More on Testing
and
Large Scale Web Apps

Testing

- Functionality Tests
  - Unit tests: E.g. Mocha
  - Integration tests
  - End-to-end - E.g. Selenium
  - HTML CSS validation
  - forms and form validation
  - cookies - test for correct operation and deletion

Testing

- Usability testing
  - need real people to test your application
  - navigation
  - consistency
  - content

Testing

- Interface testing
  - application - requests sent to server and results displayed
  - web server - handles all requests correctly
  - database server - queried give expected results
  - when connection between layers fails, appropriate error messages
- Database testing
  - integrity
  - response time
  - data retrieved is shown accurately

Testing

- Compatibility testing
  - works on all major browsers, including older versions
- Performance testing
  - load test - normal and peak
  - stress test - push beyond peak
  - crash recovery
- Security testing
  - regularly audit!

Data-driven decision making

(Kate Hudson - Mozilla)

- In the world of continuous deployment, how and when are decisions made?

Healthy software?

- Doesn’t crash (very often)?
- Trustable data
- Preserves data
- Easy to use
- Performs well (?)
- Handles errors gracefully

Performs well?

Goal: Minimize user-perceived delay

- Reduce time to fetch a resource
  - server processing time
  - browser caching
  - HTTP pipelining
  - minimize JS resources and use shared libraries
- Optimize rendering speed
  - layout
  - CSS
  - Javascript optimizations
- Make loading time appear shorter
  - loading indicators
  - deferred loading

Monitoring

- Detecting when software is not healthy

- Monitoring software: CPU, memory, #crashes
- Look at HTTP response codes: 500, 400

Roll-out

- Server-side: Canary release pattern
  - update one server, check monitoring
- Client-side:
  - staged roll-out - to a small group of users
    - must be able to split up users
  - some percentage get new code
  - by single country or language (not en-us)
  - opt-in release channels (beta channel)
    - can collect more data, bug reports, feedback

Firefox

- Different versions depending on risk
  - beta, nightly, aurora
  - users expect some instability

- Prefs - about:config in Firefox to turn on/off experimental features

- Test Pilot - add on where users can try out new features

Measure User Behaviour

- User engagement:
  - track clicks
  - retention

AB testing

- Control group and experimental cohort - users unaware of participation
- Change some aspect of the application:
  - order of operations
  - amount of information shown at one time
  - placement of buttons
  - text of buttons
  - choice of images

AB testing

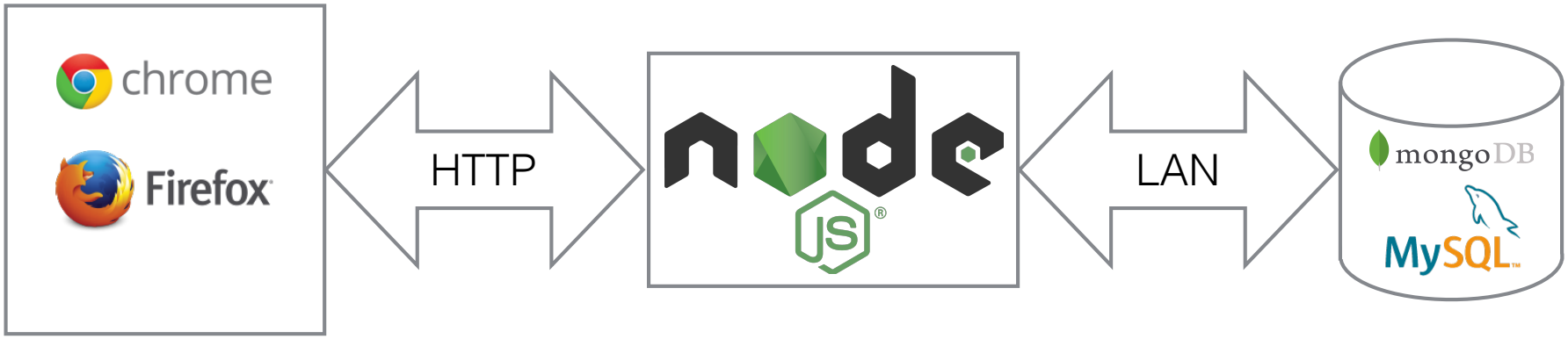
- Important to run one experiment at a time
- Run long enough so that statistical significance
  - aim for 90-95% certainty
  - need a lot of users to do this
- Results must be correlated with right metrics
  - sometimes conclude that neither is more effective
  - sometimes see unexpected effects - e.g. a huge drop in engagement
  - novelty effect

