# Spring cloud consul

## 依赖

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-consul-all</artifactId>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-consul-dependencies</artifactId>

<version>1.2.1.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

## Health 地址自定义

地址及端口更改

spring:

cloud:

consul:

host: localhost

port: 8500

warn:需要放在bootstrap.xml 文件中

应用更改，健康检查地址需修改

spring:

cloud:

consul:

discovery:

healthCheckPath: ${management.contextPath}/health

healthCheckInterval: 15s

放在application.yml

注意：由于目前客户端Eureka 和Consul都是服务注册中心。目前不能相互兼容。所以consul客户端都禁用Eureka 注册发现功能。（不久将来会有所兼容）

## 使Consul实例ID唯一

在大多数情况下，这将允许一个服务的多个实例在一台机器上运行。如果需要进一步的唯一性，使用Spring Cloud，您可以通过在spring.cloud.consul.discovery.instanceId中提供唯一的标识来覆盖

*application.yml*

spring:

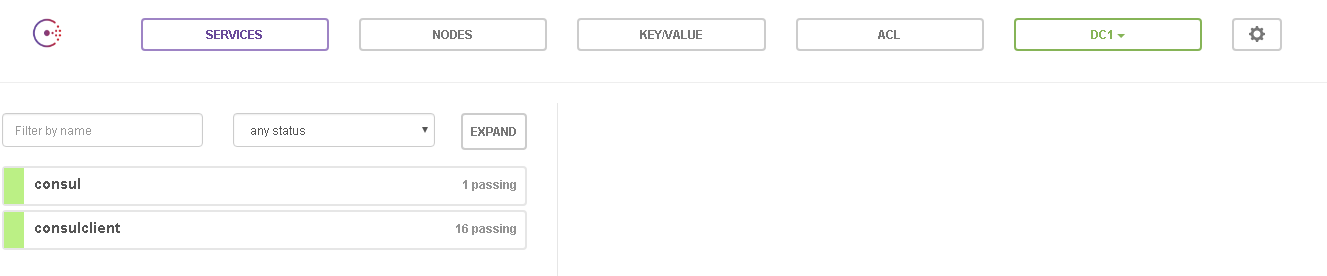
cloud:

consul:

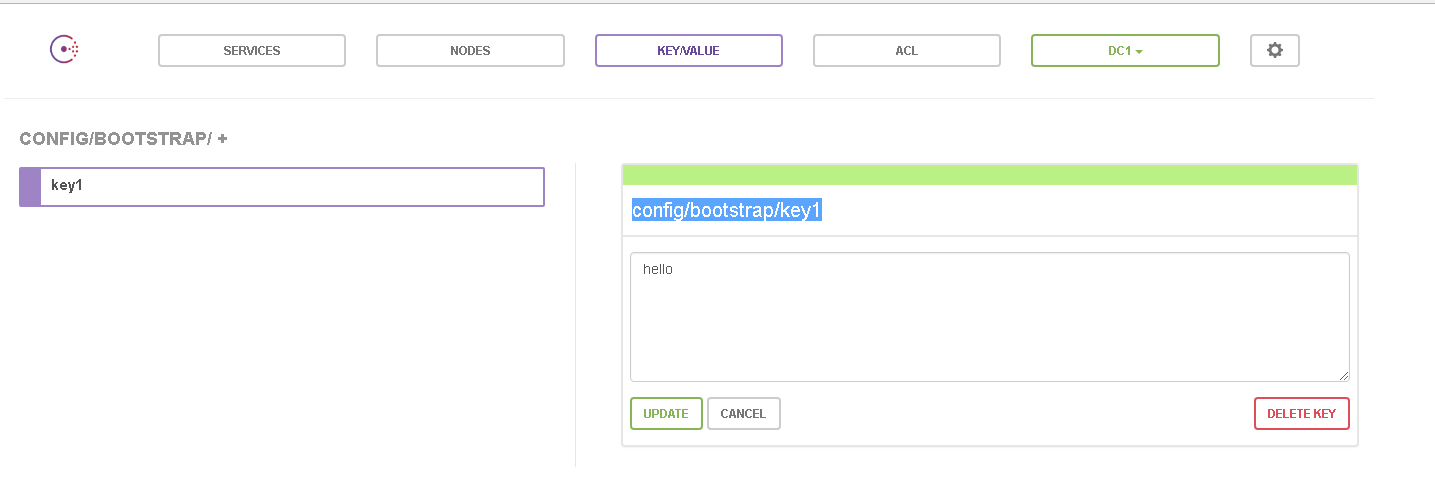
discovery:

instanceId: ${spring.application.name}:${spring.application.instance\_id:${random.value}}

