HW4

Alternative 2

Watch each of the following YouTube talks on testing, and write three takeaways for each, and answer the talk-specific questions.   (3 points per takeaway/answer)

1. Testing Google AppEngine: talks about testing the Java version of AppEngine, illustrates some of the issues in testing general Java based web services

Answer:

The three takeaways after watching the talk on testing the Google AppEngine are:

i) Previously I had never heard of Google Cloud Cover or the AppEngine Testing framework. The fact that the AppEngine testing framework could be used for local testing and GAE Cloud Cover could be used for Cloud testing was a huge takeaway from this video lecture.

ii) With respect to Google Cloud Cover, it was very interesting to note that the way it is designed is framework agnostic. Also the fact that it does not limit itself to App Engine apps was a interesting takeaway. Just taking into consideration the information about using App Engine as a grid to run any kind of tests is a huge bonus for sure

iii) The Task Queue concept explained during the Google Cloud Cover explanation in the video was a very nice concept to learn. Using Task Queues to handle the load of running the test cases in parallel wherein one task queue task is created per test and executed by the worker is very important to note. Even if we don't use the tool provided by Google, we can still takeaway the advantage of using this design to run multiple test cases

- In what ways is local testing different?

Answer:

As per the talk, in case of Local testing, the RPC layer that's part of the App engine which dispatches all the calls to backend services has assumptions about the environment it executes in.

It assumes that the dev appserver sets up the environment. As a result, if there is no dev appserver. it throws an exception. To fix this problem, we have to either setup the environment by hand by implementing lots of interfaces which is messy. Otherwise we can use the AppEngine Testing APIs to handle this.

This is different from Cloud Testing because in case of Cloud, the environment is as realistic as possible. In case of local testing, the local data store service stores everything in memory. Also we don't get accurate latency for writes and reads if we test locally, compared to what it might when it is tested in Production. Fidelity and Efficiency are the two things which are offered by the cloud testing which can't be accomplished with local testing. If testing is carried out locally, then we have to wait for the tests to finish or set up a test grid ourselves and take the responsibility for maintaining it. However if tests are run in the cloud, then the App engine takes care of handling all this for us and each test is expected to complete in 30 seconds. In case we test locally, we have ensure that we sandbox app code and test code ourselves which is not really the case if it's done through the cloud, as that would be taken care of by the Cloud Cover. In case of local testing we have to set up the environment correctly to be able to carry out tests which is not really a big deal in case of Cloud because all tests are invoked over HTTP using the GAE Cloud cover framework.

Thus these are all the overall ways in which local testing differs.

- What is the isolation problem?

Answer:

A isolation problem occurs when the tests running in parallel are not thread-safe or if the test data is not "thread-safe". This lack of thread safety causes flakiness.

The problem with having such an issue is that it is very difficult to replicate or debug it. Even if it is replicated it might not occur the same way.

To handle this issue, Google Cloud Cover provides the option of using ThreadsafeDelegate instead of ApiProxy.Delegate and Namespaces where the data can be scoped and be contained within that namespace.

2. Testing at Twitter: focuses on testing large scale systems

Answer: The three takeaways after watching the talk on testing at Twitter are:

i) The distinction between Open Vs Closed systems was a new concept to understand for me. The fact that an open system is what we get when requests arrive independent of the ability to handle them as compared to closed systems where there is no external sources of load. So basically the fact that a system is considered an Open system when the service is put on the internet was a good piece of information to know

ii) The concept of "Thrift" being used for a big service oriented architecture was another takeaway from the lecture. I had never heard of "Thrift" before. Thrift is a wire format similar to proto buff which is useful to test systems in isolation. Using Thrift to test components rather than the whole system is helpful. It also supports cross-platform messaging making it harder to generate load

iii) The idea of Iago and the dependency model wherein it goes from system to the load generation library. It is a load generating library and it provides enough details to write to an interface. It comes with an execution environment associated with it. Iago also uses Finagle which is a Scala library used at Twitter and it provides asynchronous RPCs over Thrift or HTTP etc., This was a set of concepts I had not used or heard of previously and it was definitely one of the other positive takeouts from this video lecture.

- What was the primary reason for the Twitter outage that catalyzed the presented work?

Answer:

The primary reasons for the Twitter outage, as per the talk, was the fact the the Twitter code base built using “Ruby on Rails” was monolithic and was not easy to manage or scale. As a result of the monolithic architecture, the desired performance is not easy to achieve because of the blob of code.

One of the other explanations provided for the outage in the talk was related to the usage of Dark Traffic. In this case if the system that is being sent the traffic does not respond in a reasonable amount of time and if the TFE (front end for Dark Traffic) is not configured correctly to handle such a scenario then it caused the outage due to the Dark traffic

- How can you test a system in production?

Answer:

Canaries:

This basically refers to a cluster of systems that are handling requests, which maybe even around 1000. When a new version of the software is deployed to one of them, then it is taking 1% of the traffic. Using this technique we can then observe if the traffic is getting better or worse as compared to the old system.

Dark Traffic:

As opposed to Canaries, this is considered little safer as per the talk. A dark traffic is a system which has a front end called TFE. TFE takes traffic and if configured properly it sends all of the traffic to the normal production systems and it will copy and send a second request to a system that has not been stood up yet. The response received from this second system is thrown away.

Tap Compare:

This is more important to people doing functional testing. It is the ability to do a dark traffic read or a dark traffic write. It then compares the results to make sure that the software that is being stood up is sending the same content as the software that is available in production currently.