Ergebnisbericht SA4E Übung2 (Marvin Mokelke, s4mamoke@uni-trier.de)

Aufgabe 1

Da dem Weihnachtsmann besonders wichtig ist, dass sein neues XmasWishes System zuverlässig und schnell in allen Regionen auf der Welt funktioniert setzt er auf eine verteilte Architektur mit Knotenpunkten auf den verschiedenen Kontinenten. Sein Plan ist es einen seiner Elfen mit dem Design eines einfachen GUI zu beauftragen, mit dem man ein Wunsch einreichen kann. Aufgrund der geringen Anforderungen sollte dies als statische Website realisierbar sein. Durch den Verzicht auf komplexere Technologien wie Server Side Rendering, kann die geplante Website mit geringer Latenz geladen werden. Außerdem kann sie mithilfe eines CDN in allen Ländern der Welt schnell von nächstgelegenem Knotenpunkt geladen werden kann. Dem ähnlich, will er die API auf allen Knotenpunkten instanziieren, um die Latenz weiter zu minimieren. Die Daten will er in einer Datenbank speichern, von der auf mindestens drei Knotenpunkten eine Instanz verfügbar ist. So ist auch bei der Datenspeicherung eine geringe Latenz realisierbar. Zudem synchronisieren sie Datenbankenbanken sich regelmäßig gegenseitig. So können die Nutzer bei Ausfall eines Knotenpunktes einfach umgeleitet werden und die Wünsche sind zu jedem Zeitpunkt mindestens 3-fach gespeichert, was für die Redundanz des Systems spricht. Die geplante Architektur kann auch Abb. 1 entnommen werden.

