

/ Guides (<https://developer.lightbend.com/guides/>)

 **Akka**
/ Akka gRPC Quickstart with Java ([index.html](#)) / Testing gRPC

gRPC

Testing gRPC

**Quickstart
with Java
([index.html](#))**

0.1.0*

Maven

**Streaming
gRPC
([streaming.html](#))**

**Testing
gRPC
([testing.html](#))**

The tests in the Hello World example illustrates use of the **JUnit** (<https://junit.org/>) framework. The test coverage is not complete. It only shows how to get started with testing gRPC services. You could add to it as an exercise to increase your own knowledge.

Let's look at the test class definition in the `GreeterTest.java` source file:

ON THIS PAGE:

Testing gRPC

Unit testing

Add streaming tests

```

coppackage com.example.helloworld;

import akka.actor.testkit.typed.javadsl.ActorTestKit;
import akka.actor.testkit.typed.javadsl.TestKitJunit5;
import akka.actor.typed.ActorSystem;
import akka.actor.typed.javadsl.Behaviors;
import akka.grpc.GrpcClientSettings;
import akka.http.javadsl.ServerBinding;

import com.typesafe.config.Config;
import com.typesafe.config.ConfigFactory;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.ClassRule;
import org.junit.Test;

import java.util.concurrent.CompletionStage;
import java.util.concurrent.TimeUnit;

import static org.junit.Assert.assertEquals;

public class GreeterTest {

    // important to enable HTTP/2 in server ActorSystem
    private static final Config config = ConfigFactory
        .parseString("akka.http.server.preview.enabled=true")
        .withFallback(ConfigFactory.defaultApplicationConfig());

    @ClassRule
    public static final TestKitJunitResource testKit =
        new TestKitJunitResource(ActorTestKit.create());

    private static ActorSystem<?> serverSystem = testKit.getActorSystem();
    private static ActorSystem<?> clientSystem;
    private static GreeterServiceClient client;

    @BeforeClass
    public static void setup() throws Exception {
        CompletionStage<ServerBinding> bound = new GreeterService()
            .bind()
            // make sure server is bound before using client
            .toCompletableFuture().get(5, TimeUnit.SECONDS);

        clientSystem = ActorSystem.create(Behaviors.empty());
        // the host and TLS certificate config are picked up from the config
        client = GreeterServiceClient.create(
            GrpcClientSettings.fromConfig("helloworld.GreeterServiceClient")
                .withPort(bound.getPort()),
            clientSystem
        );
    }

```

```

}

@AfterClass
public static void teardown() {
    ActorTestKit.shutdown(clientSystem);
    client = null;
}

@Test
public void greeterServiceRepliesToSingleRequest(
    HelloReply reply = client.sayHello(HelloRequest
        .toCompletableFuture()
        .get(5, TimeUnit.SECONDS);
    HelloReply expected = HelloReply.newBuilder().s
    assertEquals(expected, reply);
}

}

```

Note how we create two `ActorSystems`, one for the server and another for the client. The test is then using the client to verify that it retrieves the expected responses from the server.

Unit testing

The above test example is a full integration test using real client and server, including communication via HTTP/2. For some testing of the service implementation it might be more appropriate to write unit tests without interaction via the gRPC client. Since the service interface and implementation doesn't require any gRPC infrastructure it can be tested without binding it to a HTTP server.

```

coppackage com.example.helloworld;

import akka.actor.testkit.typed.javadsl.ActorTestKit
import akka.actor.testkit.typed.javadsl.TestKitJunit5
import akka.actor.typed.ActorSystem;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.ClassRule;
import org.junit.Test;

import java.util.concurrent.TimeUnit;

import static org.junit.Assert.assertEquals;

public class GreeterServiceImplTest {

    @ClassRule
    public static final TestKitJunitResource testKit

    private static ActorSystem<?> system = testKit.system()
    private static GreeterService service;

    @BeforeClass
    public static void setup() {
        service = new GreeterServiceImpl(system);
    }

    @Test
    public void greeterServiceRepliesToSingleRequest(
        HelloReply reply = service.sayHello(HelloRequest.newBuilder().build())
        .toCompletableFuture()
        .get(5, TimeUnit.SECONDS);
        HelloReply expected = HelloReply.newBuilder().build();
        assertEquals(expected, reply);
    }

}

```

Add streaming tests

As an exercise to increase your understanding you could add tests for the **streaming call (streaming.html)**, both as integration test and unit test style.

The Akka documentation of **Testing streams** (<https://doc.akka.io/docs/akka/current/stream/stream-testkit.html>) might be useful.

TECH HUB

LIGHTBEND

Guides

(<https://developer.lightbend.com/guides/>)

Subscription

(<https://www.lightbend.com/subscription/>)

Docs

(<https://developer.lightbend.com/docs/>)

Training

(<https://www.lightbend.com/services/training>)

Blog

(<https://developer.lightbend.com/blog/>)

Consulting

(<https://www.lightbend.com/services/consulting>)

Forums

(<https://discuss.lightbend.com/>)

About Us

(<https://www.lightbend.com/about>)

Get Started

(<https://developer.lightbend.com/start/>)

FOLLOW LIGHTBEND

(<http://www.facebook.com/lightbendOfficial>)

(http://twitter.com/intent/follow?source=followbutton&variant=1.0&screen_name=lightbend)

(<http://www.linkedin.com/company/lightbend-inc->)

(http://www.youtube.com/c/Lightbend-TV?sub_confirmation=1)

(<http://www.twitch.tv/lightbendtwitch>) (<https://www.lightbend.com/blog/rss.xml>)

(<https://github.com/lightbend>)

