ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 3 Issue 3 September 2023 / Page No: 113-115
Paper Id: JETA-V3I7P114 / Doi: 10.56472/25832646/JETA-V3I7P114

Original Article

RPC Replay for Robust Testing of Golang Applications

Nilesh Jagnik
Los Angeles, USA

Received Date: 30 July 2023 Revised Date: 28 August 2023 Accepted Date: 25 September 2023

Abstract: *Integration and functional testing of systems is quite important for ensuring reliability and correctness. However end-to-end testing is susceptible to randomness due to interactions with external services, which may behave non-deterministically. This presents a need to create hermetic test environments, so that systems can be tested in a vacuum, i.e., without interacting with external dependencies. This normally requires setting up fakes and mocks to stub responses from interactions with external dependencies. However, stubs and mocks need to be manually created and maintained which adds toil. RPC Replay is a simple solution to this problem. RPC Replay automates the task of creating stubbed responses. The grpcreplay package in Golang provides support for RPC Replay of gRPC calls. In this paper, we study RPC Replay, its benefits, caveats and usage in Golang applications.*

Keywords: *Record And Replay, Software Testing, Software Debugging.*

I. INTRODUCTION

Testing of software is extremely important for ensuring productionized applications are free of errors, vulnerability and performance issues. Thorough testing provides confidence that newly added features and changes do not have an adverse effect on behavior of an application. Catching issues early enhances the reliability and consequently the user experience of clients.

There are many types of testing that can cover different aspects of a system's behavior. Unit testing is the most basic type of testing which tests the behavior of a small part of code – usually a single file. Unit tests are expected to test a single unit of code. In unit tests, interactions with external systems like databases and RPCs are usually stubbed. This is to prevent external factors from interfering with testing and to enable the code to run in a predictable environment.

Apart from unit testing, which is used for testing a single unit of code, integration testing is used for testing interactions between multiple units of code. Finally, functional tests are used to test end-to-end functionality of systems. Integration and functional testing require setting up a hermetic test environment. Hermeticity is needed so that the behavior of the system under test is not affected by external factors during tests. Hermetic tests are more reliable and less flaky because there is no reliance on external dependencies.

Since most systems make external RPCs to components outside the system, setting up a hermetic test environment can be quite a challenge. Normally, fakes and mocks need to be set up for stubbing interactions with external dependencies. Setting up and maintaining fakes can be quite time consuming and complicated. RPC Replay is an easier alternative which requires minimal setup and maintenance work.

With RPC Replay, responses from interactions with external dependencies are recorded. These responses are used to stub interactions with external dependencies when tests are run. This achieves stubbing remote calls without much work required. The grpcreplay package can be used for RPC Replay of gRPC calls in Golang.

II. RPC REPLAY

RPC Replay is done with the help of tools that record and replay RPC calls originating from the system under test to external dependencies. These tools usually have two modes when running a test - record and replay.

A. Record Mode

In the record mode, the system under test is allowed to make RPC calls to real services. The tool then records the requests sent and the responses received for each request. These are then stored in a new file alongside the test files. Any time new RPC calls are added to the system under test (or existing ones are modified), the record mode should be rerun to capture new interactions.

B. Replay Mode

In the replay mode, the record file is read by the tool. The interactions recorded in record mode are used to stub RPC calls to external systems. This ensures that the behavior of external dependencies is fixed in record mode and does not change in replay mode.

III. ADVANTAGES OF RPC REPLAY

A. Hermetic Test Environment

RPC Replay can be used to remove unpredictable behavior caused by external dependencies. Any unexpected behavior in the system under test can not be attributed to external factors and is a result of issues in the system itself. This makes tests robust and reliable.

B. Test Latency

RPC Replay involves stubbing RPC responses from external services. This saves on time by not calling real services from tests. This can make tests much faster, leading to quicker testing cycles.

C. Easy Setup

RPC Replay libraries make it very easy to achieve stubbing of RPCs. With RPC Replay, developers need to manually stub RPCs (or create fakes). Stubs and fakes would have a much high development and maintenance cost. In addition, it is quite difficult to generate real responses manually as external service behavior could be difficult to predict.

D. Performance Regressions

RPC Replay removes performance regressions due to external dependencies and make it easier to detect when the system under test has performance bottlenecks.

IV. RPC REPLAY IN GOLANG

The grpcreplay package in Golang can be used for easy recording and replaying of gRPC calls. Let us go over the usage of this package.

