Where is the risk?

Stuart Ashman – VanQ Discussion

Introduction

- What is Risk Based Testing?
- How do I approach it?
- What tools and techniques do I use?
- What are the results?
- So what?

What is risk based testing? - traditionally

- An approach used to prioritize testing to focus on the areas where risks have been identified in order to;

- Reduce the risk to an acceptable level
- Ensure you are spending time on the highest risk areas first
- Sometimes used to estimate effort to mitigate risks
- Facilitate test design (choose the parameters that expose the risks)

What is risk based testing? – for me

- An efficient way to test – only spend your time on the highest risks
- One of the best ways to focus on defect prevention rather than defect detection
- Provides confidence and clarity for release decisions

How do I approach risk based testing?

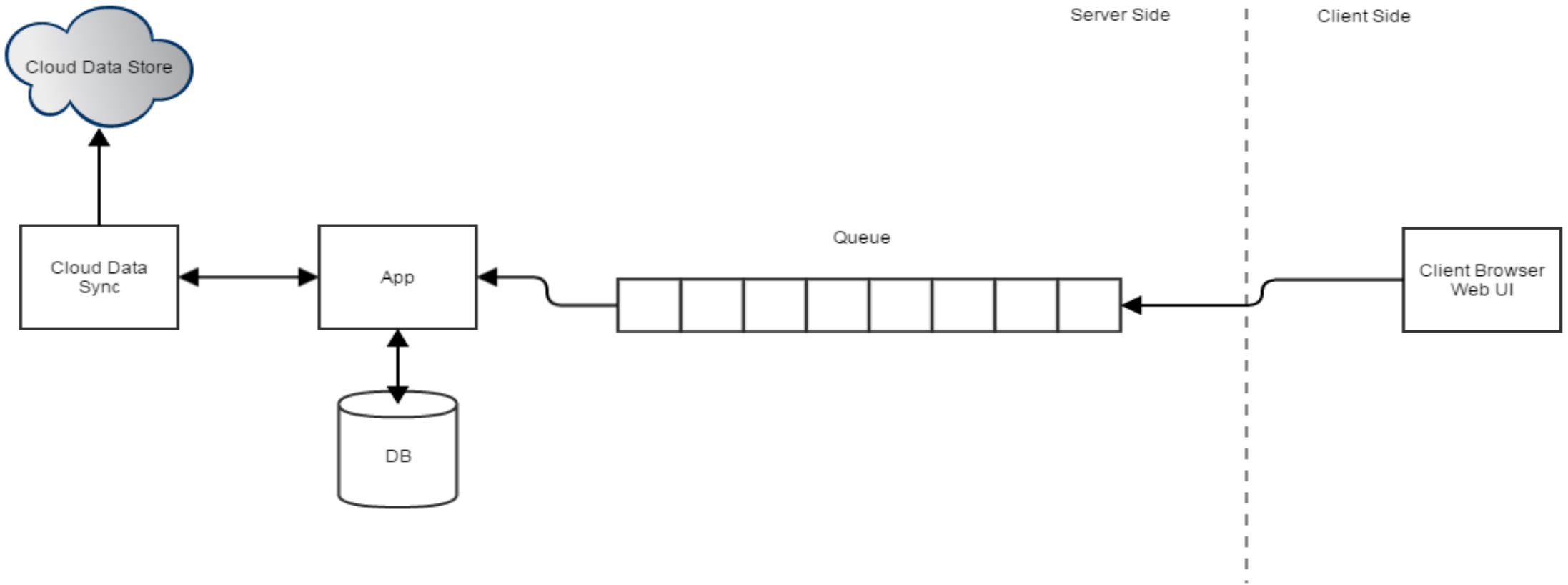
- Ensure you start before code is designed and written
- Begin with questions about what is changing and why
- Develop your understanding
- Draw diagrams and model the system or the areas of change
- Apply a heuristic to help form more questions and to deepen your understanding
- Identify risk areas
- Assess likelihood and impact of those risks
- Focus on the highest areas of risk and try to isolate them and ask more about them
- Discuss mitigations for the risks you have identified

No really, how do I do that?!

