V3 search engine test plan

This test plan requires the creation of 3 test suites.

1. **Smoke**

Very quick to run. One user is added and searched, and the record is returned. This is simply to test that the service is reachable and that result is reported. It is ideal for this to be very quick to run to inform devops ASAP.

Medium

A much larger test suite then "Smoke ". Focusing on V3 search engine itself. Searches will executed on all the main user fields using the various query syntax.

2. **Full**

Very extensive regression suite that checks that the surrounding functionality of the endpoints has not been accidently changed or affected. For example:

Deliberately exceeding the ***X-RateLimit-Limit*** to verify expected behaviour.

Obtain the ***X-RateLimit-Remaining*** and ***X-RateLimit-Reset*** values and verify expected behaviour as per documentation:

> *"Each new request reduces this number by 1. For each minute that passes, requests are added back, so this number increases by 1 each time."*

Changing or removing scope from the API or API users and ensuring expected behaviour.

NB. The terms Medium and Full have been used here simply to indicate relative size. In practice a business specific name should be used.

Backward Compatibility

In addition to the three test suites there should be a backward compatibility test. This is simply a staging sever running current production code but running the DB from the next iteration. A test suite should then be run against this server. This test is to secure zero downtime deployments.

The process for checking a Release Candidate

1. Jenkins will to schedule a deployment to staging server of both the new search service code and the test suites.

2. Jenkins will run the test suites conditionally (on success of previous suite) in sequence: Smoke, Backward Compatibility, Medium, Full.

3. SDET to review any failures of the test suites and discuss the error logs with the search team.

Manual exploratory testing

Manual exploratory testing, is a very important part of the test effort. It is an opportunity for quality advocates to use their intuition, and to take on the viewpoint of the end user. Exploratory testing is most effective when done using the test charter methodology. With very large applications the test charter data and results should be subjected to meta-analysis to inform future test charters. NB. This is more for UI testing and not applicable to the v3 search engine.

Non-specific Quality Advocacy

In addition to the automation detailed above, some additional real world steps could be introduced:

A quality advocate should be able to review pull requests. The purpose is not to check the code, but to see that unit tests have been sufficiently altered/created. If there is new code but no new unit tests, the question should be asked if this is an omission or legitimately not required.

If there is a policy that rejected pull requests should have a detail explanation with reference to a Jira story (for example) then a quality advocate can ensure that this has been followed.

There should be close communication between quality advocates and infrastructure teams in the lead up to deployment. Specifically with regards to any changes that are being made from ***either*** side. If infrastructure are patching/updating servers or changing VM templates, or upgrading firewalls it is essential to coordinate so that this isn't done on the same day as a major release (or even the same hour). Conversely if a new feature release creates a new endpoint, or new content type, this may need to be whitelisted in the firewall by infrastructure. It should be possible to utilise the backwards compatibility staging environment to test the effect of any planned changes.

NB. Having learned a bit more about NodeJS this week, I feel the following suggestion is not within scope. It could be an idea for research and development.

Performance testing can be implemented at a very early stage for example to introduce lightweight, idempotent UDP port calls to a listener that logs all the received packets into mongoDB. This can be used for performance trend analysis or at very least a bad smell detector. The 'canary in the mineshaft'. The same type of calls can be made from the test suits so you compare the performance profile alongside the profile of the usage in that same time frame. If this is implemented then every test becomes a performance test.

Other considerations

An investigation to ascertain how many enterprise customers are still using the search engine V2. Although this has been deprecated since 2019 if it is deleted we need to understand the impact with regards to SDKs and new limitations, for example v3 limit of 1000 users per query, or that wildcard searches are no longer available on app_metadata/user_metadata fields. Appropriate communication should be made to avoid customer support having millions of support tickets raised.

Status of gates can be made available through a dashboard for consumption by stakeholders outside of the Quality team. For example a web page that shows a named pipeline stage with simple 'traffic light' colour code. This tends to be a very popular feature with managers but test flakiness is a problem. Credibility is lost if every time a gate shows red you have to tell everyone that it's actually green but just flaky.