## Consul web ui

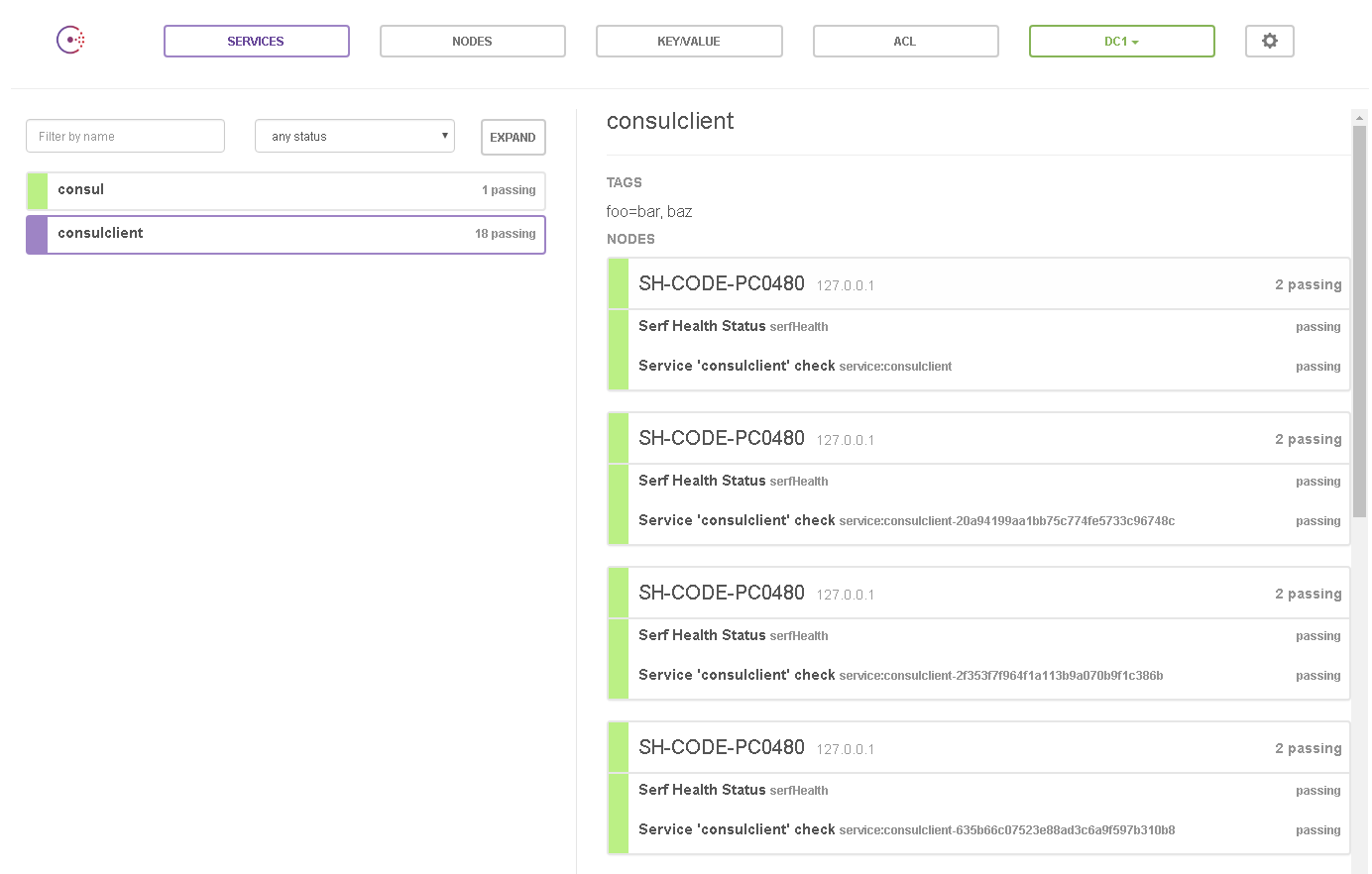


## Key-value configuraton



可以通过web ui 设置更新key value 实现服务配置。

## 服务检测



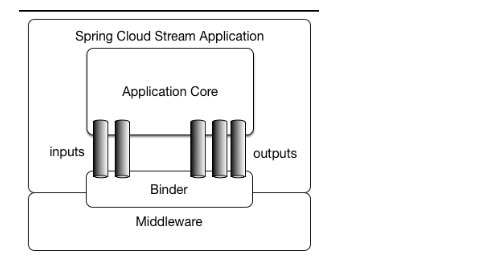
Web UI 提供服务检测health 功能。

# Spring Cloud Stream

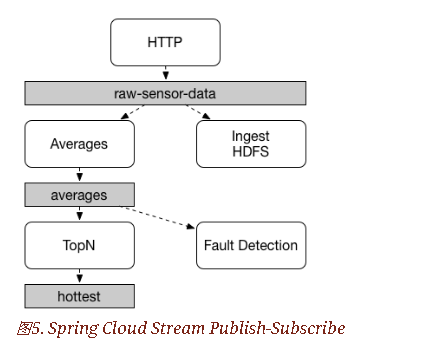
## 依赖

<**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-stream-dependencies</**artifactId**>  
 <**version**>1.0.0.RELEASE</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**dependencyManagement**>  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.1.35</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-bus-amqp</**artifactId**>  
 <**version**>LATEST</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-stream-rabbit</**artifactId**>  
 <**version**>LATEST</**version**>  
 </**dependency**>  
</**dependencies**>

