Architecture Decision on using Microservices or Serverless Functions with Containers

Baskaran Jambunathan
Vels University,
Chennai,India
shriyabaskaran@yahoo.com

Kalpana Yoganathan
Vels University,
Chennai, India
ykalpanaravi@gmail.com

Abstract: Cloud adoption is gaining lots of momentum across the globe and enterprise are focussing not only migration on to cloud but also on developing cloud native application. There are lots of focuses on reducing and optimizing resources and hence developing application in a serverless fashion is going to be the key in the industry. Many organizations are working on application modernization and developing distributed application and hence microservice is the key focus area in converting their monolithic into microservices and use containers for easy portability across the platform and makes it more platform neutral. There are high elements of focus on whether to go for serverless or Microservice mode and should we use containers for deployment is the key debate among the people who are working in this area and are still not clear which way to go forward in the given situation. In this article we would like to explore and discuss about each technology in details and analyse the advantage and challenge of these emerging areas and suggest the ideal way to move forward in cloud.

Keywords- Containers, dockers, Microservices, Serverless, Functions, Lambda, Kubernetes

I.INTRODUCTION

Application hosting on Cloud is becoming more popular and becoming inevitable these days and organization are either forced to migrate the existing application onto cloud or developing cloud native application using services provided by service providers. In the process they are trying to re-architect, re-host, re-platform their application to the new platform or architecture and would like to exploit the features provided by the service providers and make it effective in the cloud environment. Application development in cloud becomes more modular, interoperable and distributive in nature and many new evolving technologies like docker[1] containers, microservices[2] and serverless[3] architecture are helping to meet this objective and achieve the desired result. In addition, organization wants to reduce the cost and wanted to develop and host the application as and when they need it and avoid the cost of owning and running the server on their own. Cloud service provider provides options these days to host and run the application on demand and provide API / services to do so and make the application more event based and hence when they need it or when the event occurs, the application is getting triggered and hence wanted to make use of this on-demand approach to reduce the cost and operational overhead of managing the servers on their own. We would like to introduce each technology in details and discuss in this article on the merits and demerits of all the three emerging technologies – serverless computing, microservices and container and how it is going to support cloud based application development and hosting.

II.SERVERLESS COMPUTING

Serverless computing means that it is an event driven programing where the compute resource needed to run the code is managed as a scalable resource by cloud service provider instead of us managing the same. It refers to platforms that allow an organization to run a specific piece of code on-demand. It's called serverless because the organization does not have to maintain a physical or virtual server to execute the code. In any traditional organization, they have to maintain and manage the compute resources whether they are using it or not. In serverless model, we can host and run our code in the computer resource provided by the service provider and need to pay to for the usage and no overhead of maintain and manage the server and hence no overhead cost on idle and downtime of servers. Serverless does not eliminate the server, but it only helps to move it to the backend when we are doing the design of our application or code.

Function as a service[3] (FaaS) refers to a category of cloud services where developers can develop, build and run the application without worrying on the complexity of building and managing the infrastructure that is associated with develop and run the code. FaaS is the concept of serverless computing via serverless architectures. They are expected to start within milliseconds and process individual requests and then the process ends. Building an application following this model is one way of achieving a "serverless" architecture, and is typically used when building microservices applications. Following are its characteristics.

- Complete abstraction of servers away from the

develop

- Billing based on consumption and executions, not server instance sizes
- Services that are event-driven and instantaneously scalable

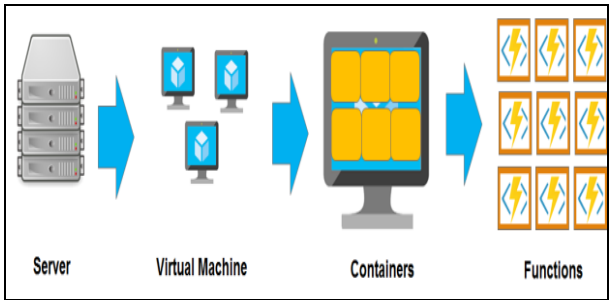

Fig.1: Evolution of Functions

One of the challenges with FaaS is monitoring the function apps. We still need to understand how often they occur, how long they take, and potentially, why they are slow. Since we don't necessarily have a server or control the resources they are running on, you can't install any monitoring software.