Ethical considerations

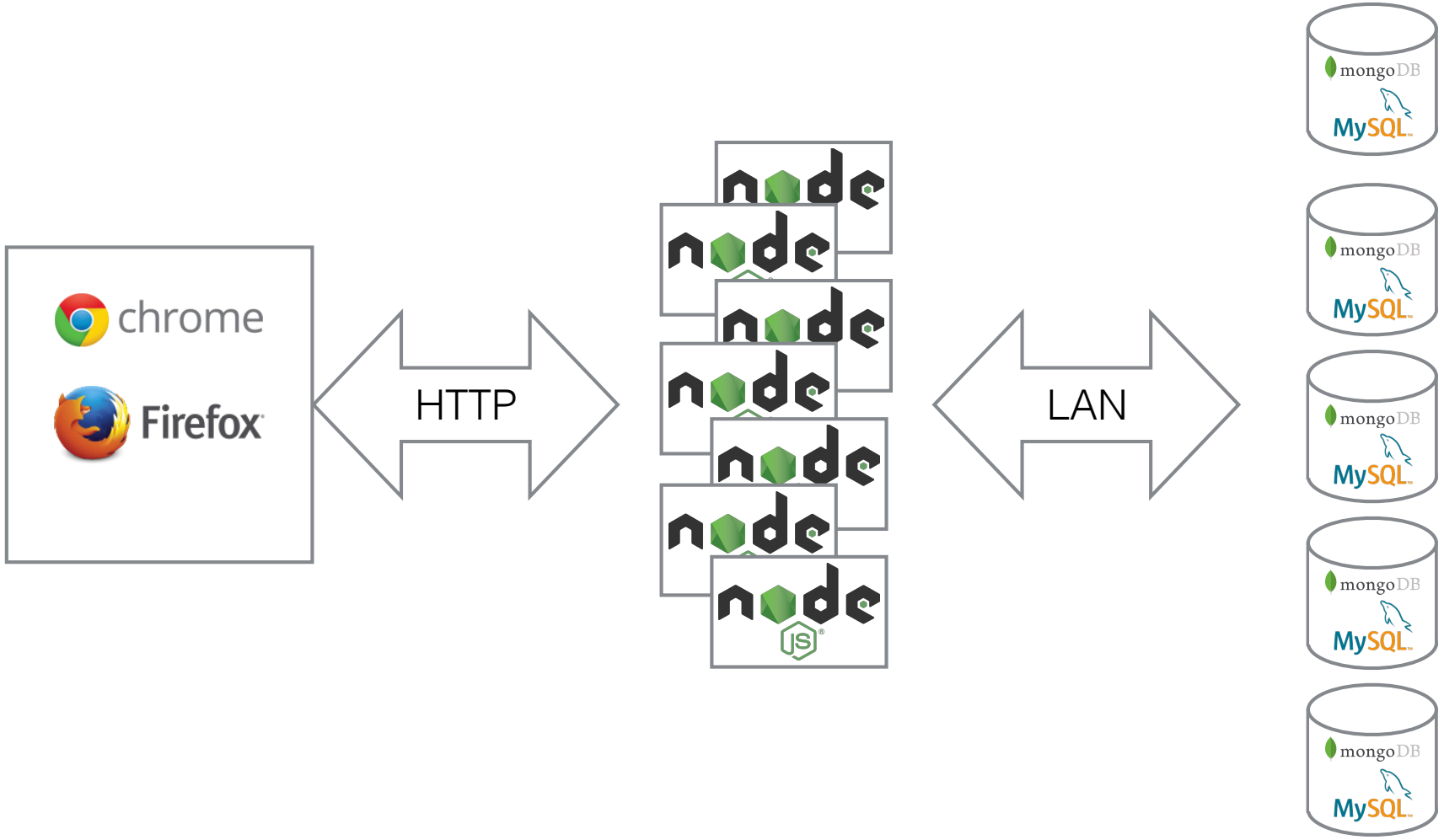
- What user data are you collecting?
- Impact on user performance?
  - E.g. Research question: Would a degradation in performance lead to lower user retention?
    - Artificially reduce introduce latency for some users.
    - Is this ethical?

Large Scale

Standard Web App



Large Scale Web App



Scale-out

- Expand capacity by adding more instances
- Pros:
  - can scale to fill needs by adding and removing instances
  - fault tolerance
- Cons:
  - manage multiple instances and distribute work

Scale Out: Which server to send to?

- Browsers want to speak HTTP to a web server
  - Use load balancing to distribute incoming HTTP requests across many front- end web servers
- HTTP redirection:
  - Front-end machine accepts initial connections
  - Redirects them among an array of back-end machines
- DNS (Domain Name System) load balancing:
  - Specify multiple targets for a given name
  - Handles geographically distributed system
  - DNS servers rotate among those targets
  - How to handle sessions?