•Start at the beginning, with the requirements, and ask good questions

- What problem is this change/new thing solving for the customer?
- *Get a description of the benefits and values*
- What problem is this change/new thing solving for us?
- *Get a description of the benefits and values*
- Why is this important for the customer?
- Why is this important for us?
- How are we solving the problem? What are we changing in our solution to deliver what the customer wants?
- Get someone to draw a diagram (or try it yourself and ask for feedback)

Now what? – analyze the diagram



Client enters data via the client browser web UI. The data is entered into a queue on the server. The App pulls data off the queue and processes it before storing it in the DB. The Cloud data sync periodically asks the App for any new data since the last sync and the App retrieves this from the DB and passes it to the Cloud Data Sync. The Cloud Data Sync then pushes this data into cloud based stores.

What questions should I ask?

- Quality Criteria Categories
- A form of requirements check, challenge the solution with each of the following and expose any unknowns/uncertainties as risks that will need mitigating.

- Data flow - use whiteboard diagram and follow where the data is going between components and think about what if corrupt, not flowing, slow, fast, large etc
- Reliability. Will it work well and resist failure in all required situations?
- Usability. How easy is it for a real user to use the product?
- Performance. How speedy and responsive is it?
- Compatibility. How well does it work with different browsers?
- Supportability. How easy will it be to provide support to users of the product?
- Testability. How effectively can this be tested?
- Maintainability. How easy will it be to build, fix or refactor this solution?
- Upgrade. What ramifications does this have for upgrading the production system? DB changes? Decisions based on pre-existing versus new data?
- Delivery. Feature flags? Staged release?
- Portability. How easy will it be to re-use this solution elsewhere in the product?
- Localizability. How easy will it be to localize?
- Accessibility. For example can it be accessed using a screen reader?
- Monitoring. How observable is this functionality? Can we monitor health, status or otherwise predict or diagnose failures?

And?

- Generic Risk List
- Generic risks are risks that are universal to any system. These are my favorite generic risks:

- Popular: anything that will be used a lot or used as part of the usual/expected workflow for a customer (considered as critical path through the product to get their work done)
- Complex: anything disproportionately large, intricate, or convoluted. High number of classes, complex state handling or transitions, lots of decisions
- New: anything that has no history in the product.
- Changed: anything that has been re-factored or “improved”.
- Upstream Dependency: anything whose failure will cause cascading failure in the rest of the system. i.e. if this fails can it bring the system to it’s knees
- Downstream Dependency: anything that is especially sensitive to failures in the rest of the system. i.e. is this code dependent on anything else and how could a failure in that dependency affect this.
- Critical: anything whose failure could cause substantial damage.
- Resource utilization. What resources will this use (CPU, Memory, Disk etc), and what prevents overuse
- Precise: anything that must meet its requirements exactly.
- Strategic: anything that has special importance to the business, such as a feature that sets us apart from the competition.
- Third-party: anything used in the product, but developed outside the project.
- Distributed: anything spread out in time or space, yet whose elements must work together.
- Buggy: anything known to have a lot of problems.
- Recent failure: anything with a recent history of failure.

More? – come up with your own!

•Risk Catalogs – domain specific and best sourced from your own defect experience

- What problems do you keep seeing with the product?
- Are there areas of the code that are fragile?
- Are there areas of the code that developers don’t like working on?
- Do we often forget about some important non-functional requirement?

Use a Heuristic!

