WaCC Design: flow.io

Group 24 - Jeroen & Muskan

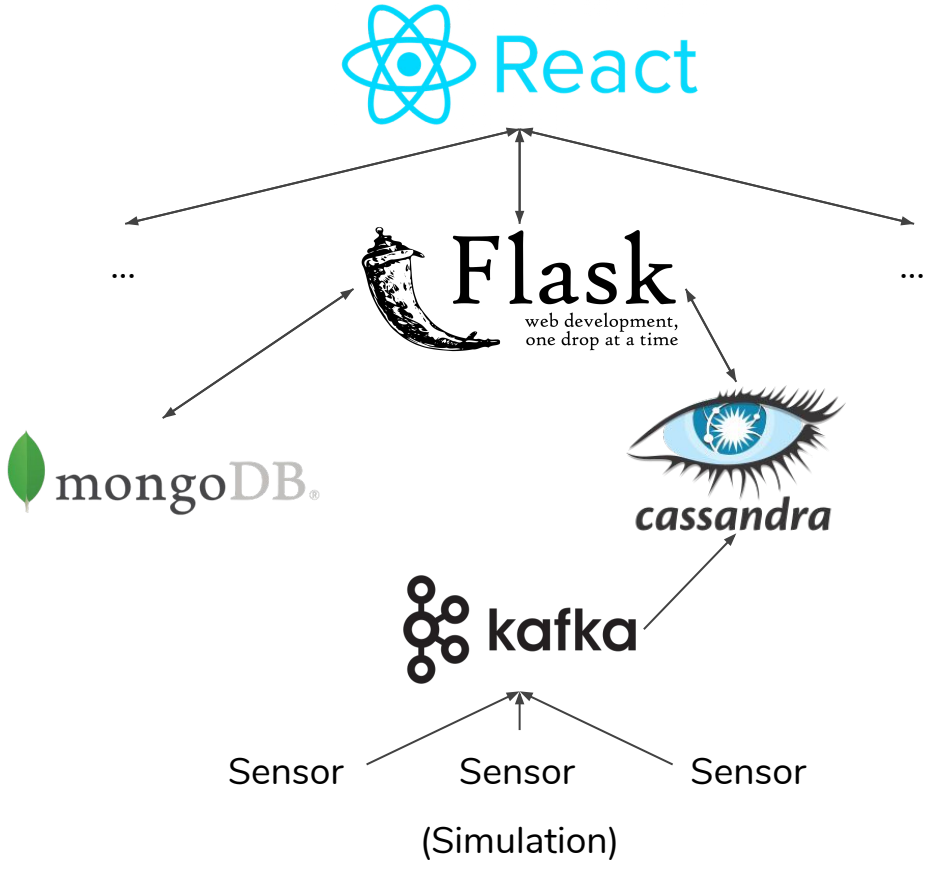
Design Requirements

Data:

- High amount of sensor updates
  - High throughput!
  - Optimizing for update writing.
- Loss of data undesirable
  - Redundancy in update handling.
  - Redundancy in update storage.
- Semi-static data of pipeline elements
  - Meta-data such as date-of-installation, uptime, etc.
  - Different storage needs than sensor data.

User-experience:

- Continuous updates, without page refresh
  - Single page application!
  - Stream updates from backend without user input.
- App should be scalable
  - This means being able to increase frontend/backend capabilities when required.



