When to use AWS Serverless vs. server based architecture?

Overview

Practically any software base solution can be hosted on AWS. Following lists top category of use cases.

- Web sites and web applications (Always available)
- Data processing and analytics (Big data batch/streaming processing applications, AI/DL/ML)
- Gaming
- IoT
- Back ends engines (micro serves; mobile apps)
- Chatbots – powering chatbot logic
- Amazon Alexa
- IT automation (policy engines; infrastructure mgmt.)

These use cases can be implemented using server-based (EC2, ECS, EKS), serverless or a combination architecture.

Serverless architecture is a methodology where the application developers outsource the provisioning, resource-management, security-management, scaling, fault management of the compute resources (on which their applications run) to the Cloud-service providers. In the case of server-based architectures, application providers would like to have a tighter control on the compute resources at the cost of performing the provisioning, resource-management, security-management, scaling, fault management for their application.

So why invent scaling, fault management and provisioning, resource-management, security-management for your application cloud can provide it for you?

Let us find out!!

Deep dive

Trend

Serverless architectures have recently gained very high popularity. Google search trends show that queries for the term "serverless" recently matched the historic peak of popularity of the phrase"MapReduce"or"MapReduce"(Figure1). Ref[1].

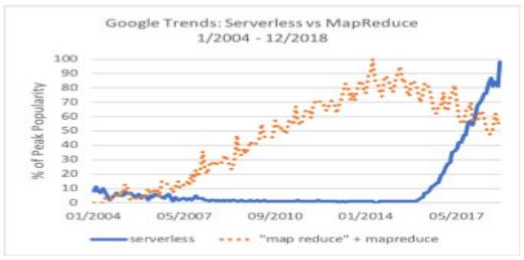

Figure 1: Google Trends for "Serverless" and "Map Reduce" from 2004 to time of publication.

Understanding serverless

Top reasons to select serverless architecture are:

1. Clients don’t have to manage server (AWS does it) instances
2. Better cost control by eliminating cost for idling (pay as you go only for used storage and compute)
3. Automatic scale and reliability
4. Automatic handling of multi-tenancy
5. Delegation of larger portion of security responsibility to AWS.

Serverless compute paradigm (Simplistic view)

1. Similar to programming concept each computation is defined as a function (event handler) that takes input and provides an output that can be persisted on storage/DB or delivered to a target consumer (consumer app, back-end app, another function down the pipeline etc.). The cloud offered is called “Function as a service”. AWS FaaS is Lambda.
2. Event handlers for specific event types are registered in the serverless framework (Lambda)
3. When an event arrives for computations, the serverless framework(Lambda) finds the compute (runtime for Lambda function) to trigger the event handler (lambda function) for execution.
    a. In case the event-handler is already running, the events are sent to the event-handler (lambda functions) in operations.
    b. If event-handler for the corresponding event is not running, serverless framework will allocate a run-time (find compute) to initiate/start the event handler and route the event to the freshly started event-handler for execution.
    c. Event handlers (lambda functions) is an independent task and do not need to communicate with other functions. Lambda-function(event-handler) is ideal for executing one functional operation

Serverless functions can be divided into the following types:

1. Mapping function (Examples from [2]: Image resizing, video transcoding)
2. Inline/pre-processing (Examples from[2]: Data validation, data filtering, data transformation) before sending to processors down the pipeline (Analytics engines, query engines etc) running on server-based services
3. Lambda pipeline that are linked via messaging system like (SQS) or object store

Comparisons between Serverless and server-based compute

Serverless	**Server-based**
Compute resources provisioned and managed by Cloud provider	Compute resource provisioned and managed by the application provider
A lambda function is short-lived and shuts-down after a fixed time (15 minutes for AWS Lambda and similar timeout for other cloud services like GCP and Azure)	Service can be run as long as needed
Data needs to be shipped to code. In efficient for big-data processing	Can run the code at the data. Example EMR
Lambda-functions may have to be loaded into run-time (allocate compute resources) to process the events.	As service runs as long as needed, service is always up eliminating the run-time cost and time needed to process a request
Scalability, availability managed by AWS	Scalability and availability have to be managed by the application provider.