•SFDIPOT or SFDEPOT or SFDPOT - http://www.satisfice.com/tools/htsm.pdf#page=4

- Structure
- Function
- Data
- Interface or Environment
- Performance or Platform
- Operation
- Time

Another Heuristic

FEW HICCUPPS - http://www.developsense.com/blog/2012/07/few-hiccupps/

- Familiarity
- Explain'ability
- World
- History
- Image
- Comparable Products
- Claims
- Users' Desires
- Product
- Purpose
- Statutes

And more …

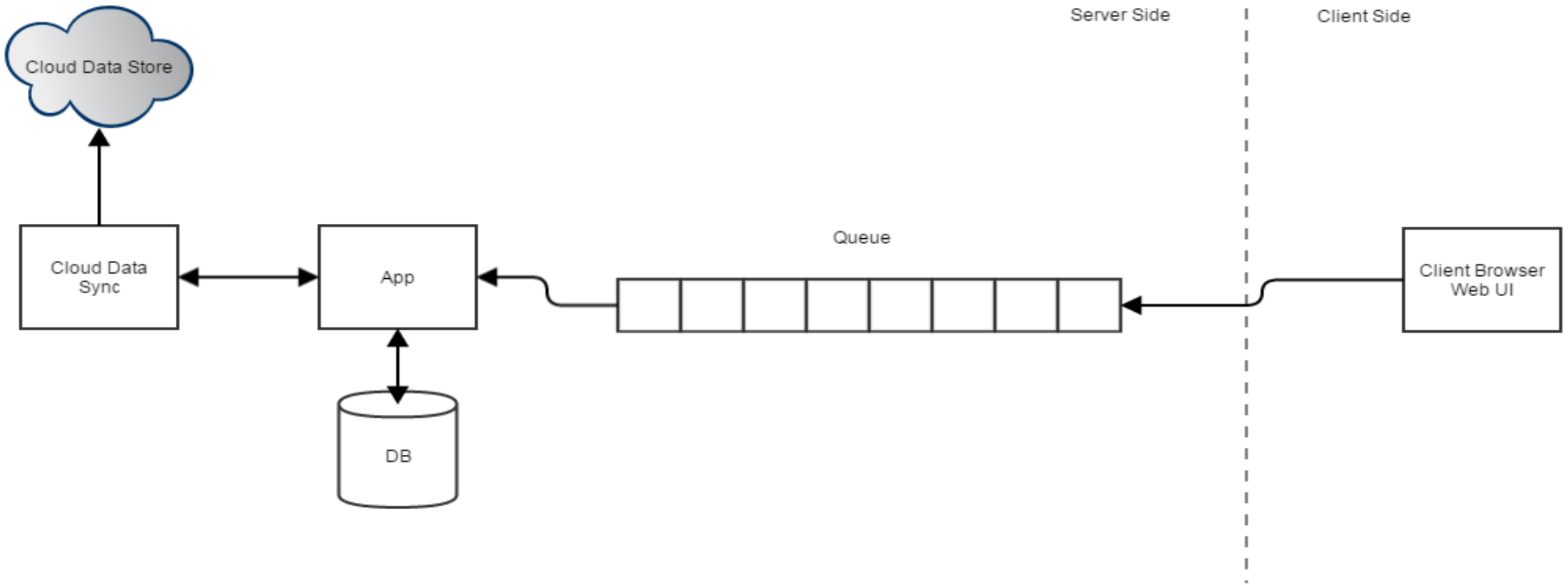
- UNUSUAL PAGE - http://qa-matters.com/2016/07/29/unusual-page-a-web-ui-test-heuristic/

- VADER - http://qa-matters.com/2016/07/30/vader-a-rest-api-test-heuristic/

How do I evaluate the risk?

