Donalds Remain Emperor System DRES

Für die Umsetzung des DRES würde ich folgende Technologien einsetzen:

Da Donald bereits Zugriff zu allen Nachrichten hat müssen diese zu Auswertung weiter geleitet werden.

1. Message Inteceptor Client MIC

   Die Nachrichten werden eingesammelt und mittel des MIC an ein End Point über eine RESTFul Schnittstelle übertragen. Die Übertragung geschieht hier leichtgewichtig über das http Protokol.

   Pro:

   Die Kommunikation über REST ist sehr performant und lässt sich gut mit mehreren Serverinstanzen skalieren. Einfache Realisierung über HTTP. Bei der Übertragung findet eine geringe Systembelastung statt und es ist einfach Wartbar.

   Con: nicht wirklich, es kann zu Redundanzen kommen bei der Übertragung, hier könnte man mit Caching entgegen wirken.

2. End Point Microservice

   Die der End Point ist ein leichtgewichtiger Microservice zb mit Spring Boot realisiert und ausgeführt über Docker-Compose oder Amazon Web Service (AWS). Die Nachrichten werden zb auf einer NoSQL DB zwischengelagert.

   Pro:

   Der Microservice wird quasi als Serverless Architektur umgesetzt und läuft mit einem Embedded Server über Docker oder AWS was eine exzellente Skalierung und Hochverfügbarkeit verspricht. AWS bietet zb das Auto Scalling, dabei reagiert das System mit Vorhersagen auf die Last um möglichst hohe Performance bei möglichst geringen Kosten zu erreichen. NoSQL ist perfekt geeignet um unstrukturierte Daten in einem dynamischen Schema zu halten. Ausserdem ist NoSQL auf extreme Performance ausgelegt und skaliert gut horizontal. Damit können große Datenmengen bewältigt werden. Es entfällt auch die Applikationsserver Wartung.

   Con: höhere Kosten für AWS. Cloud Lösung (wo sind meine Daten?) Abhilfe für beide Probleme, man könnte die Server auf einem eigenen Host mit möglichst ausreichender Bandbreite hosten und über Docker-Compose skalieren. NoSQL unterstötzt nicht wirklich ACID wie das RDBMS Datenbanken tun. Einzelne Nachrichten können somit verloren gehen. Für die Aufgabe ist es verschmerzbar oder man implementiert ein Caching oder ACK wobei das wieder die Performance reduzieren würde.

3. RabbitMQ

   Die Microservices übertragen die gesammelten Nachrichten mittels Message Queues an eine Sammelstelle.

Pro: RabbitMQ ist performant und hat eine sehr gute Horizontale Skalierung mittels Cluster Nodes. Übertragung findet hier mittels Transaktionen statt, somit gehen keine Nachrichten verloren.

Con: kein Batching, muss selbst implementiert werden. Es gibt schnellere Frameworks.