PaaS vs FaaS: Platform-as-a-Service greatly simplifies deploying applications. It allows us to deploy the app and the cloud service provider worries about how to deploy the servers to run it. Most PaaS hosting options can even auto-scale the number of servers to handle workloads and save money during times of low usage. Function-as-a-Service provides the ability to deploy what is essentially a single function, or part of an application. When deployed as PaaS, an application is typically running on at least one server at all times. With FaaS, it may not be running at all until the function needs to be executed. It starts the function within a few milliseconds and then shuts it down. Both provide the ability to easily deploy an application and scale it, without having to provision or configure servers.

Serverless can also mean applications where some amount of server-side logic is still written by the application developer, but unlike traditional architectures, it is run in stateless compute containers that are event-triggered, ephemeral, and fully managed by a third party. These types of services have been described as BaaS or Backend-as-a-Service. Such architectures remove the need for the traditional 'always on' server system sitting behind an application and depending on the circumstances, such systems can significantly reduce operational cost as it runs on-demand only.

Serverless architectures refer to applications that significantly depend on third-party services (knows as Backend as a Service or "BaaS") or on custom code that's run in ephemeral containers (Function as a Service or "FaaS").

TABLE 1: Comparison of VM, Containers and Serverless

Properties	*Virtual Machine*	*Containers*	*Serverless*
Unit of Scale	*Machine*	*Application*	*Function*
Abstraction	*Hardware*	*Operation System*	*language runtime*
Packaging	*Image*	*Container File*	*Code*
Configure	Machine, storage, Network, Operating system	Run servers, Configure application, scaling	Run code when needed
Execution	Multi-threaded, Multi task	Multi-threaded, single task	Single threaded, single task
Run time	Hours to months	Minutes to days	Microseconds to seconds
Unit of cost	Per VM per hour	Per VM per hour	Per memory/second per request

It replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. It has the following advantages in general

- Decreased time to marke
- Enhanced scalability
- Improved latency
- Reduced Operations Cost and increased development time with improved user experience

Although there are some advantages, it also has few drawbacks like any other technology such as,

- Vendor control, multitenancy problems, vendor lock-in, and security concerns are some of the problems due to the use of third-party APIs.
- Developers are dependent on vendors for debugging and monitoring tools.
- Architecture complexity - It gets cumbersome to manage too many functions, and ignoring granularity will end up creating mini-monoliths.
- Integration testing serverless apps is tough. The units of integration with Serverless (Functions) are a lot smaller than with other architectures and therefore we rely on integration testing a lot more than we may do with other architectural styles.

SERVERLESS OPTION WITH CLOUD SERVICE PROVIDERS

With increased popularity of Serverless architecture, cloud service providers like Amazon web services (AWS) and Microsoft Azure provides support for serverless

computing. AWS has Lambda[4] functions which support event based serverless computing, while Azure has "Functions"[5] to address this area.

AWS lambda service lets us run the code without provisioning or managing the servers. It lets us execute the code from few request to even thousands in a short time and scales up automatically as per the demand We need to pay as per use and need not pay when the code is not running. With AWS Lambda, we can run code for virtually any type of application or backend service - all with zero administration. AWS Lambda runs our code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All we need to do is supply your code in one of the languages that AWS Lambda[4] supports (currently Node.js, Java, C# and Python). In AWS lambda, we can run our code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table. With these capabilities, we can use Lambda to easily build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB process streaming data stored in Kinesis, or create your own back end that operates at AWS scale, performance, and security.

Azure Functions lets you develop serverless applications on Microsoft Azure. It is an event driven, compute-on-demand experience that extends the existing Azure application platform with capabilities to implement code triggered by events occurring in virtually any Azure or 3rd party service as well as on-premises systems. It allows developers to take action by connecting to data sources or messaging solutions, thus making it easy to process and react to events. It scales based on demand and we pay only for the resources we consume. It supports triggers, which are ways to start execution of your code, and bindings, which are ways to simplify coding for input and output data.