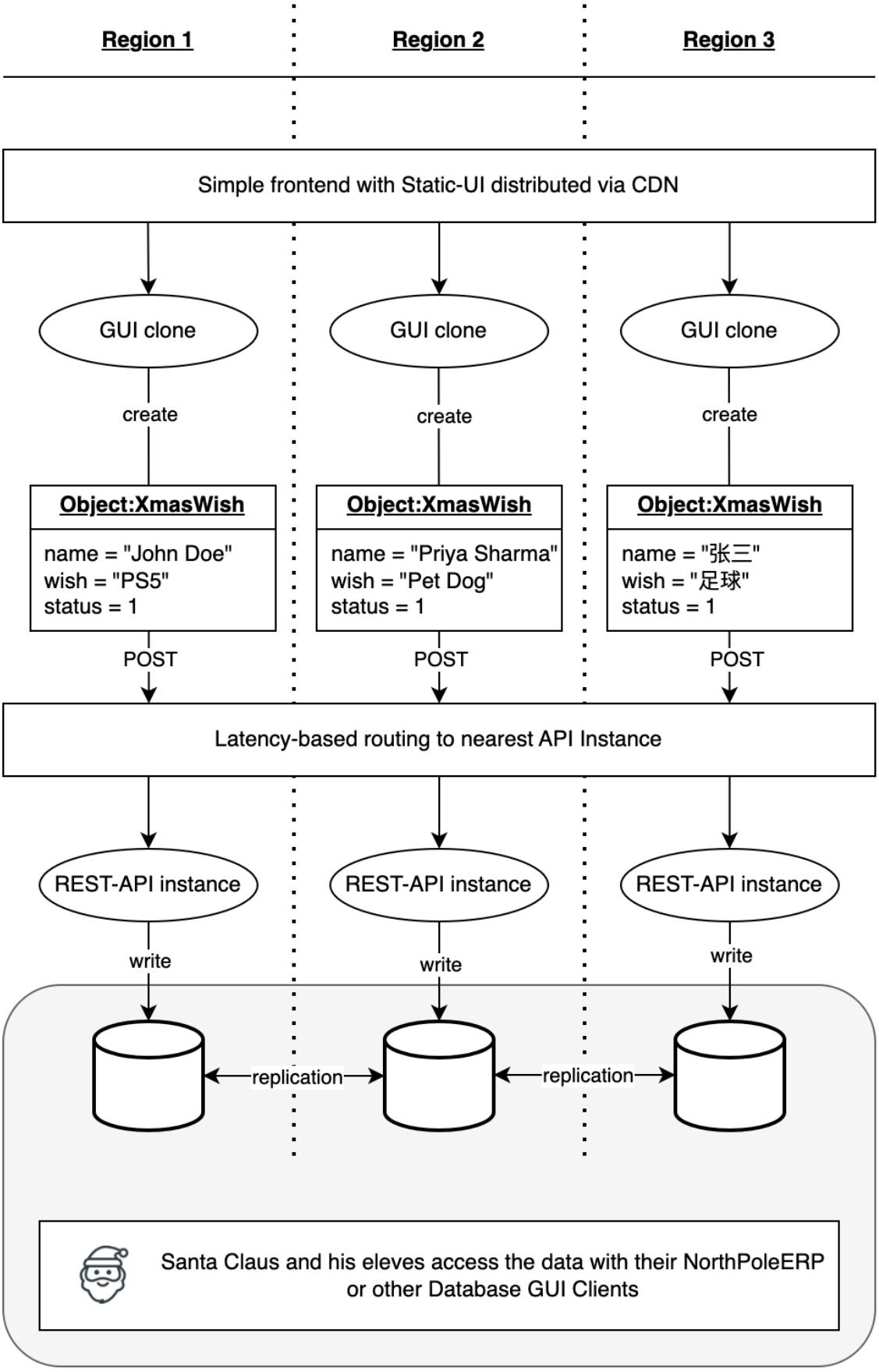

Abbildung 1: Architektur XmasWishes

Aufgabe 2

Einer seiner computeraffinen Elfen, die mit der Implementierung von XmasWishes beauftragt sind, erinnert sich an einen Artikel über Severless Computing und schlägt folglich dem Weihnachtsmann eine solche Umsetzung vor. Dabei führt er folgende Vorteile an:

- Kein weiterer Elf muss für die Serverpflege eingestellt werden
- Die weltweit verteilten Serverstandorte wären eine logistische Herausforderung für die Systemwartung
- Man verhindert das Eingehen vieler neuer Verträge (z.B. Miete für Standort, Stromkosten ...)
- Da man im Frühjahr und Sommer kaum Traffic hat muss benötigt man zu dieser Zeit nur sehr limitierte Rechenkapazitäten. Um die Verfügbarkeit während der Weihnachtssaison zu sichern, müsste man trotzdem sehr teure und leistungsstarke Server anschaffen.
- Große Anbieter verfügen bereits über strategisch gut positionierte Serverstandorte, mit denen man alle Regionen der Welt erschließen und die geplante Architektur schnell umsetzen könnte

Der Weihnachtsmann erkennt die Vorteile und entscheidet sich die Elfen mit der Implementierung zu beauftragen. Da der Weihnachtsmann bereits gute Erfahrungen mit Amazon als Logistik-Partner gesammelt hat, will er XmasWishes ebenfalls in Zusammenarbeit mit AWS realisieren. Weil er eine zu große Abhängigkeit von Amazon und die Monatsrechung Dezember fürchtet, behält er sich jedoch vor nächstes Jahr ein eigenes Servernetzwerk zu aufzubauen.

Der Plan für die Umsetzung mit den AWS-Microservices ist Abbildung 2 zu entnehmen.

In einer ersten Pilotphase soll ein Deployment mit drei Serverlocations (US-Ostküste, Frankfurt und Singapur) getestet werden. Später können dann nach Bedarf weitere hinzugefügt werden.

