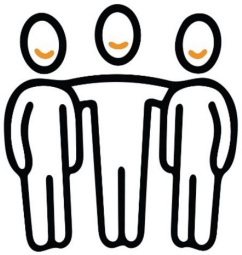
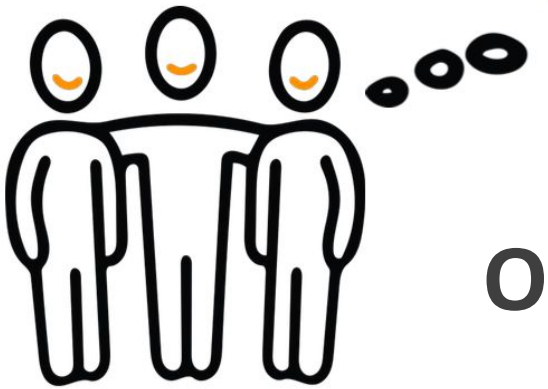
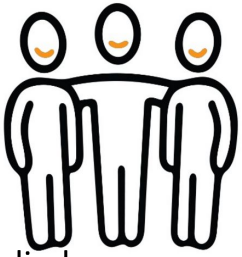




One Day Architecture Review

Client: X
GRO.TEAM: Y
Length: 1 day
Date: 5-6 March 2018
Version: Redacted as a sample

Executive Summary



- This one day analysis is the result of the review of architecture “future state” documentation supplied by X on March 2nd 2018. No other analysis or discussion was possible given the short length of the exercise.

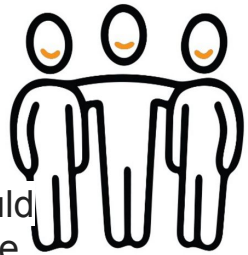
- The documentation supplied seems to suggest that a healthy, comprehensive, detailed and objective review of the current architecture and possible approaches for the future state architecture is being conducted by X and Y.

- In our opinion whilst the broad thrust of the approach and design (loose coupling, variable capacity compute resource, message passing, isolation and reuse of existing platforms) is “spot on” we would agree with the “musings” in the documentation wondering whether this architecture pattern could be achieved using simpler (and more mainstream) components.

- In particular we would question the advantages of;

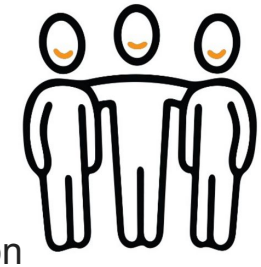
    - Building a stateful pub-sub rather than stateless event-driven future state architecture.
    - The use of grid technology.
    - Deploying an ESB.

- The overall architecture approach is viable, however other technologies could yield a more future proof and resilient result that could affect factors such as scaling, performance and cost.



General thoughts/questions.

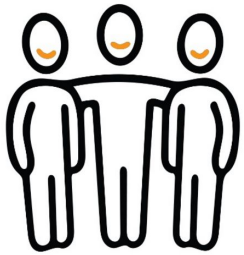
- We would recommend using a mix of REST and Microservice based architecture that would utilise event driven systems for messaging and notifications, rather than it being one vs the other. The nature of the system would seem to lend itself to distributed architecture, and there are clear distinctions between the level of service required between a grid and a data warehouse.
- Stored procedures seem to be being used to raise notifications/invalidations - they should be replaced or wrapped by Go/Java/other middleware tier for more flexibility/ability integration.
- Database scalability - is the iVector database shardable or distributable? We imagine the stored procs would limit this - further investigation is advised. Distribution, sharding, replication and backups is a must.
- The FSA looks inherently single datacentre/availability zone - what are the DR scenarios? Is Active/Active functioning ever desired? While the iVector system is in place, Active/Active for bookings would be unlikely, but the grid/search system should definitely be possible to be active/active with an appropriate local cache layer.
- The security model (authentication/authorisation etc) for clients is not clear. Given the number of potential third parties that should be able to call the system, some form of authentication, ACL is required. Additionally crucial systems communication could be encrypted.
- What is the proposed purpose/use of the CDN? It's normally for public data unless you are proposing to layer some sort of API management in front of it.
- Most people would consider C# as a "sunset" language now. Additionally choosing a Microsoft solution will determine the operating system whereas languages like Java or Golang should enable additional technological solutions like virtual system containerization.
- A micro services software design paradigm is advocated - particularly if a stateless message based system is implemented.



The use of grid computing thoughts/questions….

