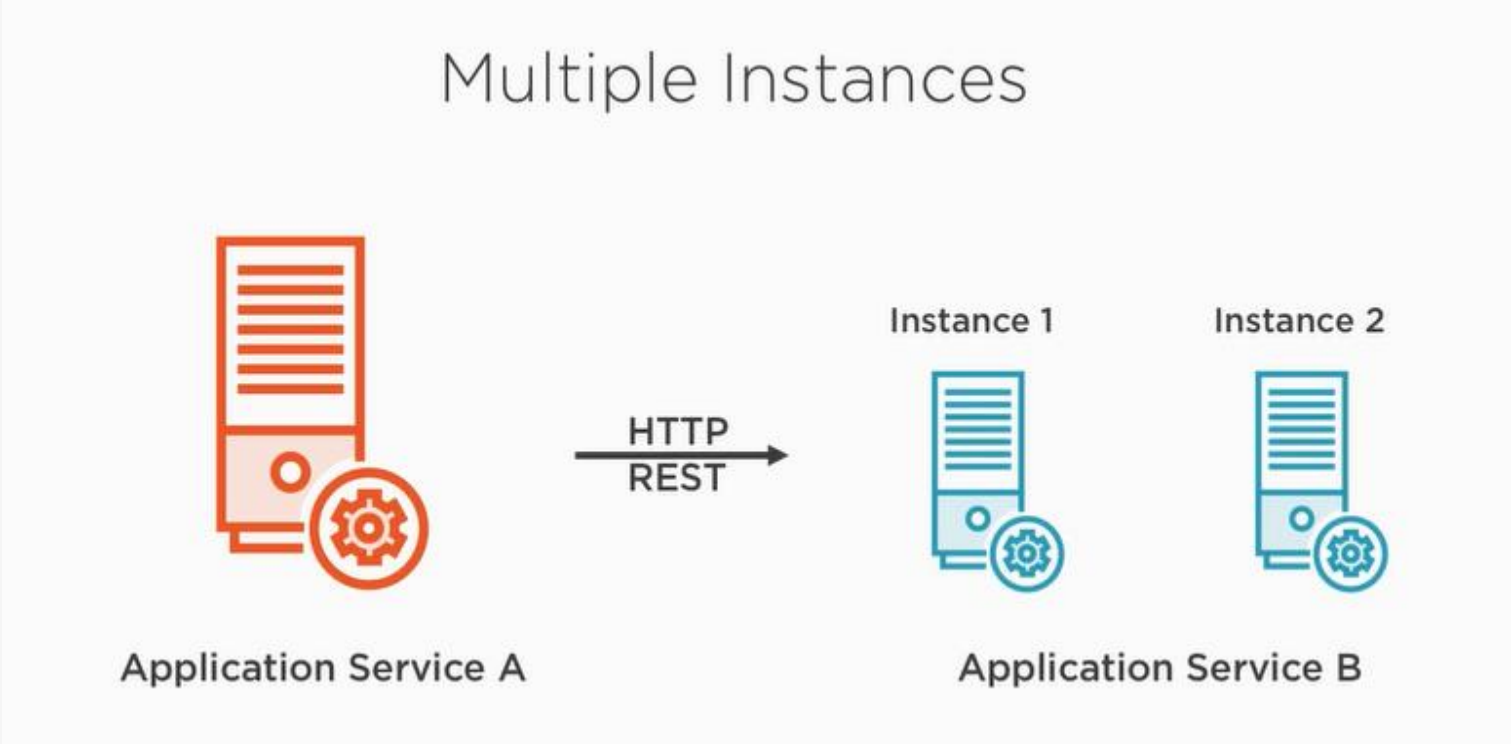


Service Discovery?