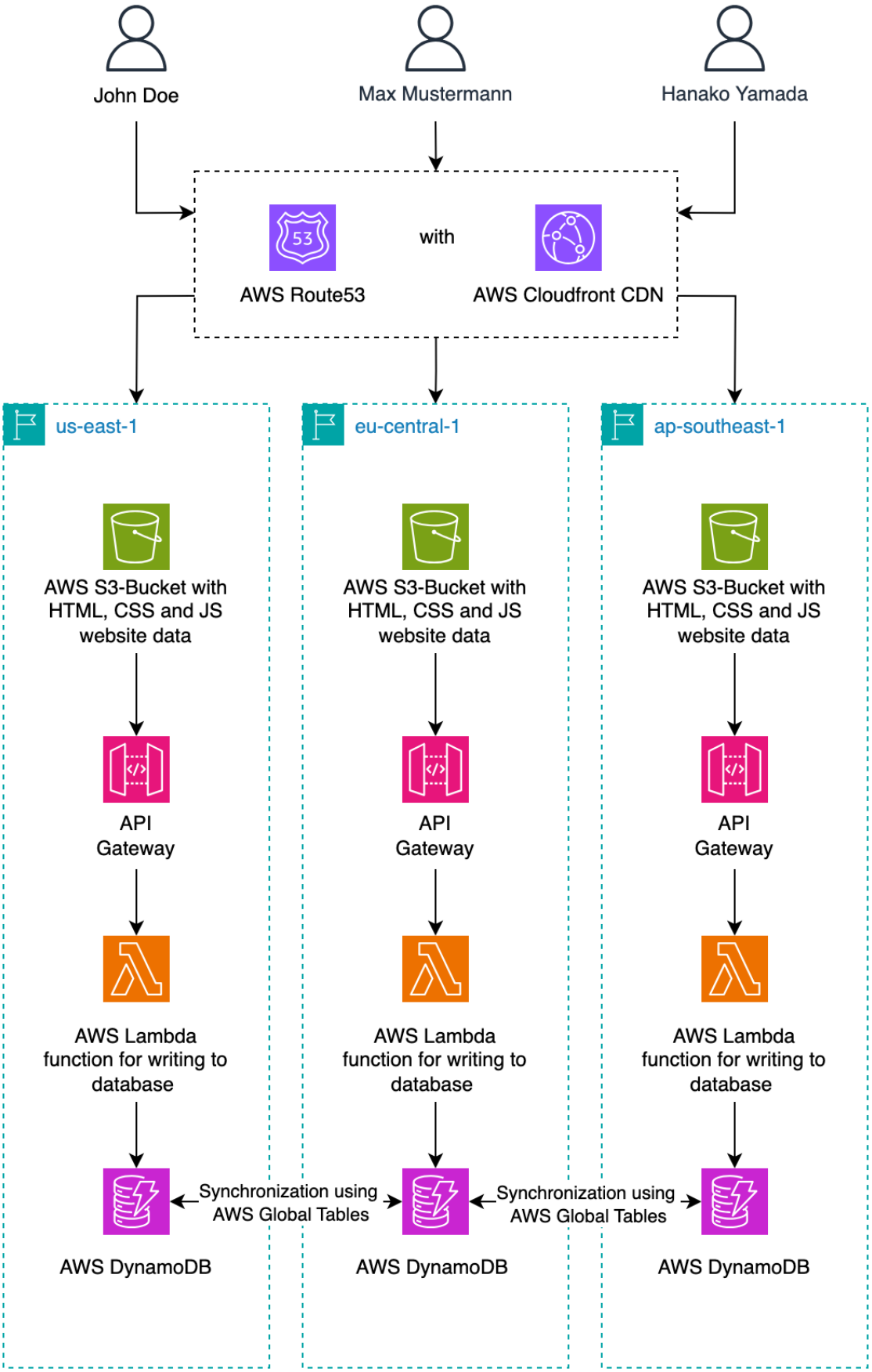

Abbildung 2: Umsetzung von XmasWishes in AWS

Aufgabe 3

Da der inhärente Vorteil einer Serverless Architektur ist, dass sich die Ressourcen dynamisch an den Bedarf anpassen, wäre es witzlos eine Implementierung der oben aufgeführten Architektur auf ihre Performancelimits zu testen. Außerdem wäre die Implementierung für Sie nicht durch einfache Replikation nachzuvollziehen. Als Alternativlösung habe ich daher XMasWishes als Apache Camal App implementiert. Genauer habe ich einen REST API Endpoint definiert, zu dem POST-requests, die ein Objekt vom Typ XmasWish enthalten, geschickt werden können. Diese werden dann mithilfe von H2 in einer lokalen SQL Datenbank abgelegt. Außerdem gibt es eine Website, mithilfe derer die Nutzer ihre Wünsche an Santa Claus übermitteln können. Diese wird ebenfalls über einen GET API Enpoint in der Camal App ausgeliefert.