The intention here is not to discuss these features in detail, rather we want to analyse how serverless, microservices and containers are evolving in today's scenario and how do we choose the right approach for building our cloud solutions and their limitations which will help organisation to decide the right choice of tools and technology in their cloud transformation journey. In the below sections authors will discuss on this further and provide their recommendations based on their experience and analysis.

CONTAINERS AND MICROSERVICES

In our earlier research [1][2] we have analysed and discussed a lot about containers and microservices on how and where it is being used and its advantages. In order to bring the context into the current discussion, we just want to touch upon briefly on these technologies in comparison. Containers, as we discussed are light weight, portable, and can be easily provisioned and deployed on to any platform whether it is datacentre or cloud – in a virtual environment. It is highly secured, scalable and language neutral. There are many containers available but docker is the most accepted one and are open source, have its own ecosystem to manage. Like VM, container also has its own life cycle to manage and every cloud service provider, be it a public cloud like AWS, Azure, Google or private cloud like OpenStack, VMware supports Docker containers. For example – AWS has elastic container service (ECS)[4] and Azure provides Azure container services. They manage with the native services and also provide supports to Dockers. In addition, all these service providers support DevOps services to manage operations and support Orchestration tools like Kubernetes[2], Docker Swarm[6] and Mesos[6].

Microservices are the new paradigm of application development where application is converted from monolithic into microservices. It decomposes the application into set of collaborating services – each service implement a set of narrowly related functions and communicate with each other with either synchronous or asynchronous protocols. The services are designed based on business domains and a group of services can address business functionality. These services are small in size, isolated, easy to scale and fault tolerant. They are meant for developing distributed applications. They are developed and deployed independently and some time it has its own database as well and makes it completely isolated. Another advantage is that it collaborates well and services written in different languages can interact with each other while addressing the business functions. It also simplifies build and release process and support complete devops functionality. Every cloud service provider supports Microservices architecture and has its own API and services to support this development model and enable developers to build a truly distributed application.

The relation between containers (Dockers) and microservices is that, it's isolated language neutral and distributive nature helps developers to easily containerise the application and deploy across. When we convert a monolithic to microservices or when we design a new microservice based application, we create services which are self-contained and able to perform the task independently and can be scaled to meet the demand. Hence each service can be deployed in a container and can be ported across platform to make it secured and truly distributed in nature. Hence containers portability and light weight behaviour and microservice, scalable, and distributive nature – gel well with each other and helps in modernizing application to meet the business demand.

MICROSERVICES AND SERVERLESS

Microservices is the approach of breaking down a large application into a collection of smaller, modular services. The benefit of this approach is that it becomes possible to build, test, and deploy individual services without impacting the entire product or other services adversely. As microservices enable smaller, faster releases, they allow new features to be released to only a subset of users initially, and then to the entire user base once the feature meets quality expectations. Even though microservices architecture is more complex, especially at the start, it brings much-needed speed, agility, reliability, and scalability, which are critical to today's DevOps teams.

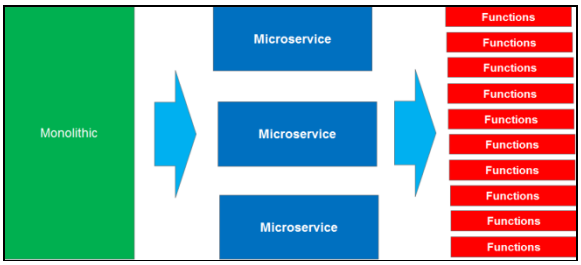

Fig.2: Conversion of Monolithic to Functions

Serverless computing is the concept of abstracting away the infrastructure server layer, so that developers can focus on building an app without worrying about scalability, availability, and security of hardware servers. From a usage point of view, the application utilizes as much networking and storage as it needs. This means that when there's little usage, hardware resources aren't lying idle; at the same time, when there are spikes in usage, the infrastructure can scale to any extent without having to manually provision new servers on the fly.

