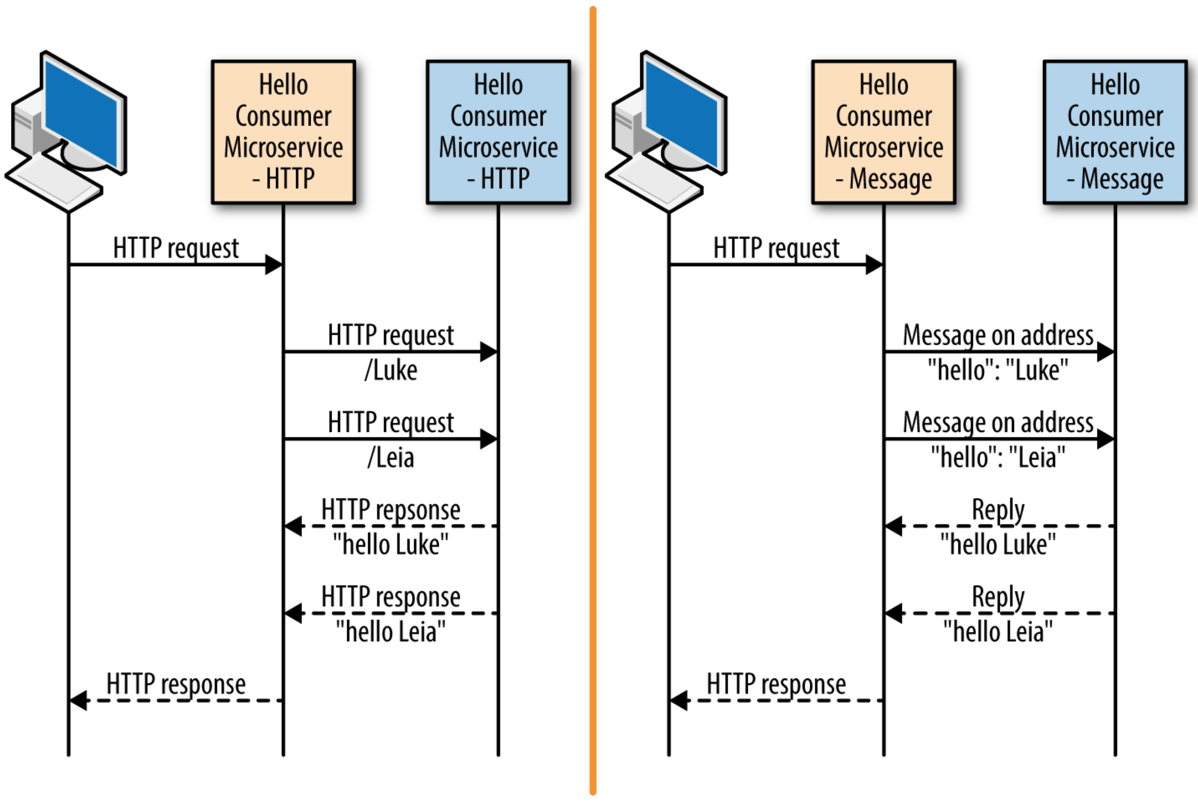
7 Building HTTP & Reactive Microservices using Vert.x

1. hello service
2. hello consumer microservice



Hello Consumer Microservice - HTTP
Hello Consumer Microservice - HTTP
HTTP request
HTTP request /Luke
HTTP request /Leia
HTTP repsonse "hello Luke"
HTTP response "hello Leia"
HTTP response
Hello Consumer Microservice - Message
Hello Consumer Microservice - Message
HTTP request
Message on address "hello": "Luke"
Message on address "hello": "Leia"
Reply "hello Luke"
Reply "hello Leia"
HTTP response

The microservices implemented using HTTP and message-based interactions

Two different ways to use Vert.x APIs: callbacks and RxJava.

the hello microservices are implemented using the callback-based development model,

the consumers are implemented using RxJava.

