Service Discovery with Spring Cloud Eureka

Implementing Passive Service Discovery

What is it, and why would you use it?

Objectives

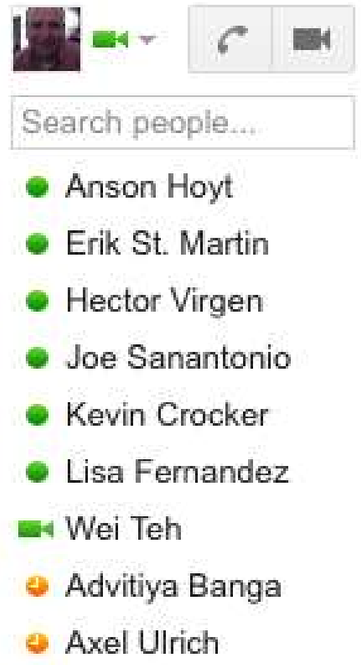
- At the end of this module, you will be able to
  - Explain what Passive Service Discovery is
  - Build and Run and Spring Cloud Eureka Server
  - Build, Run, and Configure a Eureka Client

Module Outline

- **Service Discovery**
- Eureka Server
- Discovery Client
- Service Discovery Considerations

Service Discovery - Analogy

- When you sign into a chat client, what happens?
  - Client 'registers' itself with the server – server knows you are online.
  - The server provides you with a list of all the other known clients



- In essence, you client has “discovered” the other clients
  - ...and has itself been “discovered” by others

Service Discovery

- Microservice architectures result in large numbers of inter-service calls
  - Very challenging to configure
- How can one application easily find all of the other runtime dependencies?
  - Manual configuration – Impractical, brittle
- Service Discovery provides a single 'lookup' service.
  - Clients register themselves, discover other registrants.
  - Solutions: Eureka, Consul, Etcd, Zookeeper, SmartStack, etc.
    - (Developer Preview versions of Zookeeper & Consul available as of this writing)

Eureka –
Service Discovery Server and Client

- Part of Spring Cloud Netflix
  - Battle tested by Netflix
- Eureka provides a 'lookup' server.
  - Generally made highly available by running multiple copies
  - Copies replicate state of registered services.
- “Client” Services register with Eureka
  - Provide metadata on host, port, health indicator URL, etc.
- Client Services send heartbeats to Eureka
  - Eureka removes services without heartbeats.

Module Outline

- Service Discovery
- **Eureka Server**
- Discovery Client
- Service Discovery Considerations

Making a Eureka Server – Building, part 1

- Include minimal dependencies in your POM (or Gradle)
  - Spring **Cloud** Starter Parent
  - Spring Cloud Starter Eureka Server

```
<parent>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-parent</artifactId>
  <version>Angel.SR4</version>
</parent>

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

Does this look familiar?

- Same parent as config server
- “cloud-starter-eureka-server” instead of “cloud-config-server”

Making a Eureka Server – Building, part 2

- Switch Eureka on with @EnableEurekaServer:

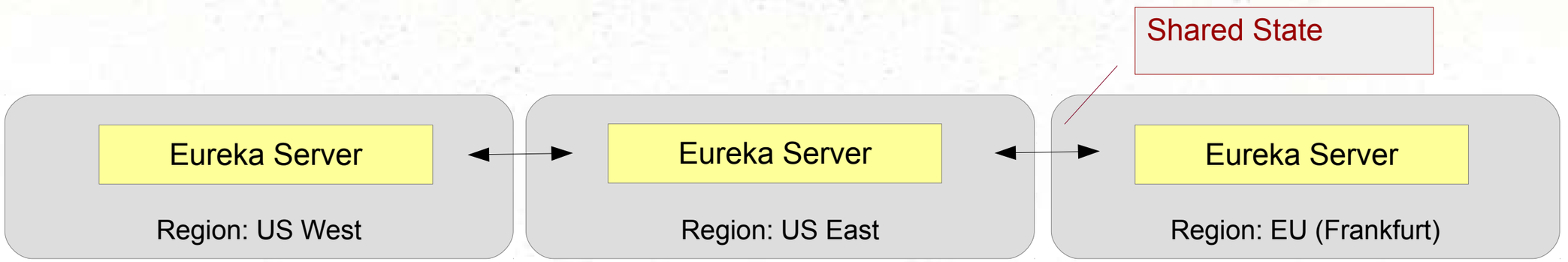
```
@SpringBootApplication
@EnableEurekaServer
public class Application {

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

- That's It!

Multiple Servers

- Typically, multiple Eureka servers should be run simultaneously
  - Otherwise, you'll get many warnings in the log
  - Eureka servers communicate with each other to share state
  - Provides High Availability
- Each server should know URL to the others
  - Can be provided by Config Server
  - One server (JAR), multiple profiles



Module Outline

- Service Discovery
- Eureka Server
- **Discovery Client**
- Service Discovery Considerations

Registering with Eureka – Part 1

- Use the Spring **Cloud** Starter parent as a Parent POM:

```
<parent>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-parent</artifactId>
   <version>Angel.SR4</version>
</parent>
```

- ...OR use a Dependency management section:

```
<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-starter-parent</artifactId>
         <version>Angel.SR4</version>
         <type>pom</type>
         <scope>import</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```

Look familiar?
exactly the same options
as a spring cloud config client.

Registering with Eureka – Part 1

- Add Dependency:

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

- ...enable:

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {

}
```

- ...and specify the location of the Eureka server:

```
# application.properties
eureka.client.serviceUrl.defaultZone:  http://server:8761/eureka/
```

Default fallback.
Any Eureka instance will do
(we usually want several comma-separated URLs)

@EnableDiscoveryClient

- Automatically registers client with Eureka server
  - Registers the application name, host, and port
    - Using values from the Spring `Environment`.
    - But can be overridden.
    - Give your application a spring.application.name
- Makes this app an “instance” and a “client”
  - It can locate other services

Service ID (Eureka VIP)
Corresponds to
spring.application.name

```
@Autowired DiscoveryClient client;
public URI storeServiceUrl() {
    List<ServiceInstance> list = client.getInstances("STORES");
    if (list != null && list.size() > 0 ) {
      return list.get(0).getUri();
    }
    return null;
}
```

Module Outline

- Service Discovery
- Eureka Server
- Discovery Client
- **Service Discovery Considerations**

What is a Zone?

- Eureka server designed for multi-instance use
  - Single instance will actually warn you when it runs without any peers!
- Eureka Server does not persist service registrations
  - Relies on client registrations; always up to date, always in memory
    - Stateful application.
- Typical production usage – many Eureka server instances running in different availability zones / regions
  - Connected to each other as “peers”

Which Comes First?
Eureka or Config Server?

- **Config First Bootstrap** (default) – Use Config Server to configure location of Eureka server
  - Implies `spring.cloud.config.uri` configured in each app

- **Eureka First Bootstrap** – Use Eureka to expose location to config server
  - Config server is just another client
  - Implies `spring.cloud.config.discovery.enabled=true` and `eureka.client.serviceUrl.defaultZone` configured in each app.
  - Client makes two network trips to obtain configuration.

References

- http://techblog.netflix.com/2012/09/eureka.html
- https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka

Summary

- Passive Service Discovery -
  - Having services register themselves and find others automatically
- Spring Cloud Eureka Server
  - Convenient wrapper around Netflix Eureka libraries
  - Holds registrations, shares information on other registrants
  - Synchronizes itself with other servers
- Spring Cloud Eureka Client
  - Connects to server to register, and obtain information on other clients.

Exercise

Create and Run Several Applications coordinated by Spring Cloud Eureka

Instructions: Student Files, Lab 4