Service Discovery
by Pratap Kumar

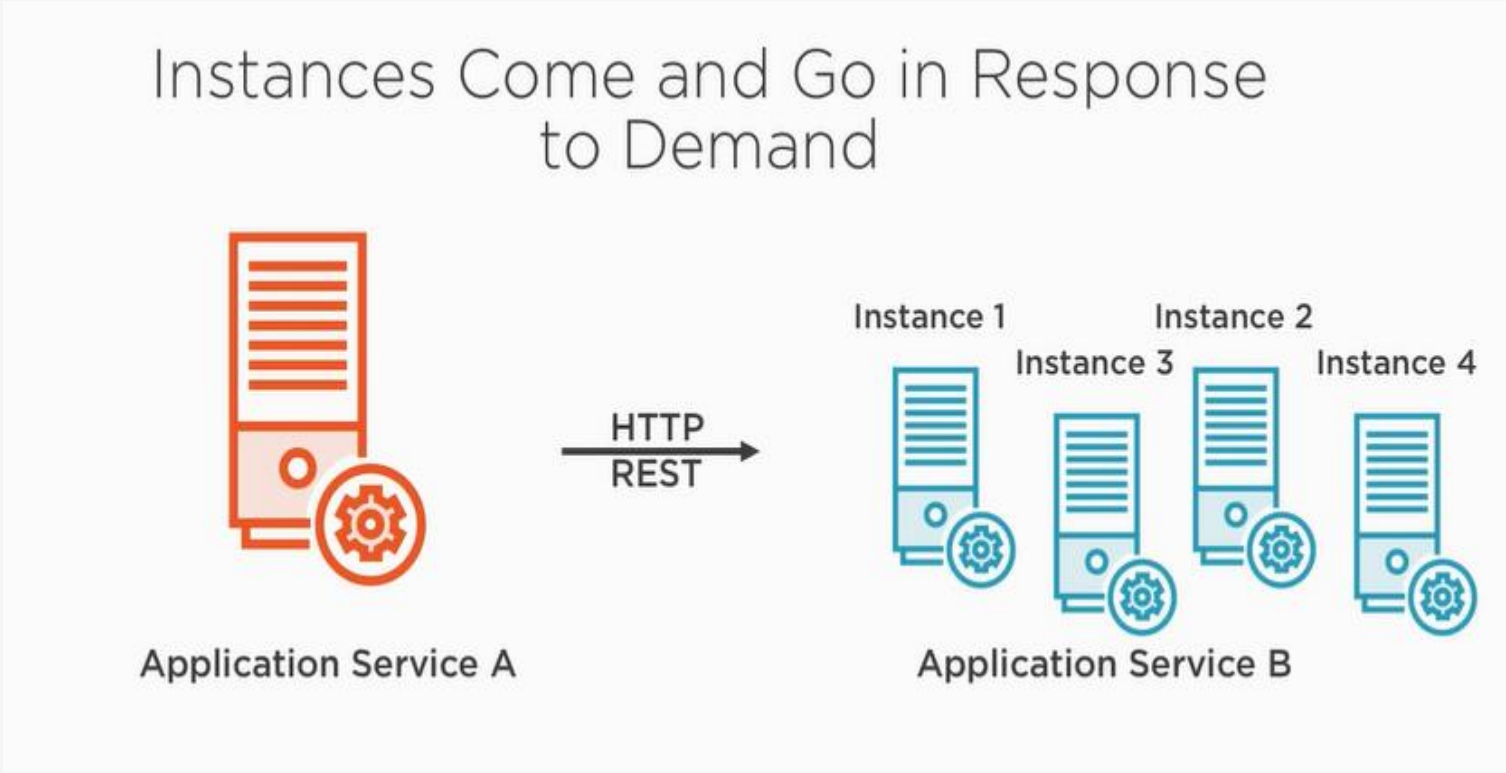
Service Discovery?