## 应用模型



**发布 - 订阅**



@EnableBinding注释添加到应用程序中，以便立即连接到消息代理，并且可以将@StreamListener添加到方法中，以使其接收流处理事件。以下是接收外部消息的简单接收器应用程序。

## 编程模型

根据channel先定义接口，stream包含默认3个接口（sink，source，processor）, 对于每个绑定接口，Spring Cloud Stream将生成一个实现该接口的bean。调用其中一个bean的@Input注释或@Output注释方法将返回相关的绑定通道。

### 接收端代码：

**public interface** MyMQInterface {  
 String INPUT=**"channel2"**;  
 String ***OUTPUT***=**"channel1"**;  
  
 @Input(MyMQInterface.INPUT)  
 SubscribableChannel inputChannel();  
 @Input(MyMQInterface.***OUTPUT***)  
 MessageChannel outputChannel();  
}

@EnableBinding(MyMQInterface.**class**)  
**public class** MyMQReceiver {  
 @Autowired  
 **private** MyMQInterface **myMQInterface**;  
  
 @StreamListener(MyMQInterface.***INPUT***)  
 **public void** process(String message) {  
 System.***out***.print(**"do service about message here: "**+ (StringUtils.*isEmpty*(message)?**""**:message));  
}  
}

@SpringBootApplication  
**public class** StreamApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(StreamApplication.**class**, args);  
 }  
}

### 发送端代码：

**public interface** OutputSource {  
 String ***OUTPUT*** = **"channel2"**;  
 @Output(***OUTPUT***)  
 MessageChannel output();  
 String ***INPUT*** = **"channel1"**;  
 @Output(***INPUT***)  
 SubscribableChannel input();  
}

@RestController  
**public class** SayHello {  
 **private** OutputSource **source**;  
  
 @Autowired  
 **public** SayHello(OutputSource source) {  
 **this**.**source** = source;  
 }  
 @RequestMapping(method = RequestMethod.***GET***, value = **"sayHello/{message}"**)  
 **public void** sayHello(@PathVariable(**"message"**)String message) {  
 JSONObject returnObj=**new** JSONObject();  
 returnObj.put(**"message"**,message);  
 **source**.output().send(MessageBuilder.*withPayload*(returnObj.toJSONString()).setHeader(**"Content-Type"**,**"application/json;charset=UTF-8"**).build());  
 }  
}

@EnableBinding(OutputSource.**class**)  
**public class** Sender{  
@StreamListener(OutputSource.***INPUT***)  
 **public void** process(String message) {  
 System.***out***.print(**"do service about message here: "**+ (StringUtils.*isEmpty*(message)?**""**:message));  
 }  
}

@SpringBootApplication  
**public class** StreamApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(StreamApplication.**class**, args);  
 }  
}

将发送端的output channel设置为接受端的input channel ，即可完成消息通信。

## 消息聚合

@SpringBootApplication  
@EnableBinding(Source.**class**)  
**public class** SourceApplication {  
 @Bean  
 @InboundChannelAdapter(value = Source.***OUTPUT***)  
 **public synchronized** MessageSource<String> timerMessageSource() {  
 **return new** MessageSource<String>() {  
 **public** Message<String> receive() {  
 **return new** GenericMessage(**new** SimpleDateFormat().format(**new** Date()));  
 }  
 };  
  
 }  
  
}

@SpringBootApplication  
@EnableBinding(Processor.**class**)  
**public class** ProcessorApplication {  
  
 @Transformer  
 **public** String loggerSink(String payload) {  
 **return** payload.toUpperCase();  
 }  
}

@SpringBootApplication  
@EnableBinding(Sink.**class**)  
**public class** SinkApplication {  
  
 @ServiceActivator(inputChannel=Sink.***INPUT***)  
 **public void** loggerSink(Object payload) {  
 }  
}

@SpringBootApplication  
**public class** SampleAggregateApplication{  
  
 **public static void** main(String[] args) {  
 **new** AggregateApplicationBuilder()  
 .from(SourceApplication.class).args("--fixedDelay=5000")  
 .via(ProcessorApplication.class)  
 .to(SinkApplication.class).args("--debug=true").run(args);  
 }  
}

多个channel协同处理消息。

## 多个Binders连接多个mq实例

## 依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-stream-binder-rabbit</artifactId>

</dependency>

## 配置

spring:

cloud:

stream:

bindings:

input:

destination: foo

binder: rabbit1

output:

destination: bar

binder: rabbit2

binders:

rabbit1:

type: rabbit

environment:

spring:

rabbitmq:

host: <host1>

rabbit2:

type: rabbit

environment:

spring:

rabbitmq:

host: <host2>