newt|global
Aiming At Customer Delight

info@newtglobal.com

CONTINUOUS INTEGRATION & CONTINUOUS DELIVERY

TESTING MICRO SERVICES PART II OF II

Combination of many Micro Services form a complex application. When it is necessary to test each service as a unit and component, it also becomes essential to test the service in an integrated environment and make sure the Application behaves as expected. More emphasis has to be given around User Acceptance testing and monitoring the application as a whole post production. The objective of applying test strategies is a part of implementing Continuous Integration and Continuous Deployment, at the same time to reduce Technical debt in the application.

INTEGRATION TESTING

In a Micro Service environment, each service communicate with other service whenever there is a request and each service has their own data storage. It becomes necessitate to test the integration between these services and interactions between the components. Integration testing tests each service requests and determines the error paths in cases of service failure or timeout errors. Essentially, Integration tests for basic success at each request.

Things to consider:

- Connection failures could cause false errors
- Unit testing and Contract testing can be used to test the behavior of the system
- All these test can be moved to the build pipeline as part of Continuous Integration

Integration tests can be executed once the build is pushed to the staging environment after running the Unit, Component and Contract tests. As basically, after the services are tested individually, we need to make sure that the interactions between them are also verified. Integration test collects all the module to exercise communication paths and check for errors at each module when they interact with other services. This layer of testing is critical in a micro service architecture as it ensures that the system works well and the dependencies with the external services behave as expected.

END TO END TESTING

In order to verify the system meets the external requirements, an end-to-end test is required. In a state-full system where there lot of calculations, algorithms and behavior involved, there are chances that any of the test doubles go missing or the black box testing that was done stands in-effective when you put together to a single state. Setting up an end-to-end test is more of a sanity check to make sure that all of the other test haven't missed anything.

An Automated User Interface testing also becomes a part of the end-to-end testing and yet cannot be ignore as this is going to be the face for the consumer. Manual UI Testing becomes rigorous when there is a build released each day. There are various open source tools to do this job and this can integrated as part of the DevOps pipeline to perform a User Acceptance Testing and validate the Test case results. An end-to-end is also followed by non-functional tests, such as testing for Security, Stability and Performance. Whilst in most of the cases, this is executed less frequently, however can be combined with a specific set of protocol or use case.

CODE TESTING

In most of the scenarios, the job of a Developer ends when the code is checked-in. How do we make sure that the Developer has followed the right code standard? Drive adoption in Development team with security self-service approach that is easy to configure and enables parallel execution. In a complex application where there are multiple developers working to develop various micro services with many new build ready for deployment, it gets difficult to keep a track on the code when there is a bug. The job gets tedious to allocate the bug to the concerned developer. An Intelligent ticketing and tracking solution with # tag conceptualization reduces the manual effort and essentially automates the entire code check-in/check-out process.

Things to Consider:

- Auto Triage and Framework with baseline plugins
- Reporting with a Dashboard view
- Pre-Commit and Post-Commit Testing Stages

APPLICATION MONITORING

Change requests and new features keeps piling, with the Developers and Testers having less time to react when there is a production issue. In most of the cases, unknown errors and downtime issues keeps the production team busy in supporting the applications with a build that almost passed the quality 100%. When one of the few service fail to communicate with each other or do not respond to any of the request over HTTP, then we would have to deep dive into the application to do a root cause analysis. An Application Monitoring Framework takes the pain of the Production Team by discovering such bottle-necks in live scenarios and provide regular updates to the Development team on the ongoing issues.

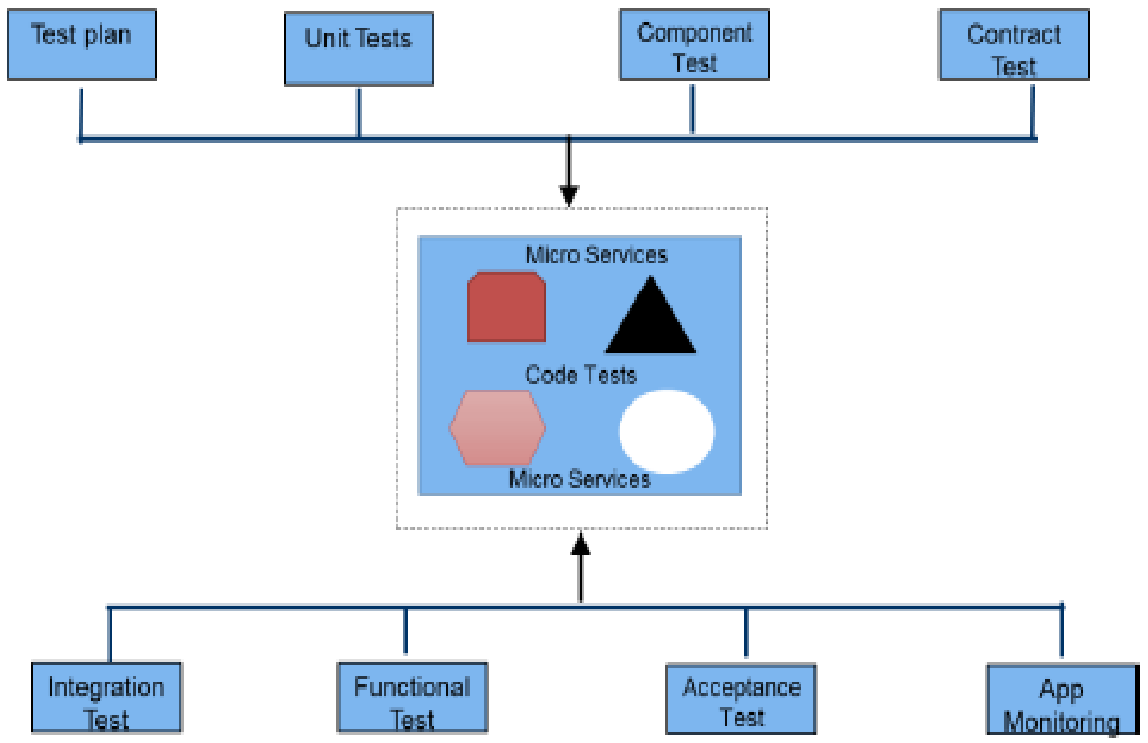

Fig.: Application Test Strategy – Micro Service