Load-balancing switch ("Layer 4-7 Switch")

- Special load balancer network switch
  - Incoming packets pass through load balancer switch between Internet and web servers
  - Load balancer directs TCP connection request to one of the many web servers
  - Load balancer will send all packets for that connection to the same server.
- In some cases the switches are smart enough to inspect session cookies, so that the same session always goes to the same server.
- Stateless servers make load balancing easier (different requests from the same user can be handled by different servers).
- Can select web server based on random or on load estimates

nginx

- Web server designed for scalability
- Load balancer
  - can handle SSL processing
  - application health checks (server fails)
  - session persistence
  - limits to mitigate DOS
  - bandwidth limiting

Scale-out assumptions

- Any server will do
  - Different requests from the same user can be handled by different servers
  - Requires database be shared across servers
- What about session state?
  - should be fast because it is accessed on every request
- Web sockets?
  - cannot load balance each request

Scalability and Sessions

- Pass session state around in cookies if possible
- Cache session state in memory, but store in Database
- Sticky sessions
  - Load balancing - hash on client IP
  - With TLS/SSL offloading can use session cookie

Scale-out storage

- Data sharding - spread database across instances
  - each piece of the database is called a shard
  - Tolerate failures (and improve performance) by replication
  - Increases complexity - Applications must place data across multiple databases
- A Router decides where to send the request(s) to.
- Key questions:
  - how to organize data across shards
    - by key, by tag, geolocation?
  - replication

Memcache

- Main memory caching system
- Key-value store (both keys and values are arbitrary blobs)
- Used to cache results of recent database queries
- Much faster than databases:
  - 500-microsecond access time, vs. 10's of milliseconds
  - Writes still go to database, so no performance improvement
  - Cache misses still hurt performance
  - Must manage consistency in software (flush memcache data when database is modified)

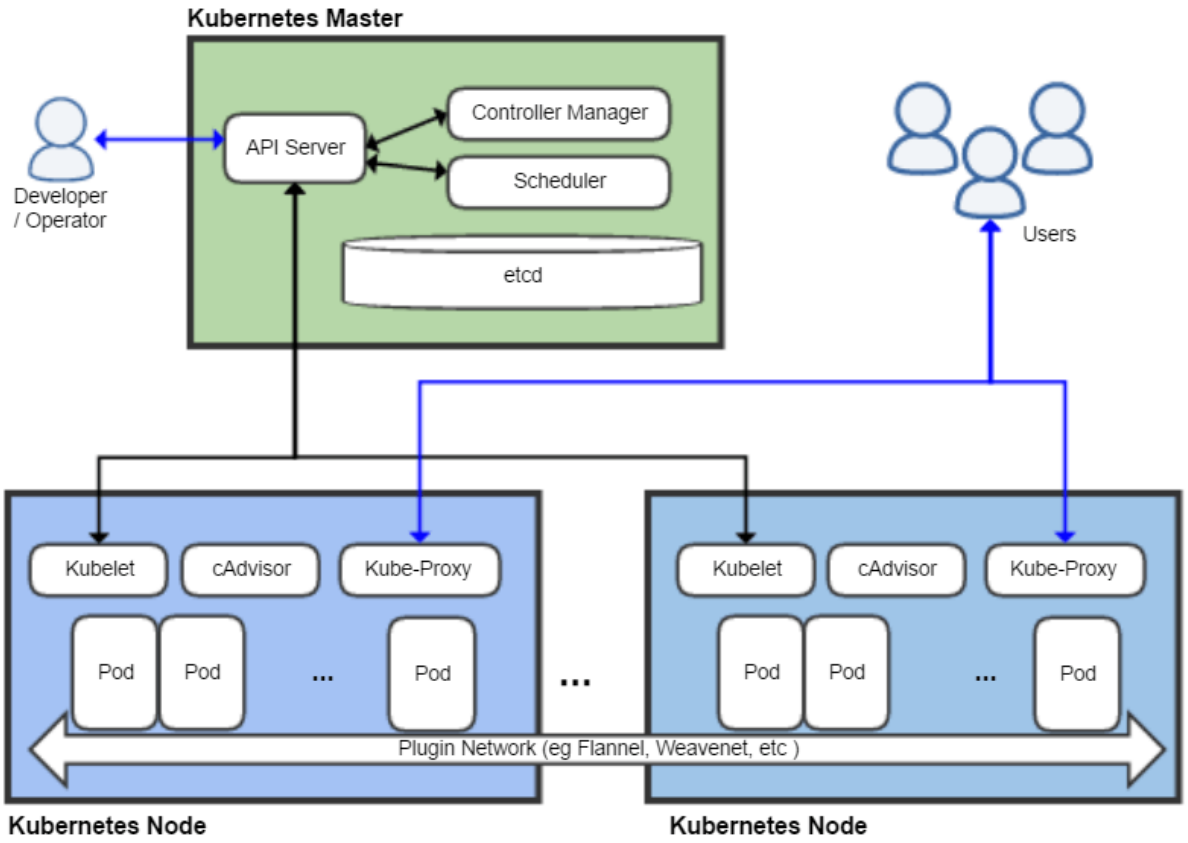
Scalability

- Building this architecture is hard!
- Need data centre expertise
- Figuring out the right number of components is hard

- Cloud computing:
  - allows for dynamic addition and removal of resources
  - outsources data centre management

Kubernetes

"Open source system for automating, deploying and managing containerized applications"



Deployment options

- ngrok
  - public URL for exposing local web server
  - relays traffic through ngrok process running on local make
- heroku
  - public place to deploy web apps
  - supports wide range of technologies
- glitch.com
  - build web apps in the browser
- Firebase
  - back end as a service
  - real time database (web sockets to NoSQL DB)
- Others (?)