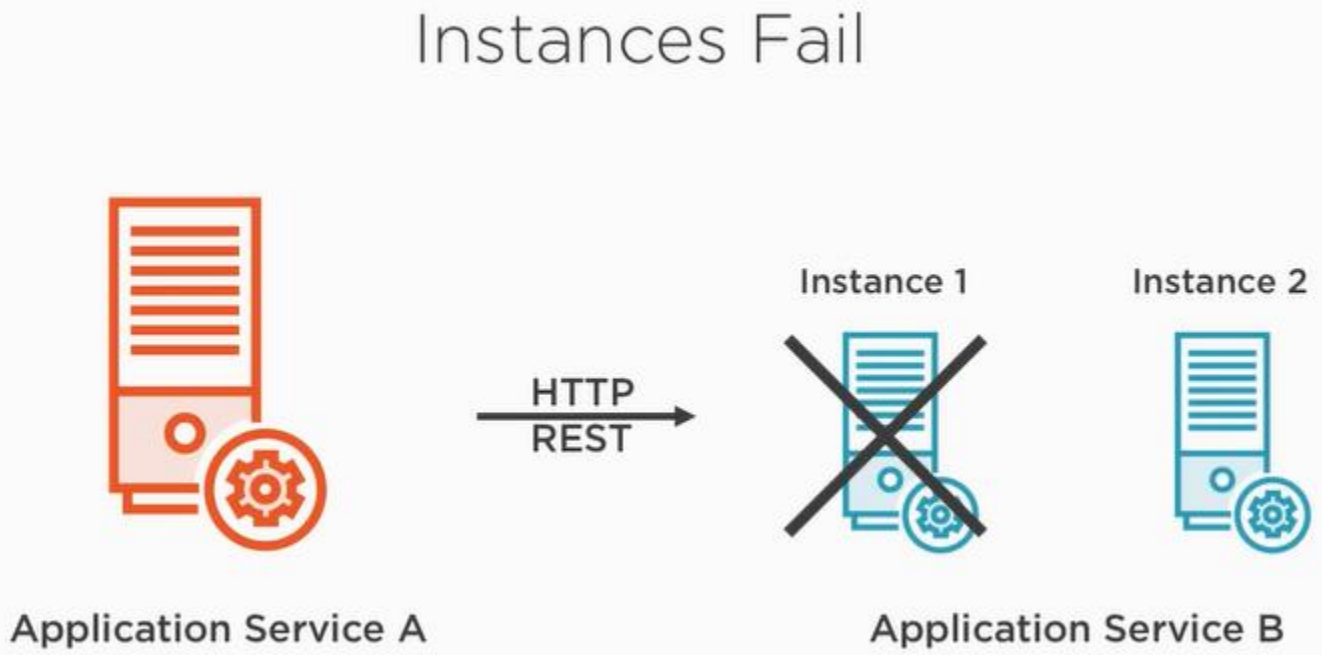
Service Discovery?

Service Discovery
by Pratap Kumar

Cloud?



Service Discovery?



The Simple Approach is far too static and (frozen in time) for the cloud!

Solution : Service Discovery server

What is a Service Discovery?

- Service Discovery name itself is explaining about behavior, so in case of MicroServices there are many services running on the internet, so we need one centralized places from where we can directly find out which service is running on which IP and which PORT.

- Service Discovery is how applications and MicroServices locate each other on a network

- service Discovery is a single lookup service and self maintaining, you don't need to add clients because clients register themselves.

Why do we need service Discovery?

- There are a large number of MicroServices and all services are inter-related and they are communicating with each other and this very challenging to configure so with the help of service discovery it is automated

- You are writing some code that invokes a service that has a REST API so in order to make a request, your code need to know the network location like IP address and port of a service instance. in a traditional application running on a physical hardware, the network locations of the service instances are relatively static means always same.

- In Modern, cloud based MicroServices application, however this is much more difficult problem to solve
- because service instance have dynamically assigned network locations the set of service instances changes dynamically because of auto-scaling, failures and upgrades

Service Discovery

- **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

Role of Service Discovery

- **Recognize the Dynamic Environment**
  - Services has to Advertise their existence and disappearances in a microservice architecture.
- **Have a Live view of healthy service**
  - Instance failure are detected and becomes invalid discovery result in a good system
- **Avoid hard coded references to service location**
  - Service has no prior knowledge about the physical location of services in this short of architecture.
- **Centralized list of available service**

