I'm in the Mood for Org

Nandaja Varma Nandakumar

refs

Abstract

Serverless Computing is an up and coming platform as a service offering where the cloud provider manages and allocates resources needed to keep the application running. This lets the developer focus on the application development and not on server maintenance. Alongside off loading the provisioning and maintenance of the server, Serverless computing also reduces resource waste by scaling up and down the allocation depending on the load and the configurations. The users only pay for the resources that were used by the application thereby saving huge operational cost on their infrastructure hosting.

Although Serverless might sounds like the holy grail of application hosting, the current state of art technology fall short in several places to meet the industrial requirements. Data intensive applications, streaming applications, and distributed computing are some of the fields that could be benefited heavily by implementation on Serverless platforms in terms of ease of development, efficiency and cost. But all the existing platforms offer very poor performance in these fields and works mostly via workarounds and n number of third party tools.

This thesis analyses the Serverless paradigm in depth, pointing out the reasons for this reduced adaptability. To solve these issues, we propose a lightweight extension to an existing Open Source Serverless platform, OpenFaaS, by provide flexibility, scalability and adaptability, while making sure not to violate the notion of functions. Our implementation tries to reduce the operational gap between the industrial applications and theoretical ideas produced by researches in the academia in the past few years. This thesis also offers a deep study of the full potential and limitations of Serverless thereby making it clear to the reader why more innovations are necessary in this field.

Contents

1 Introduction

Serverless can easily be considered as the new generation of platform as a service. It can be thought of as an infrastructure where the programmer send their application as functions in a predefined format, in a supported programming language as documented by the provider. This function get hosted at a certain endpoint which can be triggered with certain events supported by the

platform. In short, instead of having continuously running servers, functions operate as event handlers and when the functions execute, the equivalent CPU usage is paid for by the user. This has huge economical and architectural implications that is still waiting to be explored in its full power. While the developers worry about the logic of handling the requests/events, the infrastructure provider takes care of receiving the request, responding to them, capacity planning, task scheduling, and operational monitoring[**gotoconf**].

In the current industrial applications, data intensiveness of the applications are increasing day by day paving way to adopt several resource heavy tools to do stream processing, distributed processing etc. More than often CPU and memory loads in these machines tent to vary a lot and rather than having a dedicated server to accommodate the whole range of requirements, it makes perfect sense to convert into a Serverless workload thereby saving up on operational cost, resource waste, and ease of development. Having said that, the current commercial offerings of Serverless do not work very great with such workloads.

This is mostly due to the sheer nature of the serverless paradigm of being completely stateless, thereby forcing the developers to use external block storages for data store and communication. In this thesis, we try to extend serverless to leverage its full potential by introducing an efficient form of state thereby providing flexibility, scalability and adaptibility at the same time not violating the notion of functions in these platforms. We will be extending an Open Source serverless platform called OpenFaaS considering its simplistic and expandable architecture.

Currently most of the commercial serverless offering are closed source and vendor locked in to their respective platforms by cloud providers. But in the past couple of years the field has gotten a lot of traction in the academia and a lot of Open Source alternatives are being widely adopted. This being the case, a lot of these works hasn't been properly applied in the industry, some because of the absense of proper integrations with the industry standard tools, and some because of the operational gap between the theoretical ideas and the practicality or usability in the field. This thesis tried to reduce that gap by proposing a very secure and multi-tenant implementation of a state-ful Serverless setup which can be easily used for production quality applications. A focus on the possibility to monitor the application performance and usage provides a possibility to do fine grained billing of the resources and thereby

contributing to the easy adaptability of the extension.

Using our proposed Serverless setup, we try to efficiently run a Extract-Transfer-Load(ETL) workload on streaming data. ETL basically is a pipeline that involves receiving data from source, cleaning and transforming it, and loading it to a sink. We will split the whole operation into multiple functions as per the Serverless notion and have them communicate data and state internally to complete the pipeline thereby reducing the latency and external bottlenecks.

This document describes more on Serverless paradigm, the shortcomings of it, the ones we are trying to solve, our solution and evaluation. It is split into several sections as follows:

In Section 2, we go a bit in depth to understand the history of cloud infrastructure and the technological innovations that led to Serverless paradigm. We also look in detail at the characteristics and nature of Serverless. We look at some commercial Serverless offerings and understand how in the programming world Serverless has influenced even in the way of developing. We will also see what limitations it holds at its current state of evolution and on solving which issue are we particularly interested in, in the scope of this thesis.

In Section 4, we look at the current state of research in the field of Serverless technologies and some related works.