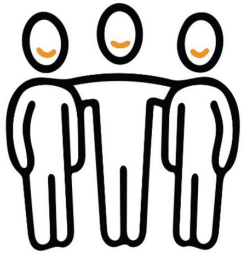
- The investment to deploy a grid approach is high (£250k). Is it value for money?
- It goes without saying that there is better reliability/scalability/distributability out of a non shared memory solution if everything can fit in single instance memory.
- We think it's questionable overall whether there is actually a need for a Grid - given the dataset (20GB) fits trivially in memory for most modern servers/virtual machines, the correct selection of indexes and data structures would probably double this - let's say 64GB to be safe - still safely within commodity box sizing.
- The use case described could probably be achieved using a grid approach - although it feels like the scenario is composited of a number of results with enhancements which are probably distinct from the grid requirement itself.
- Amazon Solutions like Lambda or instance auto scaling feels like a natural solution - especially if it is a periodic burst up/down scenario described.
- If it's largely a batch processing challenge then there are probably more efficient ways of doing it.
- Remaining on .Net, the use of a functional language like F# would be an option to take advantage of single server multiple cores without necessarily needing to grid process (with its additional overhead, brittleness and complexity).
- We would advise a consideration of different languages other than F#, C# or .NET as they will lock down the choice of the operating system though.
- It would be necessary to put an explicit API over the top of the compute grid infrastructure.
- It would appear that the grid is largely "read only" from a user/search perspective, even if the underlying data is volatile. This would imply a level of “cacheability” of search results - would be good to do some light weight benchmarking to see whether precomputing results gains anything versus ad hoc computation + caching.

The use of grid computing thoughts/questions...contd



- What sort of loads are expected? There is an implicit assumption that the load will be high without any traffic shape information e.g. if the searches are largely for the same result vs each search is completely bespoke. It's unlikely that all searches are completely different (eg loading a default page will probably yield the same data for most clients).
- Different caching methods for different scenarios are advised. Pages can be cached partially and dynamic content inserted. Technologies like Varnish are worth considering.
- Is there any natural sharding of the data? Data resilience must be provided. Sharding, replications and backups must be in place to provide robustness.
- What's the data caching strategy? I'm sure there is one but it's hard to evaluate the potential solutions without understanding it.
- Is there any requirement for personalised searches? How does this affect the dimensions of the data being searched.
- Does GridGain offer ACID compliance? It seems to.
- What is the default search engine? Are we considering using different search engines, SOLR, Elastic Search. Is there a need for any additional features like document similarity, full text search?
- Are there any infrastructure constraints? Is the hosting solution limited to privately hosted servers or is it possible to use cloud based hosting like AWS, Rackspace, Digital Ocean. Were the infrastructure configuration tools taken into account? Puppet, Chef, Ansible?

The use of an ESB thoughts/questions.

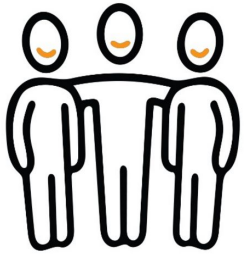


- Why is an ESB being proposed? The modern trend is for fast dumb pipes. An ESB looks like overkill for this use case. We're not clear why an ESB over a message bus (eg RabbitMQ/NSQ etc) would be necessary - the implicit assumption about reliable/transactional delivery needs to be investigated further to establish whether this is the only usage pattern - far more likely that there is are a number of service levels/quality of services involved.
- Are we conflating 3 activities?

  Namely
  - Notifications which would require an action by a downstream system.
  - Cache updates.
  - Firehose-like functionality to populate the Data Warehouse.

- Third party data availability data should be distributed via the message bus/firehose - any "grid" should ideally be fed purely via this - also allows testing of new deployments of the grid without having to replicate everything. A cache/microservice for reading from cold for a new startup on the grid is needed as well.
- Having Stored procs place things on the message bus reliably will significantly increase latency of book() etc calls since there will need to be a distributed 2 phase commit to do this atomically.
- There should be some well defined concepts for what to place on the message bus - an implicit assumption is interoperability between message creators/consumers.

Conclusion



The overall thinking described in the documents seems robust and a good culture of open discussion/debate to drive to optimum solutions seems to exist on the future state architecture project.

We question whether some of the technology selection options described in the documents would be the right ones.

In particular;

Grid approaches are complicated, brittle and expensive and have largely been superseded by non shared memory solutions and cloud horizontal scaling.

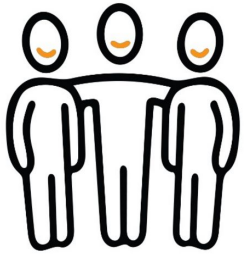
The modern trend is to deploy fast dumb pipes and abstract complexity into micro services. ESB usage is definitely on the wane and only ever really used these days for vendor integration reasons.

We didn't see anything in the documents that implied a REST pub/sub approach would not work for the use case and this is by far the most common solution taken to these kind of use cases.

Thank you for inviting us to join the discussion. We only had one day to look at it but we enjoyed reviewing the landscape and options.

Please give us a shout if we can be of any assistance in the future.

Appendix



Some links to write ups of the experience we gained building idempotent event driven micro services platforms at Hailo and other places;

https://sudo.hailoapp.com/services/2015/03/09/journey-into-a-microservice-world-part-1/

https://sudo.hailoapp.com/devops/2015/07/01/microservices-a-devops-architecture-at-hailo/

http://www.infoq.com/articles/microservices-gilt-hailo-nearform

http://www.computing.co.uk/ctg/news/2404340/hailo-we-built-hailo-on-aws-and-here-s-why-we-ll-keep-using-it

http://www.slideshare.net/nathariel/aws-summitlondonhailo2014

http://martinfowler.com/articles/lmax.html

https://GRO.TEAM/2-0-projects-always-a-graveyard-for-ambition/

https://GRO.TEAM/how-to-select-new-technologies/