Problem with status Quo

- Outdated configuration management DBs
- Simplistic HTTP 200 health checks
- Limited load balancing for middle-tier
- DNS is insufficient for micro services
- Registries can be single points of failure

Spring Cloud & Service Discovery

Service Discovery by Pratap Kumar

Spring Cloud & Service Discovery

Service Discovery by Pratap Kumar

- **Discover service with**
  - spring cloud consul
  - spring cloud zookeeper
  - spring cloud Netflix

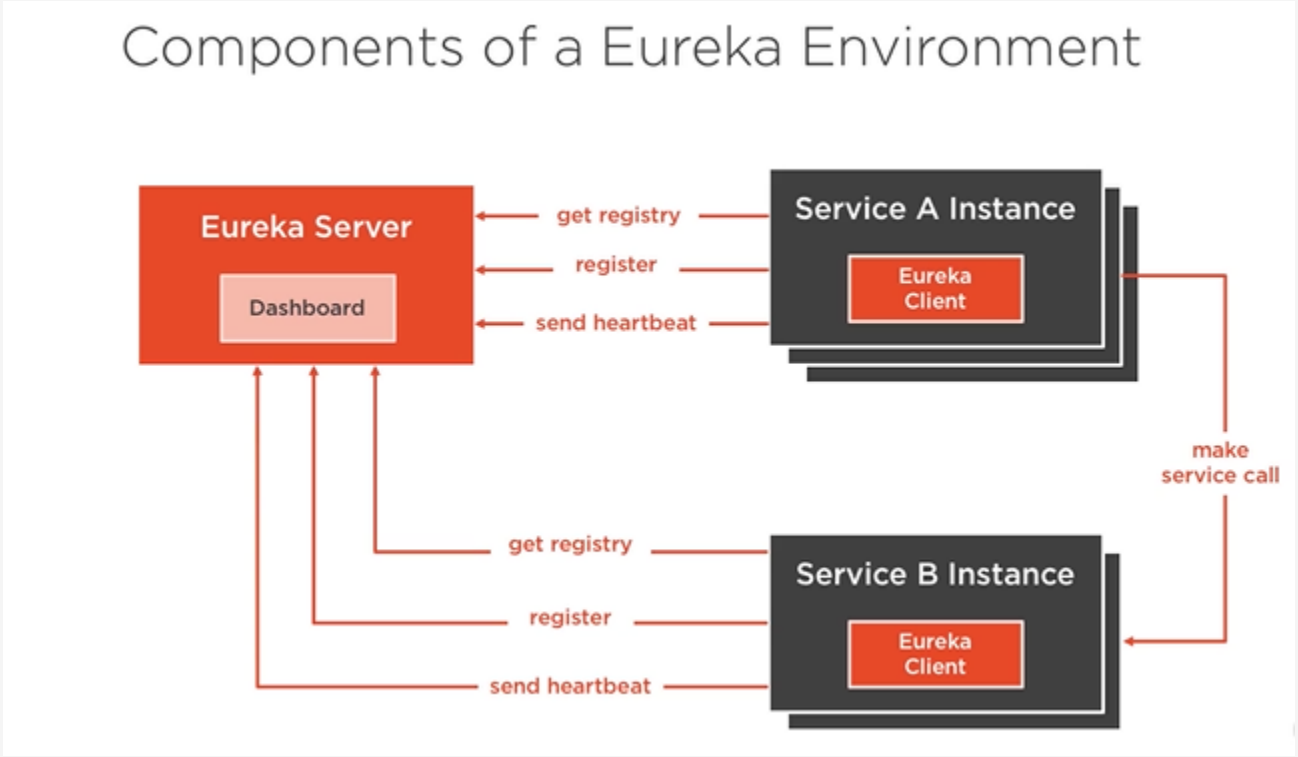
- Netflix OSS + Spring + Spring Boot = Spring Cloud Netflix

- Spring Cloud Netflix
  - Spring Cloud Netflix Eureka server
  - Spring Cloud Netflix Eureka Client
  - Other Spring Cloud Netflix Projects

The History of Eureka

- First Released by Netflix team in 2012
- Used for Middle Tier load balancing
- Integrated into many other Netflix projects

