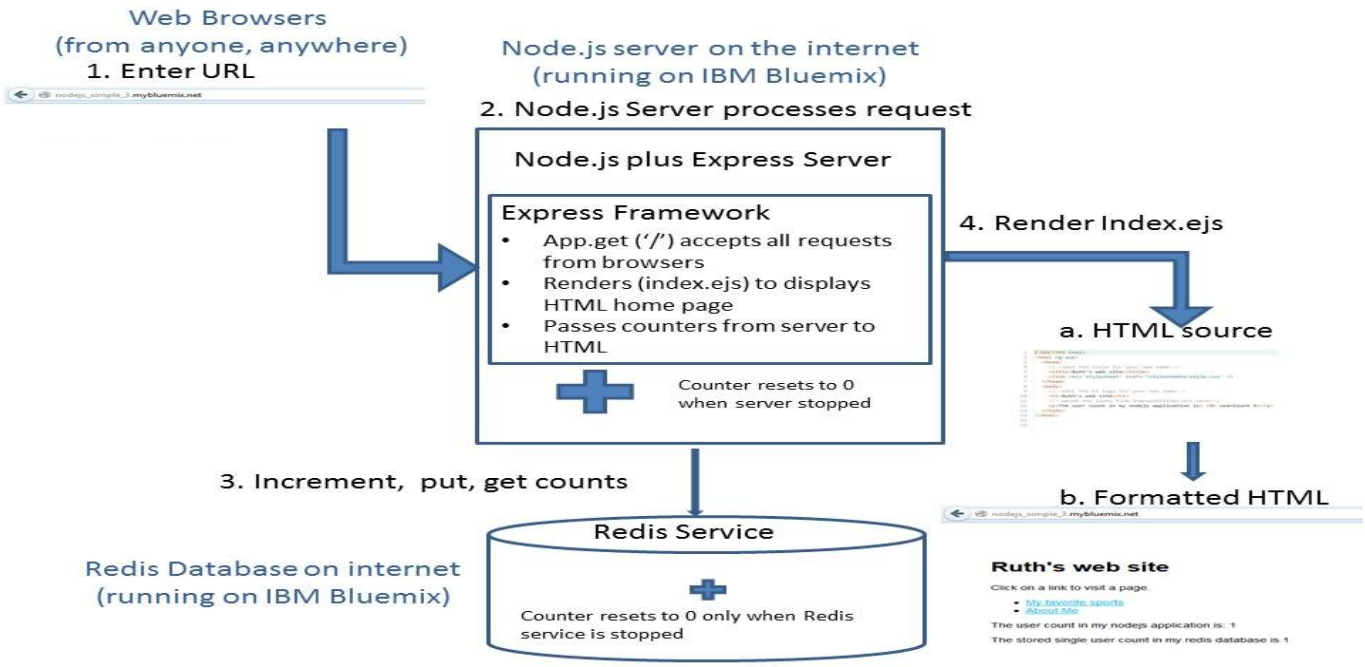
Node.js Server – Concept 3

In this project, you will again start a Node.js Server with the Express framework on IBM Bluemix on the internet, and make some changes. This time, the project also uses a Bluemix database service (Redis) to persist the counter data when your Node.js application is stopped.

1. Web browsers can make requests to your server at the Bluemix URL your server is brought up on.
2. The server accepts the requests, and increments an internal counter. A second counter is also incremented and maintained in a Redis Database Service accessed through Bluemix. The code within your Node.js application formats HTML with the value of the internal counter, and the database counter.
3. In the web browser, the web site and counters are displayed.



4. You will stop and restart the server. When you stop the server, the counter in memory goes away, but the Redis Service continues to run and that counter is not reset. Therefore, when you restart the server, the in-memory counter is reset, but the Redis counter does not start over.

5. You will also completely delete your application and the associated Redis Service. When you remove the Redis Service, the counter in Redis is deleted. Therefore, when you redeploy the application, a new instance of the Redis Service is created, and the count is reset back to the beginning.

6. You will also make some changes to the code. You make changes in the DevOps Services Integrated Development Environment, and deploy the new version of the application back to Bluemix. The code you write is javascript, and the example walks you through adding another counter to the Redis database, and formatting it on the HTML page.

The calls to Redis are asynchronous, meaning that the call to increment the count might not happen before the call to retrieve the count. The application needs the be sure the increment is done before the value of the counter is retrieved. In other words, the calls need to be synchronized. In order to achieve this, the code is written as a single statement (“pyramid of doom”) to ensure the ordering of the increment before the retrieval of the value.