Zur einfachen Replizierung habe ich alles in einen Docker Container gepackt. Näheres können Sie im README des Repositories lesen.

Zum Stresstest auf meinem PC (Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz mit 16GB RAM) habe ich K6 genutzt. Dort lassen sich eine Anzahl an Virtual Users (VU) definieren, die dann API anfragen generieren. Bei meinen Tests hat sich die Region zwischen 15k-16k VUs als „Sweetspot“ erwiesen, bevor die ersten http-Anfragen fehlschlugen.

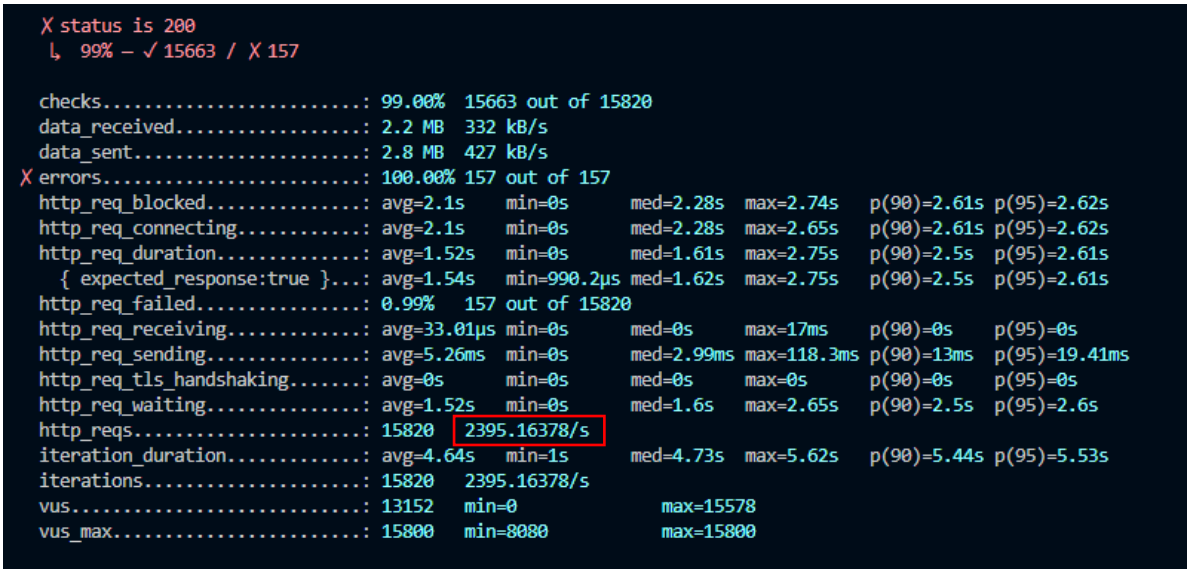

Abbildung 3: Screenshot von API Test mit 15,8k VUs

Auf meinem localhost schafft die API zwischen 2300 und 2500 HTTP-Anfragen pro Sekunde zu beantworten.

Aufgabe 4

In der gleichen Camel-Anwendung habe ich ebenfalls eine Seite erstellt, auf der man eine JSON-Datei oder einen ganzen Ordner hochladen kann. Entsprechen die Dateien

einem bestimmten Schema werden diese an die API gesendet und in der Datenbank abgelegt. So können nicht nur die Dateien aus den Scans in der Datenbank gespeichert werden, sondern auch Nutzer mit ausgiebiger Wunschliste, schnell alle Wünsche an Santa übermitteln.