In Section 3, We present the proposed solution for our Serverless setup going into detail about how certain unacceptable limitations can be overcome.

In Section 4, the implementation of the system including the architecture and the tools used is presented.

In Section 5, we go on with the evaluation of our system as opposed to standard Serverless workloads.

We move on to Section 6 to understand the limitations of our proposed system.

In Section 7, the future work that can be done in this direction is laid out before the reader.

2 Background and Motivation

The term serverless have been vaguely thrown around the domain of cloud infrastructure in the past decade as the breakthrough resource(and hence money) saving tool that lets the developers focus on application logic rather than the deployment and server maintenance. Having said that, it is often hard to define what exactly serverless is since the service offering tend to change based on the cloud provider and the interpretations of the users. It is fair to say that serverless is a huge leap in the direction of using computational power as a resource which can be paid for as per the usage. Although the terminology is irrelevant, we will be focusing on the serverless offering called Function-as-a-Service(FaaS) where the cloud providers offer a platform to which we can upload our application code to(complying to the API rules) and get uninterrupted service of the same at an endpoint irrespective of the traffic or data load. Paying only for what resources has been used adds to the attraction of the domain. In this section, we will understand more about this technology, the popular commercial offerings the same, and its limitations and the current state of research. We will also analyze the popular data processing and streaming pipelines in the industries these days and why serverless computing fall short in being the right tool of development and deployment here.

2.1 Serverless Computing

2.1.1 Evolution of cloud resource management

In the past 3 decades, software deployments and infrastructure management has seen a lot of innovation and evolution. Before diving into the current industrial standards, it is important to understand the evolutions in this field to get a better grasp on the technological innovations that bought this about.

Dedicated servers Even almost 10 years ago this was the norm in most enterprises. Dedicated servers are physical machines. The general practice was to have server racks on the premise of the company which are maintained by system administrators and all you software hosted there. Although this method offers advanced security and high availability, it is often common that a lot of physical resources were underutilized and each resource was

for single client. Not to mention the horrific environmental impact of the reserved heavy hardware which are not completely utilized.

Dedicated virtual machines(BaaS) Virtualization technology changed the face of software infrastructure by decoupling applications from the underlying hardware. Virtualized servers are not physical machines, they are a software construct. Virtual servers run on dedicated servers, the resources of which are divided between several virtual servers.

Virtualization usually involves installing a virtualization software(Hypervisor) on an existing operating system and then having multiple operating systems on it, sharing all the resources of the underlying operating system, yet providing great security and isolation.

Despite the IO overhead of the virtualization, this technology reduced the resource wastage to a very great extend. The enterprices could share their hardware into multiple virtual machines and have different hosting and computation in each of the them.

The advent of the cloud hosting happened around this period. Companies started offering hosting spaces or virtual private servers(VPS) that would give you the feeling of having a real system although it is a virtualized system which is sharing the resources with other VPSs. This reduced a lot the amount of work and energy spent on maintaining server racks along with the terrible underutilization of resources.

This kind of offering, generally called as Infrastructure as a service, went through a series of changes during the past decade. Starting from a fixed pay per month plan to pay per the hours the systems are up, the domain offers quite a lot of flexiblility.

With the advent of virtualization, the job of system administrators became unnecessary in most companies and gave birth to job profiles called DevOps(development and operations) which are application developers focusing on the provisioning of the VPSs and deploying the software to these servers.

Although IaaS solved the hassle around infrastructure provisioning to a great extend, virtual systems and application load still remained independent. Applications still gets dedicated virtual machines even if the load/traffic to and fro the application is not constant all the time. This meant that a lot of resources are actually being wasted.

Containers: A gamechanger in the world of virtualization has been containers. Even though Linux containers has existed for a very long time, in the past decade, containers were made a lot more approachable as a technology. Containers can be thought of as a very light weight virtual machines that exist in the user space as isolated environments with its own resource views and limits. Containers share the kernel with the host operating system as opposed to the virtual machines. This makes the containers extremely lightweight making it the ideal candidate for running applications. What makes container based deployments special as opposed to the ones deployed directly on the host is the consistency of the environment. The application execution environment can be recreated and ported from one system to another without affecting the functionality of the function or having to reinstall the whole binary dependencies on the new machine. Reproducability of the production environment even in the local exactly, meant that the development/testing cycle became much more efficient.

Autoscaling The ease of the way in which one can limit the resources and tweak these parameters conviniently contributed majorly to the world of autoscaling which basically meant resources to a virtual system was added or removed as per requirement. This allowed the cloud providers to sell services that could add or remove resources when needed and charge the user accordingly.