	NOTE: Even when using auto-scale, application provider has to appropriately setup for auto-scaling which is not needed in the server-less model

So it is obvious, the down side of serverless architecture is lack of control on the cloud resources where the application runs which prevents end to end fine-tuning of the application i.e. preventing precise resource optimization and responsiveness.

Decision making

While the serverless approach guarantees resources, reasonable response time and useful for many use cases a conscious decision shall be made on serverless verses server-base compute for the target application in question.

So when should serverless architecture “not” be used? Here are a few questions to ask

1. What if your application needs a very fast and predictable response time (for example real time applications)?
2. What if you need to move large amounts of data, quickly across a data processing pipeline/workloads i.e. where data processing requires large memory?
3. What if you have legacy code that you want to deploy with minimal re-architecture?
4. What if you have to perform complex chain of operations involving multiple functions with the constraint of response time?
5. When large amount of data needs to be processed (by shipping code to data rather than data to code)
6. When very large amounts of memory is needed (FaaS has limitation on the maximum RAM size)

If the answer to any of the above questions are “yes”, server-based architecture shall be considered.

Scope of the discussion*:*

1. *These arguments are applicable to architectures for proprietarily developed application, requiring decisions to run target application on a server based architecture or server-less (FaaS) services like AWS Lamba.*

2. *The discussion excludes the use cases requiring custom/specialized hardware or well defined specialized cloud services like AWS Kinesis.*

Once you decide what architecture is best for your application, next step is to put the architecture is place.

Let us see a simplistic view of server-based and server-less architectures.

Architecture patterns

Figure 1 and 2 shows the typical architecture pattern for a server based and serverless architecture respectively.