Eureka



Service Discovery by Pratap Kumar

Service Discovery Key Components

Creating Eureka Server

- Add spring-cloud-starter-eureka-server Dependencies
- Standalone or clustered Configuration
- @EnableEurekaServer annotation
- Numerous Configuration option

Eureka Dashboard

- Enabled by default
- Shows Environment info
- Lists registered services and instances
- View service health

Creating Eureka Server

- Create a Spring Boot Project with Following Dependencies
  - **Actuator , Devtools , Eureka Server**

```
@SpringBootApplication
@EnableEurekaServer
public class MainApplication{
        public static void main(String []  args){
        }
}
```

Eureka Server Configuration

- In the application.properties
  - Spring.application.name=discovery-server
  - eureka.client.register-with-eureka=false
  - eureka.client.fetch-registry=false
  - server.port=8761

  - eureka.datacenter=hyderabad
  - eureka.environment=production

Registering a Service with Eureka

Registering a Service with Eureka

- Eureka in class path leads to registration
- Service name , host info sent during bootstrap
- @EnableDiscoveryClient and @EnableEurekaClient
  - Register that service itself
  - Makes it a client
- Sends heartbeat every 30 seconds
- Heartbeat can include health status
- HTTP or HTTPS supported

Demo

- Create a spring boot starter project (hello-service)
- Add project dependency on eureka
  - **Actuator , Devtools , Eureka Discovery , web**
- Annotate the primary class
- Add bootstrap and application properties
- Start up microservice and see the registry
- Start a second instance and see in registry

Demo

```
@SpringBootApplication
@EnableDiscoveryClient
//  @EnableEurekaClient
public class MainApplication{
        public static void main(String []  args){
        }
}
```

Demo

Service Discovery
by Pratap Kumar

```
@RestController
public class HelloService {
        @Value("${service.instance.name}")
        private String instance;

        @RequestMapping("/")
        public String helloSerive() {
                return "Hello from, "+instance;
        }
}
```

Demo : 1st approach

Service Discovery
by Pratap Kumar

- **In the application.properties**
  - spring.application.name= service
  - eureka.client.service-url.defaultZone=http://localhost:8761/eureka

Demo : 1st approach

Service Discovery
by Pratap Kumar

- **Run Configuration:**
  - Name: instance 1
  - main type: configure main class name
  - override properties:
    - server.port:8081
    - service.instance.name:instance1

- **Run Configuration: Duplicate current Configuration**
  - Name: instance 2
  - main type: configure main class name
  - override properties:
    - server.port:8082
    - service.instance.name:instance2

Discovering a Service with Eureka

Discovering with Eureka

Service Discovery by Pratap Kumar

- @EnableDiscoveryClient and @EnableEurekaClient
- Client works with Locale cache
- Cache refreshed , reconciled regularly
- Manually Load balance or User Ribbon
- Can prefer talking to registry in closest Zone
- May take Multiple Heartbeat to discover new services

Demo

Service Discovery
by Pratap Kumar

- Create spring-boot-starter application
- Add Dependency on Eureka
  - **Actuator , Devtools , Eureka Discovery , web**

- Update application.properties
- Annotate primary class
- Use Spring Cloud Discovery / Eureka Client library **OR**
- Use Load Balanced RestTemplate
- Replace hard-coded URL with registry lookup

Demo

- Create a Spring Boot Project with Following Dependencies

```
@SpringBootApplication
@EnableDiscoveryClient
public class MainApplication{
        public static void main(String []  args){
                Discover Service Here
        }
}
```

Using Spring Cloud Eureka Client in an Application Client

- In the application.properties
  - Spring.application.name= client
  - eureka.client.service-url.defaultZone=http://localhost:8761/eureka
  - eureka.client.register-with-eureka=false

