Chatbot on Serverless/Lamba Architecture

Nandan.A
Student, Reva University

Prof.Shilpa Choudary
Professor, Reva University

***Abstract*—The OpenLambda, a new, opensource platform for building next-generation web services and applications on serverless computation. The key aspects of serverless computation and present numerous research challenges that must be, addressed in the design and implementation of such systems. The study of current web applications, so as to better motivate some aspects of serverless application construction. Chatbots platform are used by consumers worldwide for integrating it with backend services . It is still difficult to build and deploy chatbots developers need to handle the coordination of the backend services to build the chatbot interface, integrate the chatbot with external services, and worry about extensibility, scalability, and maintenance. The serverless architecture could be ideal platform to build the chatbot.**

Keywords: Lambda Architecture, Chat-bot, Multitier Architecture, Microservices.

I. INTRODUCTION

The multi-tier application has been a well-accepted architecture pattern for decades. The multi-tier pattern provides guidelines to follow to ensure decoupling of components and scalable components that can be manged separately Multi-tiered applications are often built using a service-oriented architecture (SOA) approach to using web services. The SOA appoach utilizes the network as the boundary between tiers. However, there are many not different aspects of creating a new web service tier as part of your application. Most of the code written within a multi-tier web application is a based on the pattern itself. Examples like code that integrates one tier to another, code that defines an API and a data model that the tiers use to understand each other, and security-related code that ensures that the tiers' integration points are not exposed.

Three-tier Architecture Overview

The three-tier architecture is a popular pattern for user-facing web applications. The tiers in architecture include the presentation tier, the logic tier, and the data tier. The presentation tier represents the component that users directly interact with (such as a web page, mobile app UI, etc.). The logic tier contains the code required to translate user actions at the presentation tier to the functionality

that drives the application's behavior. The data tier consists of storage media (databases, object stores, caches, file systems, etc.) that hold the data relevant to the application. Figure 1 shows an example of a simple three-tier application.

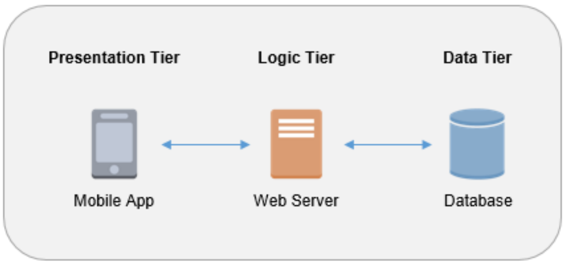

Figure 1: Architectural pattern for a simple three-tier application

In Serverless Multi-Tier Architectures a backend remains private and secure. The benefits of this powerful pattern across each tier of a multi-tiered architecture. Example of a multitiered architecture is a three-tier web application. However, you can apply this multi-tier pattern well beyond a typical three-tier web application.

The Serverless Logic Tier

The logic tier of the three-tier architecture represents the brains of the application. This is why integrating Lambda architecture and API gateway to form logic tier has its advantages. This layer allow build a serverless production application that is highly available, scalable, and secure. The application could use thousands of servers, however by leveraging this pattern management of server is not with programmer. In addition there are following benefits:

No operating systems to choose, secure, patch, or manage.

No servers to right size, monitor, or scale out.

No risk to your cost by over-provisioning.

No risk to your performance by under-provisioning.

In addition, there are specific features within each service that benefit the multi-tier architecture pattern.

Serverless is emerging as an alternative way of building backend applications. Serverless does not require dedicated infrastructure. Cloud vendors such as AWS- Amazon Web Services, Google, Microsoft, and IBM have created their versions of serverless platform. Serverless lets the developer deploy "functions". Serverless is also referred to as Functions-as-a-Service into a shared platform that is maintained by the vendor.

The functions are typically standard code snippets in popular languages that execute in a stateless fashion. It is the vendor's responsibility to keep the infrastructure running smoothly and scale resources to satisfy function executions due to changes in user demand.

Serverless life-cycle costs are typically lower than costs for dedicated infrastructure as serverless vendors do not charge for idle time. In the serverless programming model, developers deploy code snippets as "functions" into the cloud. Functions are stateless and each invocation is independent of previous runs. Functions can be invoked directly or triggered by events. Functions are not meant to be long running; hence, this encourages a software model where an application is broken up into several functions containing a small amount of logic. Non-trivial applications like .chatbots will utilize many functions chained together.

Serverless functions can be used coordinate the micro-services used by the chatbot. While serverless functions are not specifically designed for mashups, the programming model provides data flow and event-based abstractions usable for mashup architectures. In addtion, serverless hides most operational concerns from the developer and provides a scalable rutime for creating chatbot solutions.

2. Lambda background

The one specific implementation of a Lambda environment, we can consider the AWS Lambda cloud platform. The AWS Lambda programming model and some of its advantages over server-based models.

2.1 Programming Model

The Lambda model allows developers to specify functions that run in response to various events. The focus is on the use where the event is an RPC call from a web application and the uploaded function is an RPC handler. A developer selects a runtime environment uploads the relevant code, and specifies the name of the function that should handle events. The developer can then associate the Lambda with a URL. Client-side code can then issue RPC calls by issuing requests to the URL .Handlers can execute on any worker; in AWS, start up time for a new worker is approximately 1-2 seconds. Upon a load burst, a load balancer can start a Lambda handler on a new worker to service a queued RPC call without incurring excessive latencies. However, calls to a particular Lambda are typically sent to the same worker(s) to avoid sandbox reinitialization. The developer can bound the resources that may be utilized by a handler. In AWS, the cost of an invocation is proportional to the memory cap (not the actual memory consumed) multiplied by the actual execution time, as rounded up to the nearest 100ms.

