Software Requirements Specification

for

High Performance Caching of Travel Inventory Data

Version 1.0 approved

Prepared by Malcolm Frank

Kennesaw State University - Senior Capstone

3/13/2018

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Malcolm Frank	3/13/2018	Initial Version	1.0

1. Introduction

1.1 Purpose

We will design and develop a proof of concept application using Amazon Web Services ElastiCache and Redis (IMDB) to demonstrate its ability to scale and meet response time and fault tolerance requirements with appropriate load and failure test scripts. Our proof of concept application will provide an adequate solution architecture in which Travelport can implement to modify/improve their current caching system.

1.2 Document Conventions

This document follows MLA Format. Bold-faced text has been used to emphasize section and sub-section headings. Highlighting is to point out words in the glossary and italicized text is used to label and recognize diagrams.

1.3 Intended Audience and Reading Suggestions

This document is to be read by the development team, the project managers, testers and documentation writers. Our stakeholders, Travelport and all members of the development team may review the document to learn about the project and to understand the requirements. The SRS has been organized approximately in order of increasing specificity. The developers and project manager need to become intimately familiar with the SRS.

Others involved need to review the document as such:

Overall Description - Marketing staff have to become accustomed to the various product features in order to effectively advertise the product.

System Features - Testers need an understanding of the system features to develop meaningful test cases and give useful feedback to the developers.

External Interface Requirements - The software developers need to know the requirements of the device they need to build.

Nonfunctional and Functional Requirements - The software developers.

1.4 Project Scope

The scope of this project is to fasten the customer search experience. This means that the ecommerce services response time will be improved by storing data in memory to reduce time required to access it from disk.

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2. Overall Description

2.1 Product Perspective

High Performance Cache is a distributed caching system for application objects that are shared across multiple servers and require low response time, very high throughput, predictable scaling, continuous availability and information reliability. The product being developed is to serve as a proof of concept application for Travelport Worldwide Ltd. The product uses Amazon Web Services ElastiCache and Redis to host multiple virtual machines that act as a cluster of caching/filter servers which collectively function a single distributed caching system. This product was designed to demonstrate an optimized solution architecture which would eliminate single points of failure Travelport's existing application.

2.2 Product Features

The High Performance Cache contains the following key features:

1. Captures application objects in memory and creates a cluster/replica-set of the data. A cluster/replica-set means that we will have multiple RedisDB instances which mirror all of the data of each other. A cluster/replica-set consists of one Master (also called "Primary") and one or more Slaves (also called "Secondary"). Read-operations can be served by any slave, so you can increase read performance by adding more slaves to the replica-set. Write-operations always take place on the master of the replica-set and then are propagated to the slaves. Therefore writes will not get faster when you add more slaves.
    a. Clusters/Replica-sets also offer fault tolerance. When one of the members of the replica-et goes down the others take over. When the master goes down, the slaves will elect a new master. For productive deployment we will use RedisDB as a replica-set of at least three servers, two of them holding data (the third one is a data-less "arbiter", which is required for determining a new master when one of the slaves goes down).

2. Capture application objects in memory and create a sharded cluster. Meaning that each shard of the cluster/replica-set takes care of a part of the data. Each request, both reads and writes, is served by the cluster where the data resides. This means that both read and write performance can be increased by adding more shards to the cluster. Which document resides on which shard is determined by the *shard key* of each collection. The shard key was chosen in a way in which the data can be evenly distributed on all clusters/replica-sets.
3.

2.3 User Classes and Characteristics

Customer:

Remote customers most frequently will use the system during search processes (request-response) of travel data. For example, user searching for flight information for vacation or a user searching for hotels for a honeymoon.

Developers:

Developers who use Travelport's API and similar services to create travel software.

Travel Agencies:

Travel Agencies who use Travelport's API to host their travel services.

2.4 Operating Environment

The software will operate with the following software components and applications:

The software being developed will be running under Linux operating system. A series of virtual machines will be hosted on Amazon Web Services ElastiCache as RedisDB instances. The hardware will consist of ……..

2.5 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

2.6 User Documentation

A user manual will be developed for this proof of concept application once all required optimization metrics are met.

2.7 Assumptions and Dependencies

It is assumed that the hardware and virtual machines will contain enough processing power to scale to 100 billion requests per month at associated peak usage.

3. System Features

3.1 Cache Airline Travel Data

3.1.1 Description and Priority

The software will be able to add to cache the following data:

- Airline Company Name
- Flight Number
- Departure/Arrival Time
- Ticket Price Range (average lowest price - average highest price)
- Origin/Destination
- Flight Class
- Trip Duration (one-way)

This requirement is high priority.

3.1.2 Stimulus/Response Sequences

User initiates search request for flight information based on any one or combination of attributes listed above in 3.1.1.

System responds with flight data based on search criteria presented in a user friendly graphical interface or application programming interface.

3.1.3 Functional Requirements

- The system will capture the following application objects in a distributed

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1: The system shall distribute the following application objects in cluster/replica-sets:

- Airline Company Name
- Flight Number
- Departure/Arrival Time

- Ticket Price Range (average lowest price - average highest price)
- Origin/Destination
- Flight Class
- Trip Duration (one-way)

REQ-2: The system will read data from distributed cluster/replica-sets.
REQ-3: The system will write data to the distributed cluster/replica-sets.
REQ-4: The system will store and replicate data across cluster/replica-sets in memory as master - slave nodes.

3.2 System Feature 2 (and so on)

4. External Interface Requirements

4.1 User Interfaces

The High Performance Cache is a middleware caching server which will only be seen by developers, testers, Kennesaw State University's Department of Computing and Software Engineering, and Travelport during implementation, testing, or demonstration.

The system is designed run run behind the scenes, in between the end users API/GUI and the server that houses the sought after data.

4.2 Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

4.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

4.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The system shall be able to scale at least 100 billion requests on a monthly basis.
- The system shall have a request-response time of no more than 3 seconds.
- The system will be implemented using Redis In-memory Database and Amazon Web Services ElastiCache.
- The system will need at least 6 RedisDB instances (virtual machines/servers). Each replica-set will consist of two servers.
- The system will need a Redis router (server) and a config server for the cluster/replica-set. Arbiters are very lightweight and are only needed when a cluster/replica-set member goes down, so they will share the same hardware the RedisDB instances.

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and

usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: Issues List

< This is a dynamic list of the open requirements issues that remain to be resolved, including TBDs, pending decisions, information that is needed, conflicts awaiting resolution, and the like.>