Implementing HTTP Microservices

```
mkdir hello-microservice-http
cd hello-microservice-http

mvn io.fabric8:vertx-maven-plugin:1.0.5:setup \
-DprojectGroupId=io.vertx.microservice \
-DprojectArtifactId=hello-microservice-http \
-Dverticle=io.vertx.book.http.HelloMicroservice \
-Ddependencies=web
```

Consuming HTTP Microservices

```
mkdir hello-consumer-microservice-http
cd hello-consumer-microservice-http

mvn io.fabric8:vertx-maven-plugin:1.0.5:setup \
-DprojectGroupId=io.vertx.microservice \
-DprojectArtifactId=hello-consumer-microservice-http \
-Dverticle=io.vertx.book.http.HelloConsumerMicroservice \
-Ddependencies=web,web-client,rx
```

Are These Microservices Reactive Microservices?

At this point we have two microservices.

They are independent and can be deployed and updated at their own pace.

They also interact using a lightweight protocol (HTTP).

But are they reactive microservices?

No, they are not

Remember, to be called reactive a microservice must be:

- Autonomous
- Asynchronous
- Resilient
- Elastic

The main issue with the current design is the tight coupling between the two microservices

The web client is configured to target the first microservice explicitly.

If we are under load, creating a new instance of the hello microservice won't help us.

If the first microservice fails, we won't be able to recover by calling another one

It’s important to note that using reactive programming as in the second microservice does not give you the reactive system’s benefits

Can we use HTTP for reactive microservices? Yes.

But this requires some infrastructure to route virtual URLs to a set of services.

We also need to implement a load-balancing strategy to provide elasticity and health-check support to improve resilience

---

📌 The Vert.x Event Bus—A Messaging Backbone

Vert.x offers an event bus allowing the different components of an application to interact using messages

The event bus is also clustered, meaning it can dispatch messages over the network between distributed senders and consumers.

By starting a Vert.x application in cluster mode, nodes are connected to enable shared data structure, hard-stop failure detection, and load-balancing group communication.

The event bus can dispatch messages among all the nodes in the cluster.

To create such a clustered configuration, you can use Apache Ignite, Apache Zookeeper, Infinispan, or Hazelcast

Message-Based Microservices

```
mkdir hello-microservice-message
cd hello-microservice-messagemvn io.fabric8:vertx-maven-plugin:1.0.5:setup \
-DprojectGroupId=io.vertx.microservice \
-DprojectArtifactId=hello-microservice-message \
-Dverticle=io.vertx.book.message.HelloMicroservice \
-Ddependencies=infinispan
```

```
mvn compile vertx:run -Dvertx.runArgs="-cluster -Djava.net.preferIPv4Stack=true"
```

```
mkdir hello-consumer-microservice-message
cd hello-consumer-microservice-messagemvn io.fabric8:vertx-maven-plugin:1.0.5:setup \
-DprojectGroupId=io.vertx.microservice \
-DprojectArtifactId=hello-consumer-microservice-message \
-Dverticle=io.vertx.book.message.HelloConsumerMicroservice \
-Ddependencies=infinispan,rx
```

```
mvn compile vertx:run -Dvertx.runArgs="-cluster -Djava.net.preferIPv4Stack=true"
```

Are We Reactive Now?

The code is very close to the HTTP-based microservice we wrote previously.

The only difference is we used an event bus instead of HTTP.

Does this change our reactiveness? It does!

Elasticity

the Vert.x event bus dispatches messages to the
available instances and thus balances the load among the different
nodes listening to the same address.

So, by using the event bus, we have the elasticity characteristic we
need.