Lambda functions are essentially stateless; if the same handler is invoked on the same worker, common state may be visible between invocations, but no guarantees are provided. Thus, Lambda applications are often used alongside a cloud database.

2.2 Lambda Advantages

A primary advantage of the Lambda model is its ability to quickly and automatically scale the number of workers when load suddenly increases. To demonstrate this, we compare AWS Lambda to a container-based server platform, AWS Elastic Beanstalk. On both platforms we run the same benchmark for one minute. Typical AWS Lambda starts 100 unique worker instances within 1.6s to serve the requests, all Elastic BS requests were served by the same instance; as a result, each request in Elastic BS had to wait behind 99 other 200ms requests. AWS Lambda also has the advantage of not requiring configuration for scaling. In contrast, Elastic BS configuration is complex, involving 20 different settings for scaling alone. Even though we tuned Elastic BS to scale as fast as possible (disregarding monetary cost), it still failed to spin up new workers for several minutes.

3. Chatbot

The basic architecture and prototype implementation of a chatbot. Serverless computing is an ideal platform for building extensible chatbots due to its differentiating characteristics. It frees the developer from the burden of managing the scalability due to fluctuation in user demand. It allows for a normalized scripting environment that can be packaged and distributed. Finally, it also provides a critical missing piece: a means for composable services to interact with other, non conversational services to perform interesting tasks for users.

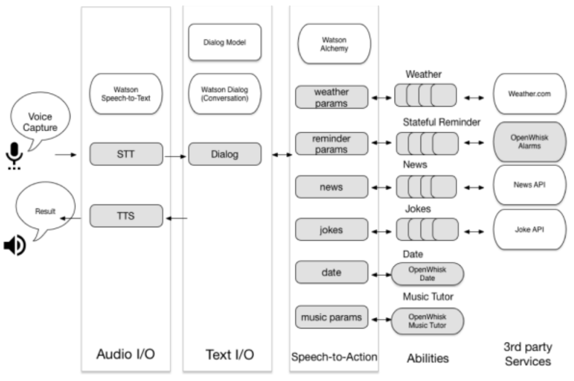

Figure2 General Chatbot Architecture

3.1 Architecture

Figure 2 shows the architecture of serverless chatbot. The basic architecture consists of four levels of serverless actions and a microservice endpoint.

The first level of processing is Audio I/O, which converts user audio input to text, and vice versa. This part of architecture is optional and only used if text is not available for input.

The second level of processing is Text I/O. This level is responsible for routing the user input to the correct set of serverless actions for domain-specific processing. In this stage, one can utilize any publically available conversation services to help route the request, as well as “box” in the user to provide feedback if the user input cannot be processed.

The third level of processing is for domain specific chatbot “abilities.” Each ability has to do at least two things. First, it must convert the raw user input into a parameterized function call. Secondly, it takes the parameters and invokes one or more actions to handle the input. Handler actions may call out to external services to complete their processing. Depending on the ability, chatbot returned results as text and/or audio output.

To deploy the chatbot, the developer needs OpenLamda Platfom. OpenLambda allows the binding of parameter values at deploy time, so the developer only has to set the value of authentication tokens once. Since our architecture consists only of serverless actions, it is possible to interact with our chatbot through any of the three layers :

1) audio for anyspoken interaction - optional;

2) text for textual input;

3) direct invocation of an ability using the serverless framework API.

4. Conclusion

The existing implementations of serverless architecure except OpenWhisk are closed and proprietary. In order to facilitate research on Lambda architectures, the OpenLambda, provides a base upon which researchers can evaluate new approaches to serverless computing and realizing the advantages OpenLambda platform. Hence we can make use of this platform to build a chat-bot. By combines the benefit of serverless technology with the flexibility of the chat based platform to provide a simple solution for weather data retrieval for the users.

REFERENCES

[1] Scott Hendrickson and Stephen Sturdevant and Tyler Harter and Venkateshwaran Venkataramani and Andrea C. ArpaciDusseau and Remzi H. Arpaci-Dusseau, Serverless Computation with OpenLambda, 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), 2016.

[2] Yan, Mengting and Castro, Paul and Cheng, Perry and Ishakian, Vatche, Building a Chatbot with Serverless Computing, Proceedings of the 1st International Workshop on Mashups of Things and APIs, MOTA '16..

[3] Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht, Occupy theCloud: Distributed Computing for the 99%, arXiv:1702.04024

[4] AWS Serverless MultiTier Architectures, WhitePaper,

[5] Yudi Andrean Phanama1, Fransiskus. Astha Ekadiyanto2 and Riri Fitri Sari3, Serverless Mobile Multiuser Chat Application Based on ChronoSync for Named Data Networking, TRANSACTIONS ON NETWORKS AND COMMUNICATIONS,V OLUME 4, I SSUE 4 ISSN: 2054 7420