Serverless computing is a key enabler for microservices-based applications. It makes infrastructure event-driven, completely controlled by the needs of each service that make up an application. Developer can have a microservice framework that does not use Serverless Architecture. A company can decide to run their microservices on a fleet of EC2[5] instances, for instance. And in theory, you could have a serverless architecture that isn't a microservice. That being said, Microservices and Serverless Architecture are nice compliments to one another. Microservices is a technical term and serverless computing is more a marketing term like the Cloud. Serverless computing is not really a technical term. It refers to a system where deployment of a software service is done on a cloudy platform, where the people who develop and deploy a service are not really interested in the underlying infrastructure. serverless computing more as a marketing term than a technical term

As for the relationship between serverless computing and microservices architecture, microservices architecture allows small teams of developers to focus on separate services, each providing a specific task, and serverless computing helps them get these microservices up and running with the least effort.

CONTAINER AND SERVERLESS

Some people think that serverless revolution is a threat to Docker containers. This is because they perceive serverless functions as an even more efficient way to deploy application code than docker containers. In one sense, it doubles down on the efficiencies that docker containers offer. Docker containers reduce the management burden associated with virtual machines and also provide more scalability. Serverless functions take things a step further. They eliminate the need to set up and manage any type of infrastructure, beyond the very basic task of loading the code for serverless functions into the serverless platform.

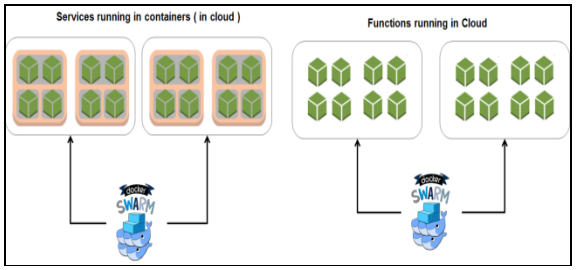

Fig.3: Services Vs Functions running in cloud

Serverless doesn't mean there is no need for Docker, and in fact developer can also consider Docker as serverless. One can use Docker to containerize these functions and run them on-demand on Docker Swarm – which is a container orchestration platform. Serverless is a technique for building distributed apps and Docker is the perfect platform for building them on. Serverless architecture was a good fit for containers as it is short, ephemeral functions invoked on demand and later user can shut down when there is no need. In addition, containers offer a high standard pre-packaging method for the functions. Developer simply can't use serverless functions to accomplish most of the tasks he does with docker containers, and vice versa. He needs both for many situations. Following are some of the situation one can think of.

- Run functions in the code as on-demand Docker containers
- Use a Swarm to run these on a cluster
- Run containers from containers, by passing a docker API socket

Docker is the perfect platform to build serverless applications. For example, the speed in which containers and the Functions execute could be a bottleneck. Deploying stacks of containers and functions is also

another consideration. Finally, the operational side like logging, monitoring needs to ramp up to support the serverless approach and understand this new operating model.

III.SCAR – SERVERLESS CONTAINER AWARE ARCHITECTURE

SCAR[7] is a framework to transparently execute containers out of Docker images in AWS Lambda, in order to run applications and code in virtually any programming language. SCAR provides the benefits of AWS Lambda with the execution environment you decide, provided as a docker image available in dockerhub. It is probably the easiest, most convenient approach to run generic applications on AWS Lambda, as well as code in your favourite programming language, not only in those languages supported by AWS Lambda.
SCAR also supports a High Throughput Computing Programming Model to create highly-parallel event-driven file-processing serverless applications that execute on customized runtime environments provided by Docker containers run on AWS Lambda.

IV.ANALYSIS AND RECOMMENDATIONS

In our detailed analysis and experimentation of the above three techniques along with supporting technologies, Author have found the following as the key observations. After careful study of these techniques, author realised that each of these approach has its own merits and demerits, based on the type of implementation one wanted to do, combination of these would yield a desired outcome and sometimes, they complement each other to provide the ideal results.

