Availability testing with DistributeMe

DistributeMe comes with a set of availability testing utils which are realized by interceptors. An additional availability testing utility is the ChaosMonkeyAgent which is currently developed and will be described in a separate article.

Why availability test?

DistributeMe provides a lot of feature to make your service *never fail* - routing, failover strategies, concurrency control and so on. However, how do you know they do what you think they do? And do you know how your system will really behave if a service if not their due to network failure or flipy cable. Another common scenario is a slow answering service, whether due garbage collection (full gc loops tend to eat up all cpu) or slow database or whatever you've got behind the service.

The availability testing utils and there to help you to tune your application to deal with such failures before they actually happen.

How does it work

Current availability testing utils are Interceptors which are added on client or server side, and modify service behavior transparently to your application.

What is in the package

Currently there are two classes of availability testing interceptors, those configured by a config file and those configured by system properties. System properties may be easier to configure, but configuration files allow more complex scenarios, as well as change of the value in runtime, due to ConfigureMe configuration reload features. All built-in interceptors are located in package *org.distributeme.core.interceptor.availabilitytesting* in distributeme-core module.

There are now following interceptors available:

Interceptor Class Name	Purpose
ClientSideSlowDownByConfigurationInterceptor	Slows the request down on the client side. Target service and duration are configured by configuration file.
ClientSideSlowDownByPropertyInterceptor	Same as above, but configured by properties.
ServerSideSlowDownByConfigurationInterceptor	Slows the request down on the client side. Target service and duration are configured by configuration file.
ServerSideSlowDownByPropertyInterceptor	Same as above, but configured by properties.
ServiceUnavailableByConfigurationInterceptor	Cancels all requests to the service with the service unavailable exception as if the service wouldn't run at all.
ServiceUnavailableByPropertyInterceptor	Same as above, but configured by properties.
FlippingClientSideSlowDownByConfigurationInterceptor	Same as ClientSideSlowDownByConfigurationInterceptor but the interceptor only acts with a given chance.
FlippingServerSideSlowDownByConfigurationInterceptor	Same as ServerSideSlowDownByConfigurationInterceptor but the interceptor only acts with a given chance.
FlippingServiceUnavailableByConfigurationInterceptor	Same as ServiceUnavailableByConfigurationInterceptor but the interceptor only acts with a given chance.

Configuration

Enabling interceptors

To enable availability testing interceptors generally, they have to be configured in the distributeme.json as all other interceptors:

```
{
        registryContainerPort: 9229,
        registryContainerHost: "localhost",

...
        "interceptor.1": "org.distributeme.core.interceptor.availabilitytesting.
ServiceUnavailableByPropertyInterceptor",
        "interceptor.2": "org.distributeme.core.interceptor.availabilitytesting.
ServerSideSlowDownByPropertyInterceptor",
        "interceptor.3": "org.distributeme.core.interceptor.availabilitytesting.
ServerSideSlowDownByConfigurationInterceptor",
...
}
```

of course interceptors can be also placed under different environments as all other configurations with ConfigureMe.

```
{
...
  "test": {
    "flip": {
      "interceptor.1": "org.distributeme.core.interceptor.availabilitytesting.
FlippingServiceUnavailableByConfigurationInterceptor",
      "interceptor.2": "org.distributeme.core.interceptor.availabilitytesting.
FlippingServerSideSlowDownByConfigurationInterceptor",
      "interceptor.3": "org.distributeme.core.interceptor.availabilitytesting.
FlippingServerSideSlowDownByConfigurationInterceptor",
    },
  }
}
```

in this case the interceptors will only become active if the component (server or client) is running in test_flip configureme environment. However, the alone presence of the interceptor is not sufficient to make it work. The interceptor (at least the prepackaged) need at least the service id of the target service, otherwise they would intercept all traffic which may be counter-productive.

There are generally two ways to configure an interceptor, configuration file or properties (There is a flavor of each interceptor that supports either configuration).

Configuration file

The configuration file which is used to configure the interceptors is called *availabilitytesting.json*.

Here is an example:

```
{
        "serviceIds": ["org_distributeme_test_echo_EchoService", "org_distributeme_test_whatever_FooService"],
        "slowDownTimeInMillis": 10000,
}
```

Please note that this configuration is reread continuously (all 10 seconds approx.) and can be changed **on-the-fly**.

Of course the standard ConfigureMe environment cascading is supported:

```
{
        "serviceIds": ["org_distributeme_test_echo_EchoService", "foo"],
        "slowDownTimeInMillis": 10000,

        "test": {
                "flip":{
                        "slowDownTimeInMillis": 1000,
                        "flip70":{
                                "flipChanceInPercent": 70,
                        },
                        "flip10":{
                                "flipChanceInPercent": 10,
                        },
                },
        }
}
```

In order to use the configuration you have to submit your environment to the process. You can either do it directly (**not recommended**), via

```
org.configureme.ConfigurationManager.INSTANCE.setDefaultEnvironment(...)
```

or, just supply the appropriate system property (**recommended**):

```
java -Xmx256M -Xms64M -classpath $CLASSPATH -Dconfigureme.defaultEnvironment=test_flip_flip10 <classname>
```

Properties

Using system properties is a bit less flexible, because you have actually to restart the process to make them work, but easier for a quick test, especially server side. The names of the properties are defined in the file

org.distributeme.core.interceptor.availabilitytesting.Constants. You can simply submit them to the process start command.

```
./start.sh -DavailabilityTestingServiceId=org_distributeme_test_echo_EchoService org.distributeme.test.echo.
generated.EchoServer
```

Configuration options

Following configuration options are supported right now:

Name in Configuration File	Name of the Property	Meaning, Values
serviceIds	availabilityTestingServiceId	Service ids. It can be one, a comma separated list, or asterisk. Asterisk means any.
slowDownTimeInMillis	availabilityTestingSlowDownTimeInMillis	Time by which the execution will be slowed down. Default is 10.000 ms.
flipChanceInPercent	availabilityTestingFlipChanceInPercent	Chance in percent for service to flip. Flip chance is only a probability, a flip chance of 50% will not guarantee that every second request fails. The flip chance is implemented as boolean flip = random.nextInt(100)<flipChance.

Running

Running availability testing is not much different from running normal DistributeMe Service environment. For my test runs, I created a script (which is part of distributeme-test) project:

```
#!/bin/bash
export VERSION=1.2.1-SNAPSHOT
CLASSPATH=test/appdata:target/distributeme-test-$VERSION-jar-with-dependencies.jar
echo CLASSPATH: $CLASSPATH
java -Xmx256M -Xms64M -classpath $CLASSPATH -Dconfigureme.defaultEnvironment=test $*
```

Now I only need to start the server:

./start.sh -DavailabilityTestingServiceId= org.distributeme.test.echo.generated.EchoServer*

and the client

./start.sh org.distributeme.test.echo.MakeRemoteCallsForever

and the party starts.

Availability testing interceptors has been added to DistributeMe in version 1.2.1-SNAPSHOT