<figure comparing virtualization/containorization and such>

Serverless The pioneers of this technology can be considered as the proprietory service Lambda by Amazon Web Services[CITE]. Several other cloud providers followed suit with similar platforms specific to their infrastructure. Before diving into the technicalities, let us understand what the industrial definitions are of the term Serverless. Like mentioned earlier, in the past two years the terms Serverless and Function-as-a-Service are quite often used

interchangeably. In terms of the resource reservation, serverless can be considered as a platform as a service solution that scales. Your application will always have enough and only enough resources dedicated to it. It will scale up and down based on the load and traffic and the developer only pays for the usage. This paradigm of autoscaling has been hence applied even to database storage solutions by major cloud providers such that even the block storage is allocated based on usage and there will be a burst of reservation as soon as a certain limit is reached. The nature of serverless makes it attractive for both developers and the cloud providers since in the case of former, it means paying much less and in case of the latter, it means they can easily provide shared tenant resource allocation units

2.1.2 Properties of Serverless

Statelessness

- the functions execute, they just take in data, process and output.

Triggers

Parallelism

Developer friendliness

Dependency management

Debugging and testing

Deployment

Logging and monitoring

Billing

2.1.3 How programming models are getting affected by this

Faas + microservices

Statelessness

2.1.4 Popular commercial offerings

AWS Lambda

Google cloud functions

Azure functions

2.1.5 Where serverless computing fall short

Although serverless computing might sound like the silver bullet of the deployment solutions, it is a field that is still being rapidly grown and researched on. There are several staggering shortcomings for this technology that makes it unsuitable for certain applications. The current offering have the following noticeable limitations.

Lack of state The serverless/auto-scaling paradigm generally push for a development style involving no state to make the infrastructure simple encouraging a functional style of development. Although this can contribute to easily scalable and parallelisable applications, it often limits the technology from being adapted in applications that are data intensive and/or requires faster response times. The fact that serverless functions don't store any intermediate state requires the application developers to use a block storage to store the data and state after the execution. This basically means communication via slow storage and adds a lot to the latency. This discourages the use of serverless in distributed computing which is actually a domain that needs very fine grained communication between the functions and usually a lot of resources are wastefully dedicated to ensure high availability.

Coordination issues among functions

1. ETL
2. Distributed Computing

Latency IO data shipping vs function shipping

Vendor lock in It is no secret that the most widely used FaaS/serverless offerings are the ones by proprietary cloud providers where they hand twist the developers into complying to their programming environment and runtime thereby forcing devs to use their technologies. What such practices contribute to is limited innovations and development around the paradigm of Function as a service itself and people re-inventing the wheel by creating custom made code and hack to fit each of these provider runtimes.

Fixed timeouts

Latencies

Start up time

Library loading time

Security issues in a multitenant environment

Function caches

Containers introducing bugs code shopping data shipping between functions adding to the latency, cost, and inconvenience. From a technical point of you this can be described as serverless architecture being a data shipping one rather than a code or function shipping one. Meaning, rather that moving the code to the platform where the data is and executing it there, serverless follows the paradigm where

2.2 Stream Processing/ETLs

2.3 Problem statement

2.3.1 With the current state of development in the field of streaming and other data

intensive applications, a serverless/FaaS platform could really help save resources and hence operational cost of applications.

2.3.2 Having a platform that can take care of the resource provisioning for you, when

you can focus on the program logic and the data engineering side, helps a lot of domain specific engineers test out and deploy their applications easily.

2.3.3 A lot of the existing platforms already do it, but most of these solutions available

commercially are extremely vendor locked in. The limitations are set for you by the cloud providers and is often very difficult to fiddle with it or to extend the system so as to support an extra runtime etc.

2.3.4 Along with this, the way current FaaS offerings deal with function compositions

and parallelism are extremely clumsy and almost always explicit. While this lets the providers have a very generic way of dealing with the platform and holds to the one way to code them all paradigm, the gateways often tend to be a bottleneck. Also the data transfer between functions always depend on a storage based off of Block IO which contribute to the latency immensely.

2.3.5 I propose a platform based off of an OpenSource FaaS infrastructure that can be

maintained by the companies which can offer a multitenant and completely elastic platform to deploy their data intensive and high throughput applications on. When I say completely elastic, it means that the intermediate datastore tend to be ephermeral and that scales to based on the usage incurred by the system and offers fine grained usage monitoring and billing if need be. Alongside, providing an easy to use API that lets one compose functions

:UNNUMBERED: t