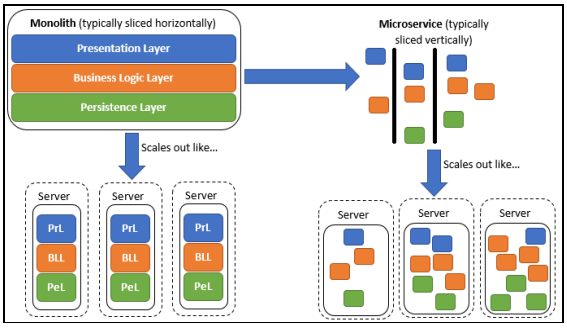

Fig.4: Deployment of Microservices and Functins

Containers – the basic for application hosting and management: Container is the ideal platform for managing the infrastructure better and can be grouped in to clusters and can be made distributed across regions or across cloud. There are few cluster management and orchestration tools like kubernetes, docker swarm and apache mesos are there to provide the ideal framework for managing the containers in the form of a cluster. Kubernetes has a concept called pods which help in grouping the containers in a cluster and can replicate them as well. It has load balancing, scheduling and service orchestration approach for managing the cluster effectively. Hence containers provide ideal platform to build a distributed, portable, scalable, secured applications and would be our first choice irrespective of whether developer chose microservices or serverless architecture. Containerisation helps in easy deployment in any cloud services or any PaaS platforms like Pivotal cloud Foundry[8], OpenShift[8] or Bluemix[8]. These PaaS providers have inbuilt kubernetes and container management platform to manage this containers effectively and made it as a self-service platform. It has complete support for build and release process including most of the devops features which enable them to host the containers and manage them completely. Mesos kind of tools provide the support to containers for even data centre servers or even our laptops and make the end to end orchestration services for containers. Hence in our analysis and observation, containers are the key for hosting the applications – whether it is monolithic, microservices or serverless and many framework provides complete support in managing containers – Dockers.

Choice between Microservices and Serverless code: As discussed earlier, both microservices and serverless code go well with containers. Microservices are in general designed based on business domain and to accomplish a particular business objective. Then can be modularised and can communicate to each other well in a truly distributed environment. It can scale well and can be secured and made independent, and also collaborate between them effectively. It is resilient, fault tolerant and failure of any service will not impact others and the overall system. The key advantage is, microservice does not necessarily need cloud services, or even virtualization. It can run anywhere within the runtime environment. Because of its modular in nature, one can easily containerize the application, dockerize, and deploy it in both public and private cloud environment. With help of the native cloud services, one can avail the scalability, security, fault tolerant features and that makes microservices more appropriate technique to use in the containerized cloud environment. It is not necessary that one need to use container or cloud or both, but with container services in cloud, microservices can be fully utilized and exploited and can be made as a de-facto standard to build modern applications which are ideally scalable,resilent and distributive in nature.

Serverless needs native cloud service support: Serverless functions cannot work in bare metal server or with Docker container alone. It needs the service provider's services to be hosted and executed. Each components of the application can be converted in to small functions and each one does a separate task at the time of any event trigger happens. It is well supported in event driven

programming, where at the time of any event triggered, functions gets executed. Multiple functions can collectively accomplish a particular transaction or task. There need not be any relation exist between functions as well. A microservice can be split in to multiple functions as well. Serverless code deployed in the container, needs serverless environment like AWS lambda or Azure functions to execute the code and deliver the result. We can deploy serverless functions within the container and deploy the container within AWS or Azure or Google environment. The only advantage of serverless is that it can be executed on demand and does not need a runtime environment always allocated to us. Since microservices and serverless can complement each other, one need to choose micro services, where code needs to run continuously as per business needs and use serverless functions – where code needs to run when a particular event occurs. There is no point in writing functions when one expect the code to run always.

Containers and Microservices for distributed application development: As discussed earlier, Microservices go well with containers as containers are the new paradigm of application development, where it can be componentized, portable, and language neutral so that it can communicate between each other well and it does not need any interface to interact between them. Containers are compact, small unit for deployment and share the runtime environment between them, it is easy to deploy microservice within the containers and can be grouped, replicated easily as a image and can be shared across the development groups. Serverless cannot communicate with each other as it is not meant for that, each one is designed to do a particular job independently, at the occurrence of a particular event. With help of any container orchestration platform, microservices can be more effective and in service discovery, orchestration and collaboration in a truly distributed environment.