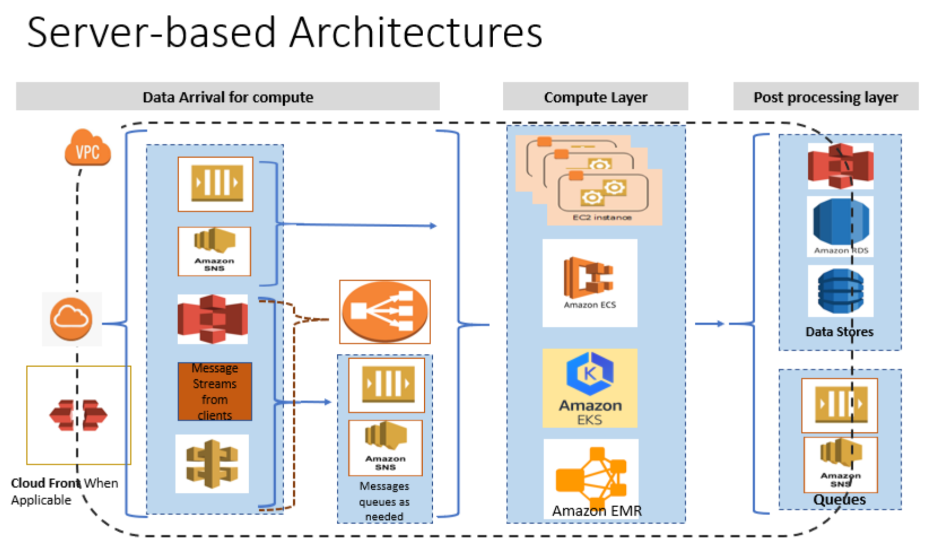

Figure 1 - Server-based architecture pattern

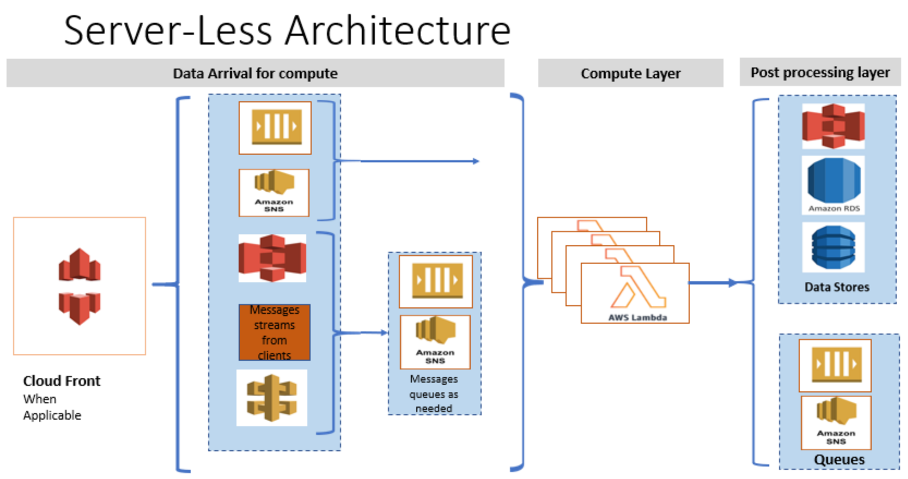

Figure 2 - Server-less architecture pattern

Each architecture involves 3 logical stages:

1. Stage 1: Data arrival for computations
2. Stage 2: Data computation/processing
3. Stage 3: Post processing stage, where the data is sent to the pre-defined destination.

Stage 1: Arrival of data for computations

Depending on the type of application this stage typically consists of an optional CloudFront, request/object receiver and an optional message buffer or a load balancer.

1. CloudFront: Used for content delivery related applications. This is not required component for all applications. Appropriate evaluation is required for justification of CloudFront in the applicable architecture.
2. The event notification (generated as a result of user action, machine action i.e. Iot Device, computation upstream by another component etc) for computation is logged into one of the following components.
    a. Messge queues: Notification produced by the publisher (user or machine) is sent to one of the following message queues for processing by the compute layer down:

        i. SNS
        ii. SQS
    b. Object store/Streams/API handler:
        i. S3 bucket: Typically used by application using bulky objects
        ii. Message streams: Streams from user apps, networking events or IOT.
        iii. API Gateway: Used by the applications using a well published API served by AWS API gateway.

3. Temporary data staging before compute
    a. Event notifications from Object store/Streams/API handler can be sent directly to the compute or sent to the queue (SQN or SNS) for temporary staging before the computations.
    b. ELB: Events from Object store/Streams/API handler could be optionally sent to the AWS Elastic Load Balancer especially when "Server-based" architecture is used to efficiently load balance the traffic cross the scaled server-instances running the computations

Stage 2: Computation stage

4. Server-based compute: These architectures uses the one of the following types of computes for processing the notification/data.
    a. EC2
    b. ECS
    c. ECK
    d. EMR
5. Server-less compute:
    a. AWS Lambda framework. Application write the Lambda function and loads into the Lambda framework. Messages/notifications from Stage 1 triggers the Lambda functions.

Stage 3: Post processing layer.

Data is either delivered directly out to internet consumer or loaded on the data stores or queues.

6. Data stores: Processed data from the compute stage is stored on one or more of the following components for consumption.
    a. S3: This could be processed objects to be fetched by the consumers later.
    b. Amazon RDS or any other relational DB: Relational data from compute stage is stored here.
    c. Amazon DynamoDB or any other NoSQL: NoSQL data from the compute stage is loaded here for later consumption.
7. Message queues: Compute stage can send the processed data to the message queues like SNS and SQS for temporary staging of the results for consumption ASAP.
8. NOTE: In the case of server-based applications, the data stores and messages queues can be either inside or outside the VPC depending on the type, access permission of consumers.

Reference:

1. ServerlessComputing:OneStepForward,TwoStepsBack JosephM.Hellerstein,JoseFaleiro,JosephE.Gonzalez,JohannSchleier-Smith,VikramSreekanti, AlexeyTumanovandChenggangWu UCBerkeley
2. AWS Lambda: https://aws.amazon.com/lambda/