Resilience

What about resilience? In the current code, if the hello microservice
failed, we would get a failure and execute this code:

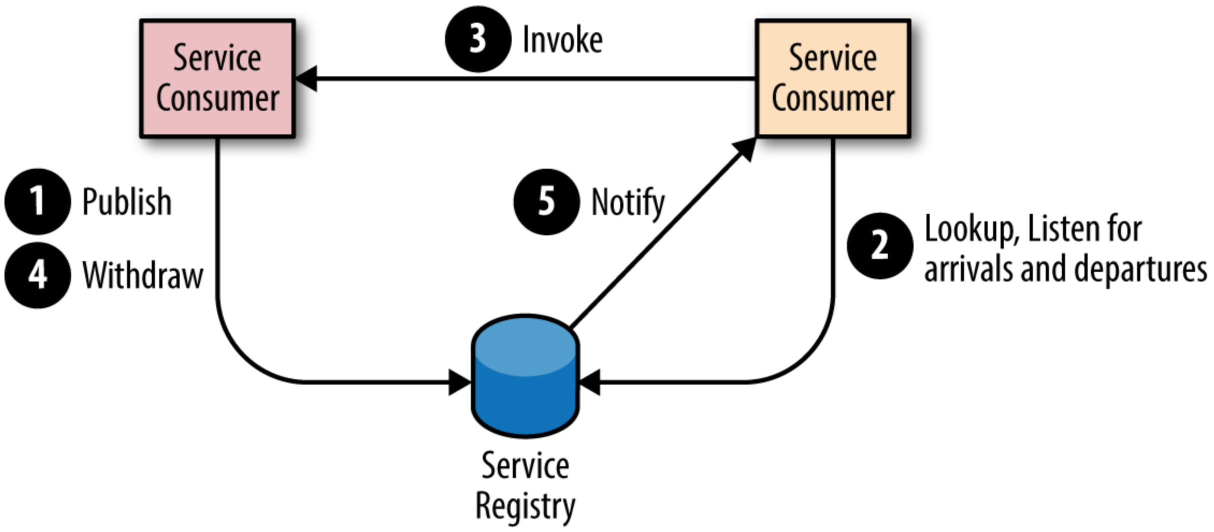
Even though the user gets an error message, we don’t crash, we don’t
limit our scalability, and we can still handle requests

Building Reactive Microservice Systems

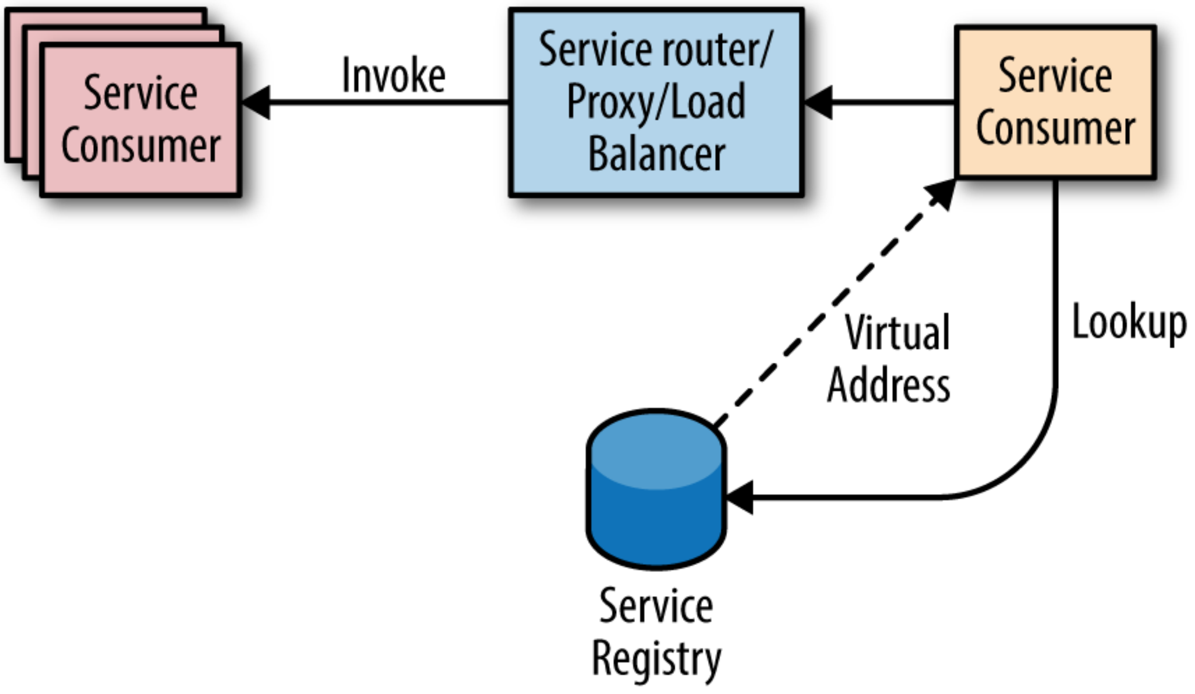
resilience and stability patterns such as timeouts, circuit breakers, and failovers