Using Spring Cloud Eureka Client in an Application Client

Service Discovery
by Pratap Kumar

Discovering Services as a Client: Two Options

Eureka server specific

```
@Inject
EurekaClient client
```

Discovery server agnostic

```
@Inject
DiscoveryClient client
```

* Spring DiscoveryClient

EurekaClient

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

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

DiscoveryClient

```java
@RestController
public class HelloController {
    @Autowired
    private EurekaClient client;
    @Autowired
    private RestTemplateBuilder builder;
    @RequestMapping("/")
    public String callService() {
        RestTemplate restTemplate = builder.build();
        InstanceInfo instanceInfo = client.getNextServerFromEureka("service", false);
        String baseUrl = instanceInfo.getHomePageUrl();
        ResponseEntity<String> response = restTemplate.exchange(baseUrl,
                                                HttpMethod.GET, null, String.class);
        return  response.getBody();
    }
}
```

DiscoveryClient

Ser
by Prata

Spring Cloud Eureka Dashboard

Service Discovery by Pratap Kumar

- Enabled by default
  - eureka.dashboard.enabled=true
- Displays useful metadata and service status
- Localhost:8761

Areas of Configuration

- eureka.server.*
- eureka.client.*
- eureka.instance.*

- Eureka Server configuration
  - Eureka Server - the discovery; contains a registry of services that can be discovered
  - All Configuration under eureka.server prefix
- Eureka client configuration
  - Eureka client = anything that can discover services
  - All configuration under the eureka.client prefix
- Eureka instance configuration
  - Eureka instance= anything that registers itself with the eureka server to be discovered by other
  - All configuration under the eureka.instance prefix

Second App

App 2

- Open existing "toll rate service " microservice (dropbox)
- Add project dependency on eureka (done)
- Annotate primary class (done)
- Add bootstrap and application.properties (done)
- Startup microservice and see in registry
- Start a second instance and see in registry
- Do same sequence with "fastpass service" microservice

- **Registering Microservice with Eureka**
  - Pratap-eureka-tollrate-service
  - Pratap-eureka-fastpass-service

Demo :2nd approach

Service Discovery
by Pratap Kumar

- bootstrap.properties
  - spring.applicaiton.name = hello-service

- application.properties
  - # server.port = 8085
  - #eureka.client.service-url.defaultZone=http://localhost:8761/eureka
  - eureka.client.register-with-eureka=true
  - eureka.client.fetch-registry = true
  - eureka.instance.instance-id=${spring.application.name}:${random.int}
  - server.port = 0
  - eureka.instance.hostname=localhost

App 2

- Open “toll rate billboard” application
- Add dependency on eureka
- Update application.properties
- Annotate Primary class
- Add a load balanced RestTemplate
- Replace hard coded URL with Registry lookup
- Test out “toll rate billboard” appliction
- Repeat with “fast pass console” application

- **Discovering Microservice from Eureka**
  - Pratap-eureka-tollrate-billboard
  - Pratap-eureka-fastpass-console

Configuring Service Health Information

Health Config

- Heartbeat doesn't convey health
- Possible to include health information
- Can extends and create Own Health Check

- Return to "toll rate" micro service and add a custom Health check
- Startup microservice and wait for error
- See service taken out of rotation by Eureka

Health Config


```
@Component
public class CustomHealthCheck implements HealthIndicator {
int errorcode = 0;
@Override
public Health health() {
System.out.println("Health check performed, error code is " + errorcode);
if (errorcode > 4 && errorcode < 8) {
errorcode++;
return Health.down().withDetail("Custom error code", errorcode).build();

}
errorcode++;
return Health.up().build();
}
}
```

Application.properties

eureka.client.healthcheck.enabled=true

High Availability Architecture for Eureka

High Availability

- *Build in "self preservation" model*
- *Native support for peer to peer registry replication*
- *Use DNS in front of Eureka Cluster*
- *Recommended to have one Eureka Cluster in each Zone*

Advanced Configuration Options

Service Discovery
by Pratap Kumar

Health Config

- *Dozens and dozens of configuration flags*
- *Set cache refresh intervals*
- *Set timeouts*
- *Set Connection limits*
- *Set service metadataMap*
- *Override default service, health endpoints*
- *Define replication limits, timeout, retries*