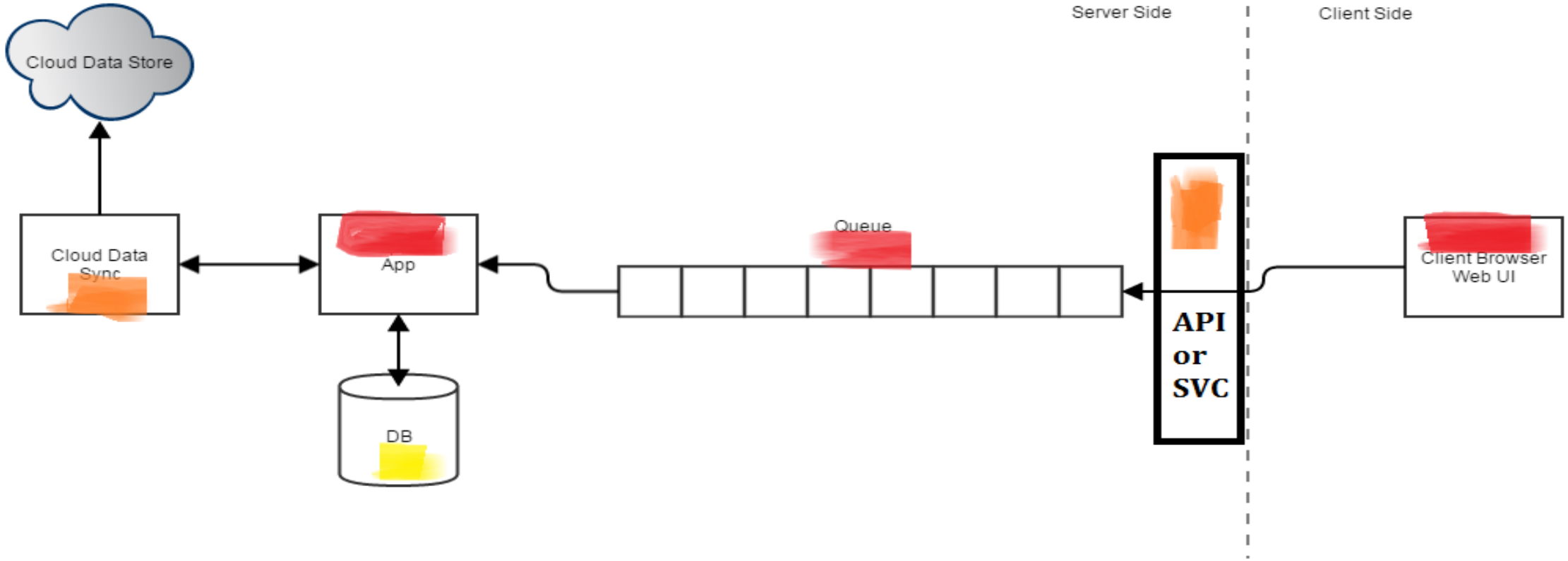
		A	B	C	D	E
		Negligible	Minor	Moderate	Significant	Severe
E	Very Likely	Low Med	Medium	Med Hi	High	High
D	Likely	Low	Low Med	Medium	Med Hi	High
C	Possible	Low	Low Med	Medium	Med Hi	Med Hi
B	Unlikely	Low	Low Med	Low Med	Medium	Med Hi
A	Very Unlikely	Low	Low	Low Med	Medium	Medium

How do I apply those to the diagram?