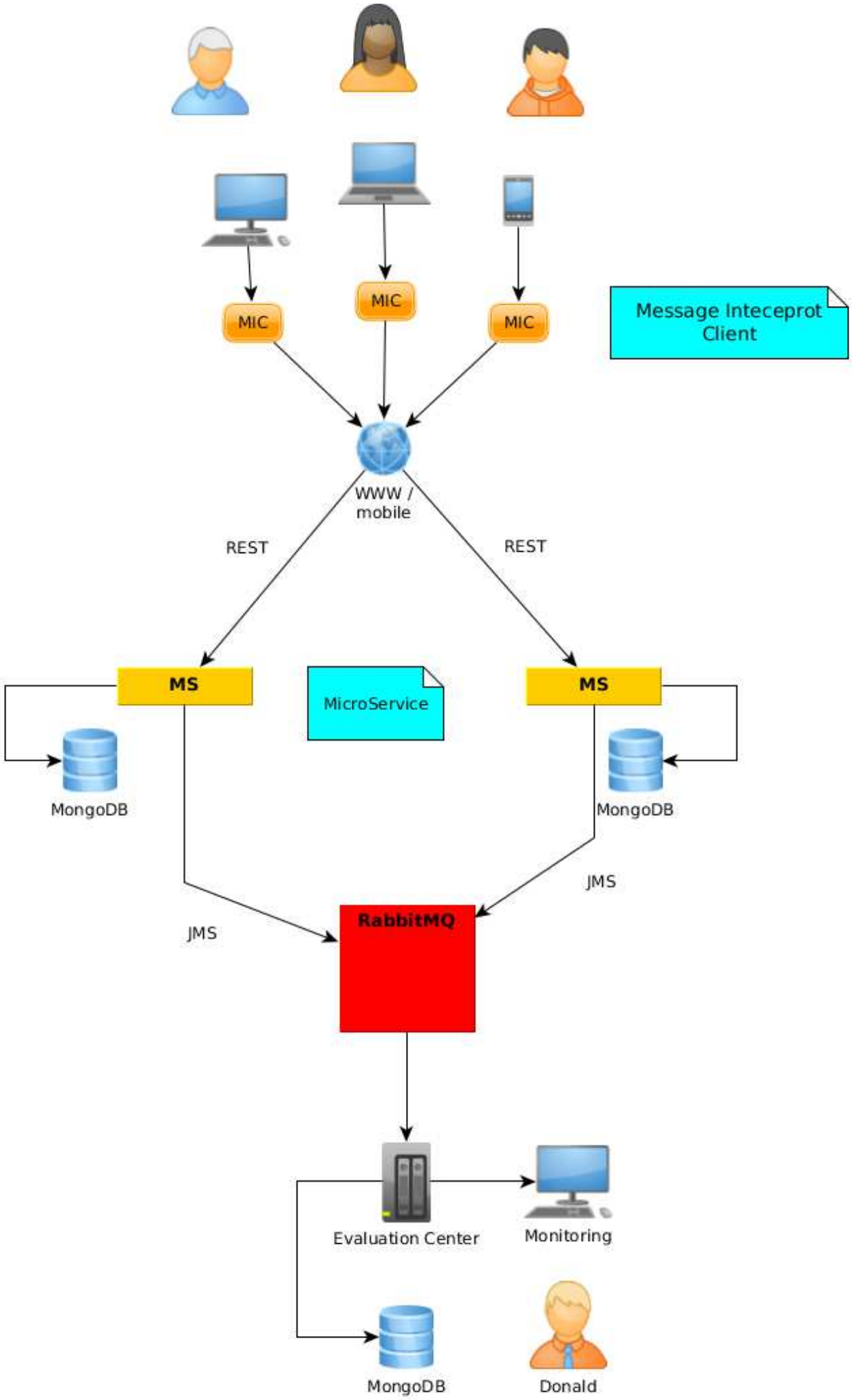
4. Evaluations Center / Monitoring

Das EC wertet am Ende die Nachrichten aus und bietet Schlussfolgerungen und oder Trends. Mittels des Monitoring kann Donald nun seine Entscheidungen treffen.

Für die Lösung wurden verschiedene Architekturen genutzt. Der Klassiker Client/Server ist in jedem Zwischenschritt erkennbar. Für die Datenübertragung wurde auf Kombination aus REST-Architektur und Message Queues gesetzt. Die Microservices sind umgesetzt als Microservice-Architektur und Serverless. Insgesamt handelt es sich bei der Lösung um ein Verteiltes System.

Die Lösung wurde so gewählt weil sie eine gute Performance und Skalierung verspricht, zu dem sind die Aufgaben klar geteilt was die Systemwartbarkeit erleichtert und die Tasks für die Umsetzung möglichst parallelisierbar macht.

Die Kosten für die Umsetzung werden für die Nachrichten Übertragung auf etwa 1 - 2 PM geschätzt. Für das Evaluierungs Center ist es nicht ganz klar. Wird es Manuell ausgewertet, wird eine KI genutzt. Geschätzt in etwa 3 -5 PM. Insgesamt also im best case 4 PM im worst case 7 PM


MIC
MIC
MIC
Message Inteceprot Client
WWW / mobile
REST
REST
MS
MicroService
MS
MongoDB
MongoDB
JMS
JMS
RabbitMQ
Evaluation Center
Monitoring
MongoDB
Donald