Service Discovery

Client- Side Service Discovery

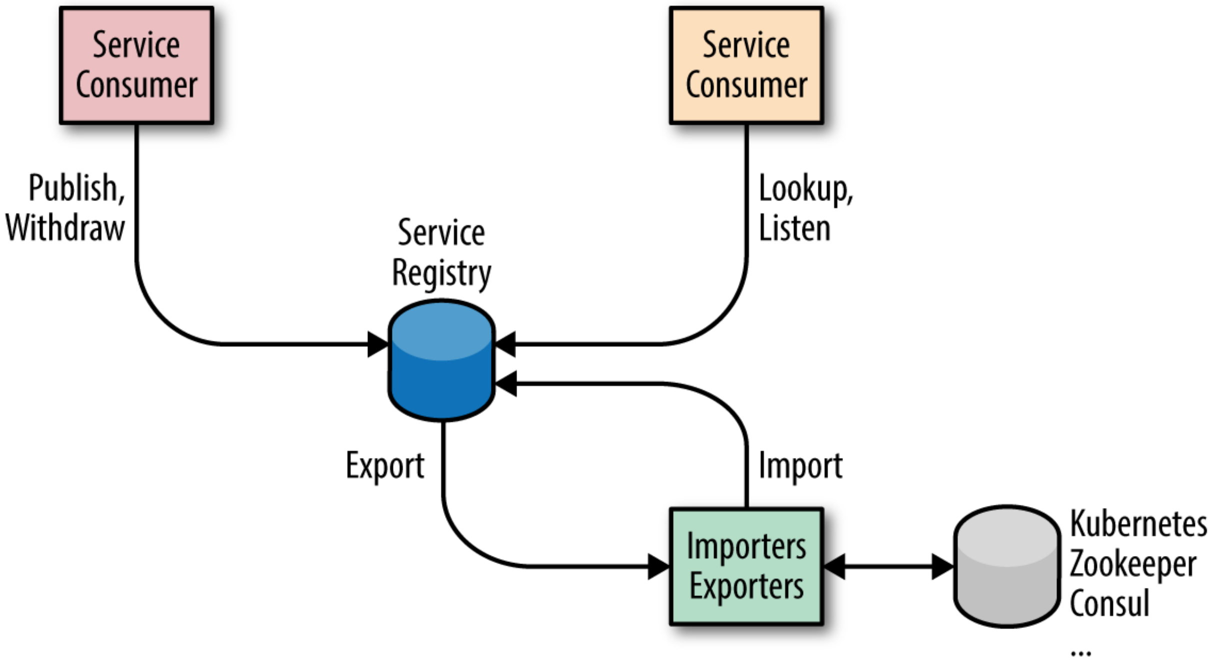


Server-side service discovery



Vert.x Service Discovery

Import and export of services from and to other service discovery mechanisms


Service Consumer
Service Consumer
Publish, Withdraw
Lookup, Listen
Service Registry
Export
Import
Importers Exporters
Kubernetes
Zookeeper
Consul
...