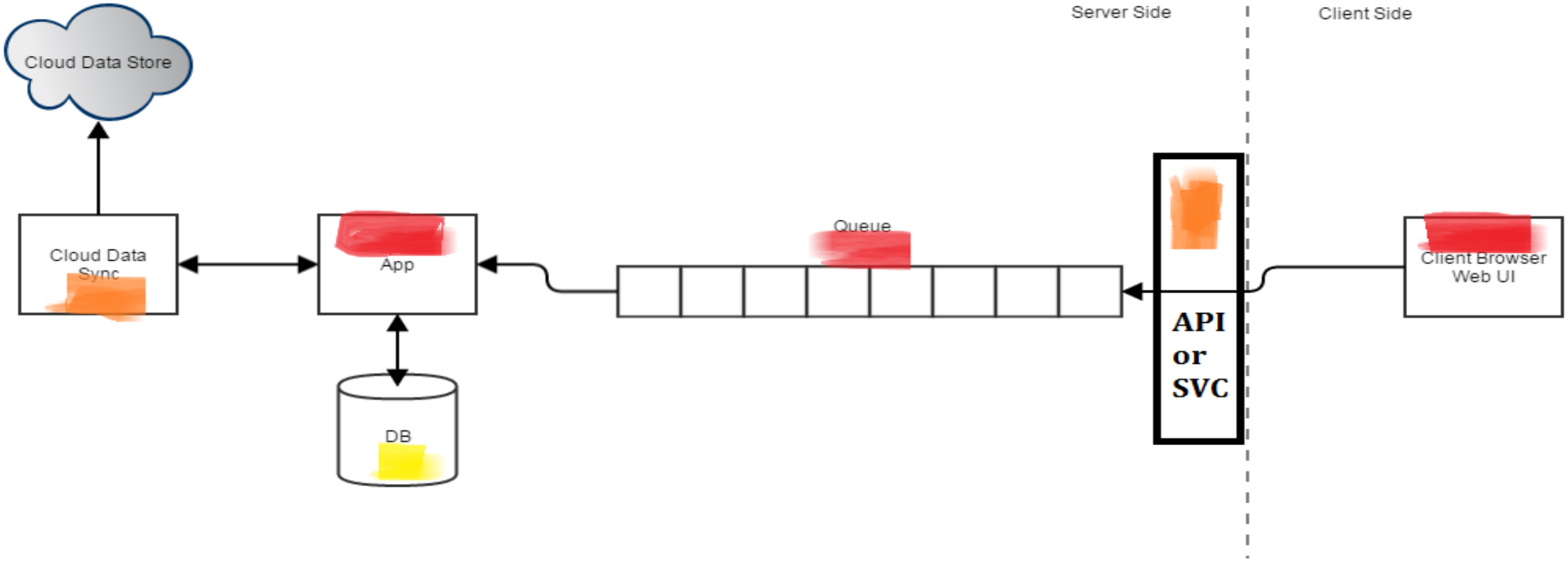
Client enters data via the client browser web UI. The data is entered into a queue on the server. The App pulls data off the queue and processes it before storing it in the DB. The Cloud data sync periodically asks the App for any new data since the last sync and the App retrieves this from the DB and passes it to the Cloud Data Sync. The Cloud Data Sync then pushes this data into cloud based stores.

What do the results look like?



Client enters data via the client browser web UI. The data is entered into a queue on the server. The App pulls data off the queue and processes it before storing it in the DB. The Cloud data sync periodically asks the App for any new data since the last sync and the App retrieves this from the DB and passes it to the Cloud Data Sync. The Cloud Data Sync then pushes this data into cloud based stores.

Now dig into the highest risk areas first



Client enters data via the client browser web UI. The data is entered into a queue on the server. The App pulls data off the queue and processes it before storing it in the DB. The Cloud data sync periodically asks the App for any new data since the last sync and the App retrieves this from the DB and passes it to the Cloud Data Sync. The Cloud Data Sync then pushes this data into cloud based stores.

What did I cover?

- Definition of risk based testing and the benefits to applying it
- Tools and techniques – questioning, diagrams, heuristics, risk matrix
- Some quick examples of how to apply tools and techniques
- Some results;
  - Diagram(s) highlighting levels of risk
  - Ability/plan to focus time in order of risk priority
  - Defects prevented by greater and shared understanding – possibly also by design
  - Test cases designed to mitigate the risks (these can be also prioritized)
  - Test results mapped to risks for release decision clarity

So what? … if I don’t do this?

- Slow feedback cycle
- Only test what you know
- Only testing how you know
- Not finding the important issues
- Not helping the team to work efficiently
- Not able to provide risk based test results (only coverage based)

Questions?

Want to know more?

Qa-matters.com

qamatters

Stuart Ashman