A. Recording

The first thing we need to do is record gRPC calls made by the system under test. To do this, a recorder object must be created. The recorder object has DialOptions which can be passed into gRPC client creation. Note that it is important to close the recorded object afterwards. The recorder saves the observed gRPC interactions into the replay file. Fig. 1 shows an example of recording using grpcreplay. Every time new gRPC interactions are added to the system under test, we need to run record mode again to capture new interactions.

```go
import (
    "github.com/google/go-replayers/grpcreplay"
    "google.golang.org/grpc"
)

func main() {
  // Filename is needed for creating recorder.
  rec, err := grpcreplay.NewRecorder("replay", nil)
  if err != nil { /*...*/ }

  // Recorder should be closed.
  defer func() {
    if err := rec.Close(); err != nil { /*...*/ }
  }()

  // Dial connection using recorder DialOptions.
  conn, err := grpc.Dial(serverAddress,
                          rec.DialOptions()...)
  if err != nil { /* Handle error. */ }

  // Use connection normally
}
```

Figure 1: Recording GRPC Interactions Using Grpcreplay

B. Replaying

Once gRPC interactions have been captured by the recorder (Fig. 1), we use the recorded interactions to create a replayer. The replayer object can serve as a fake connection. In this case, we can skip creating a real gRPC connection. The replayer reads from the replay file and generates responses for RPCs without actually invoking RPC calls on an external service.

```go
// Replayer should use the file produced by recorder.
rep, err := grpcreplay.NewReplayer("replay", nil)
if err != nil { /*...*/ }

// Replayer runs in read only mode. So no need to
// handle error.
defer rep.Close()

// Replayer returns a fake connection.
conn, err := rep.Connection()

// Use connection normally
```

Figure 2: Replaying GRPC Interactions Using Grpcreplay

C. Initial State

While in record mode, it is possible to record some initial state of the system under test. The initial state could contain random values like seeds, time or other state that may differ between test runs. Storing the initial state while recording allows loading this state during replay mode. This can be used to further remove any sources of randomness, making tests even more reliable. The initial state can be stored in the RecordOptions field when invoking the NewRecorder() factory. During replay, the Replayer.Initial() member method can be used to read the initial state.

D. Callbacks

Callbacks can be specified to recorders and replayers who can be used to read and modify requests and responses before they are written to the replay file. Callbacks don't modify the actual gRPC interactions, only what is saved in the replay file. When using callbacks to modify requests, it is important to perform the same modifications during replay as was done during record, otherwise the request will not match the recorded value.

V. CAVEATS OF RPC REPLAY

RPC Replay is a powerful technique which can be used for testing and debugging of systems. However, they come with their own set of caveats:

A. Non-determinisim

The replayer simply matches RPC requests to responses. If the system under test makes multiple RPC calls and a change in the order of these calls can create different responses, then RPC Replay is not the right solution in this case.

B. Concurrent RPCs

If the system under test makes concurrent RPC calls, or streaming RPCs, the replay order may be different that the record order of RPCs. This may make using RPC Replay challenging.

C. Timestamps

If RPCs include timestamps and the application or the external server relies on them to be correct, RPC Replay may not be usable in this case.

D. Causality

The replayer returns the response to an RPC immediately without waiting for any other conditions. So, it is possible that the order of some operations may become different than expected. As an example, in a Pub/Sub scenario, it is possible for the subscriber to be done before the publisher completes publishing.

VI. CONCLUSION

RPC Replay a powerful tool that can be used to make creation and management of hermetic test environments easier. RPC Replay libraries allow easy stubbing of RPC interactions to remove the reliance on external servers allowing integration and functional tests to become robust and reliable. The grpcreplay package in Golang allows easy record and replay of gRPC calls. However, there are certain scenarios which RPC Replay isn't suited to. Being aware of its caveats is quite important while using RPC Replay.

VII. REFERENCES

[1] grpcreplay (Jun 2019), https://pkg.go.dev/github.com/google/go-replayers/grpcreplay

[2] Go Tools for Recording and Replaying RPCs (Jun 2019)," https://github.com/google/go-replayers/blob/main/README.md

[3] Record and Replay Testing vs Scripting: Navigating the Pros and Cons for Your Testing Needs (Jan 2023)," https://www.testevolve.com/blog/record-and-replay-testing-vs-scripting

[4] Grigori Trofimov, "Record & Replay Testing: How to Move Beyond Record-and-Replay for Better Automated API Testing (Apr 2023)," https://www.parasoft.com/blog/best-api-testing-tools

[5] Ioan Solderea, "In A World Of Record And Play, Where Does Playwright Fit In? (Aug 2023)," https://www.ministryoftesting.com/articles/3cfeb5b8

[6] Amy Reichert, "4 ways to use record and playback test automation tools (Apr 2020)," https://www.techtarget.com/searchsoftwarequality/tip/How-and-when-to-use-record-and-playback-testing-tools