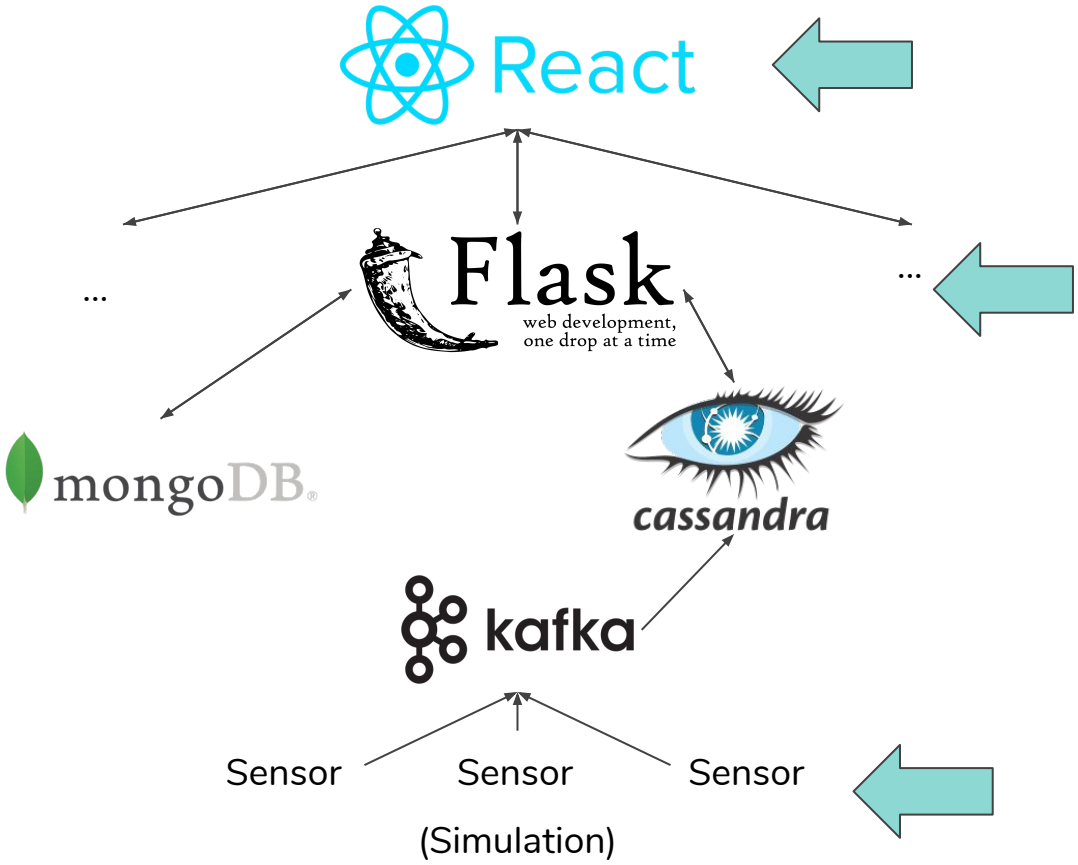
Client side

Server side

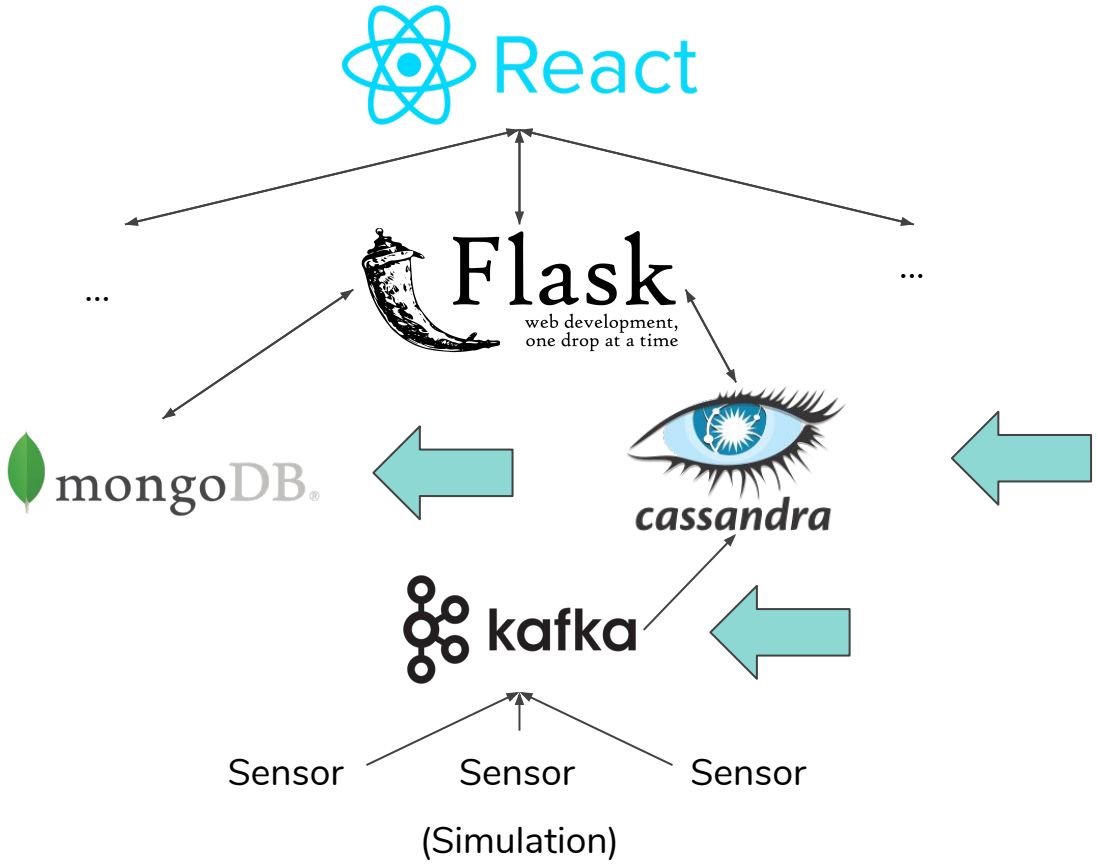
kubernetes

- Can perform load-balancing.
- Used to automatically scale clusters if traffic demands it.
- Adds element of fault-tolerance by maintaining clusters.

Flow.io mockup architecture



- React:
  - Write the UI in JS.
  - Prior experience.
  - Use sockets to stream data from the backend.
- Flask:
  - Python, for ease of use (given the time constraints).
  - Lightweight.
  - Prior experience.
- Simulation:
  - Collection of pumps, pipes, basins and valves.
  - Topology simple, given time constraints.
  - ‘Sensors’ will provide streams of data about the simulation.
  - If time allows it, will add option to tweak state of simulation by e.g. opening/closing valves etc.



- MongoDB:
  - NoSQL document store.
  - For storing (meta-)data about elements in the pipeline.
    - E.g. model type, installation time, limits.
  - Update action more efficient than Cassandra.
    - Updating existing data of pipeline elements far more common than adding new elements.
- Cassandra:
  - Extremely fast writing for sensor updates.
  - Better uptime due to multiple masters.
    - Required to keep ingesting sensor data.
- Kafka:
  - Scalable stream broker.
  - Fault-tolerance through multiple brokers.