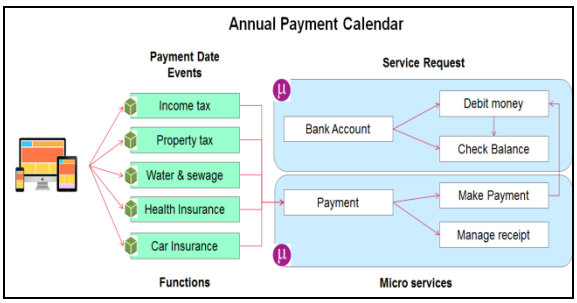

Fig.5: Design of Functions and Microservices.

Authour would like to explain the above points with a small use case where a user has to make a payment annually to multiple forms. Each of the event triggers before the cut-off dates and reminds the users. These happen once in a year and trigger an event, and the corresponding functions will be activated when it has to execute the payment task. Authours have also identified two microservices which interact with these functions and accomplish the task intended. The combination of services and functions demonstrate the ideal choice of technology to accomplish the task. Use functions for the event based stateless task and use microservices for long running process as discussed earlier.

In summary – following are our recommendations in the choice of technology.

a) Serverless functions are for stateless, short lived process, whereas micro services are for stateful[7], long running process. Hence chose the right one for your design.
b) Functions scales per request, user cannot under or over provision capacity. It is not in user control. Where containers are in user control and can be provisioned and managed as per the need.
c) Use Functions when it is really needed means, when an action has to happen at the occurrence of an event, functions get triggered. Hence need to use appropriately.
d) Although a single microservice can be converted into multiple functions, it cannot be monitored or measured hence; too many functions will impact the performance.
e) Containerizing microservices makes sense as it runs in an isolated, independent form, where as it is not always recommended to containerize functions as it is not completely independent.

Functions can be extensively used IOT (Internet of things) based applications where event is the key, where as any numerical analytics, data simulation, video streaming, machine learning kind of applications needs microservices.

V.CONCLUSION

From the above discussion of analysing serverless, microservices and docker container, Authors found that each technology has its own merits and demerits. Based on the type of use cases and the business needs, users can identify carefully which one to use and package it accordingly based on the functional need. Serverless are for event based programing where an action has to take place when an event occurs and it is mostly function driven approach where the control is with the host and not with the user. Microservices are for of modular, domain driver, highly self-contained group of services, which can work independently. A microservice can have multiple functions and hence the choice depends on the situation. Contanier can host both microservices and functions and the way one need to package is left to the architect who design the solution. Hence all the three are essential in building modern cloud based solution and by right choice of approach will make us to build a highly robust application with minimal cost and highly resilient,

scalable, secured distributed application.

REFERENCES

[1]. Baskaran Jambunathan and Dr Kalpana Yoganathan ‘” Multi Cloud Deployment with Containers”, International Journal of Engineering and Technology (IJET) - e-ISSN : 0975-4024

[2]. Baskaran Jambunathan and Dr Kalpana Yoganathan ,’Microservice Design for Container based Multi-cloud Deployment”, Jour of Adv Research in Dynamical & Control Systems, 04-Special Issue, June 2017

[3]. Serverless Programming (Function as a Service) - Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on 5-8 June 2017

[4]. AWS lambda - https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf

[5]. Getting started guide for Azure developers - https://docsmsftpdfs.blob.core.windows.net/guides/azure/azure-developer-guide.pdf

[6]. Kubernetes vs. Mesos vs. Swarm — An Opinionated Discussionhttps://blog.outlyer.com/kubernetes-vs.-mesos-vs.-swarm.

[7]. SCAR: Serverless Container-aware Architectures - Serverless Status Issue 13: July 28, 2017 - https://serverless.email/issues/13

[8]. Free PaaS Compared : OpenShift Versus IBM BlueMix https://thecustomizewindows.com/2015/07/free-paas-compared-openshift-versus-ibm-bluemix/