AWS Serverless Multi-Tier Architectures

Using Amazon API Gateway and AWS Lambda

November 2015

© 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract 3
Introduction 4
Three-tier Architecture Overview 5
The Serverless Logic Tier 6
Amazon API Gateway 6
AWS Lambda 9
The Data Tier 12
The Presentation Tier 14
Sample Architecture Patterns 14
Mobile Back End 15
Amazon S3 Hosted Website 16
Microservices Environment 17
Conclusion 18
Contributors 18
Notes 19

Abstract

This whitepaper shows you how innovations from Amazon Web Services (AWS) can change how you can design multi-tiered architectures for popular patterns such as microservices, mobile back ends, and public websites. Architects and developers can now use an implementation pattern that includes Amazon API Gateway and AWS Lambda to reduce the development and operations cycles required to create and operationally manage multi-tiered applications.

Introduction

The multi-tier application (three-tier, n-tier, etc.) has been a cornerstone architecture pattern for decades. The multi-tier pattern provides good guidelines for you to follow to ensure decoupled and scalable application components that can be separately managed and maintained (often by distinct teams). Multi-tiered applications are often built using a service-oriented architecture (SOA) approach to using web services. In this approach, the network acts as the boundary between tiers. However, there are many undifferentiated aspects of creating a new web service tier as part of your application. Much of the code written within a multi-tier web application is a direct result of the pattern itself. Examples include 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 in an undesired way.

Amazon API Gateway[1], a service for creating and managing APIs, and AWS Lambda[2], a service for running arbitrary code functions, can be used together to simplify the creation of robust multi-tier applications.

Amazon API Gateway's integration with AWS Lambda enables user defined code functions to be triggered directly via a user-defined HTTPS request. Regardless of the request volume required, both the API Gateway and Lambda will scale automatically to support exactly the needs of your application. When combined, you can create a tier for your application that allows you to write the code that matters to your application and *not* focus on various other undifferentiating aspects of implementing a multi-tiered architecture—like architecting for high availability, writing client SDKs, server/operating system (OS) management, scaling, and implementing a client authorization mechanism.

More recently, AWS has announced the ability to create Lambda functions that execute within your Amazon Virtual Private Cloud (Amazon VPC)[3]. This feature extends the benefits of combining API Gateway and Lambda to include a variety of use cases where network privacy is required. For example, when you need to integrate your web service with a relational database that contains sensitive information. The integration of Lambda and Amazon VPC has indirectly expanded the capabilities of Amazon API Gateway because it gives developers the

ability to define their own set of Internet-accessible HTTPS APIs in front of a backend that remains private and secure as part of Amazon VPC. You can observe the benefits of this powerful pattern across each tier of a multi-tiered architecture. This whitepaper focuses on the most popular example of a multi-tiered architecture, the **three-tier** web application. However, you can apply this multi-tier pattern well beyond a typical three-tier web application.

Three-tier Architecture Overview

The three-tier architecture is a popular pattern for user-facing applications. The tiers that comprise this 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.

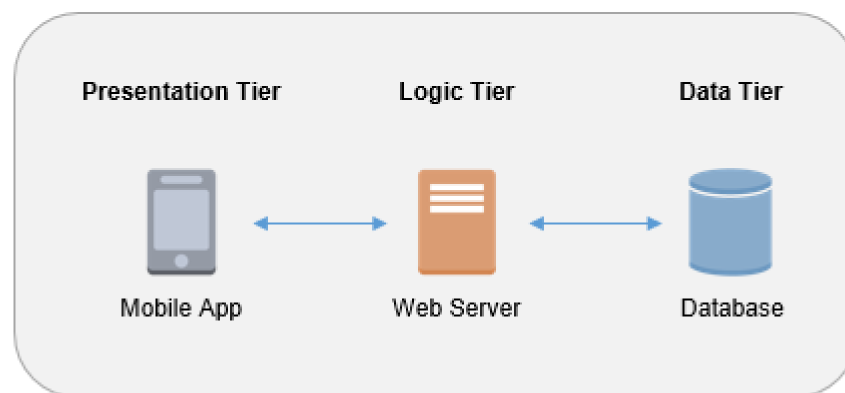

Figure 1: Architectural pattern for a simple three-tier application

There are many great resources online where you can learn more about the *general* three-tier architecture pattern. This whitepaper focuses on a specific implementation pattern for this architecture using Amazon API Gateway and AWS Lambda.

The Serverless Logic Tier

The logic tier of the three-tier architecture represents the brains of the application. This is why integrating Amazon API Gateway and AWS Lambda to form your logic tier can be so revolutionary. The features of the two services allow you to build a serverless production application that is highly available, scalable, and secure. Your application could use thousands of servers, however by leveraging this pattern you do not have to manage a single one. In addition, by using these managed services together you gain the 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.

Amazon API Gateway

Amazon API Gateway is a fully managed service for defining, deploying, and maintaining APIs. Clients integrate with the APIs using standard HTTPS requests. Its applicability to a service-oriented multi-tier architecture is obvious. However, it has specific features and qualities that result it being a powerful edge for your logic tier.

Integration with AWS Lambda

Amazon API Gateway gives your application a simple way (HTTPS requests) to leverage the innovation of AWS Lambda directly. API Gateway forms the bridge that connects your presentation tier and the functions you write in AWS Lambda. After defining the client/server relationship using your API, the contents of the client's HTTPS request are passed to Lambda function for execution. Those contents include request metadata, request headers, and the request body.

Stable API Performance Across the Globe

Each deployment of Amazon API Gateway includes an Amazon CloudFront[4] distribution under the covers. Amazon CloudFront is a content delivery web

service that uses Amazon's global network of edge locations as connection points for clients integrating with your API. This helps drive down the total response time latency of your API. Through its use of multiple edge locations across the world, Amazon CloudFront also provides you capabilities to combat distributed denial of service (DDoS) attack scenarios. For more information– read the AWS Best Practices for Combatting DDoS Attacks[5] whitepaper.

You can improve the performance of specific API requests by using Amazon API Gateway to store responses in an optional in-memory cache. This not only provides performance benefits for repeated API requests, but it also reduces backend executions, which can reduce your overall cost.

Encourage Innovation

The development work required to build any new application is an investment. You need to justify that in order for the project to begin. By reducing the amount of investment required for development tasks and time, you are free to more experiment and innovate more freely.

For many multi-tier web-service-based applications, the presentation tier is easily fragmented among users (separate mobile devices, web browsers, etc.). Those users are also often not bound geographically. A decoupled logic tier, however, is not physically fragmented by the users. All users depend on the same infrastructure running your logic tier, which magnifies the importance of the infrastructure. Cutting corners when initially implementing your logic tier ("we don't need to instrument metrics at initial launch;" "initial usage will be low, we'll worry about how to scale later;" etc.) is often proposed as a mechanism to deliver a new application faster. This can lead to technical debt and operational risk when you have to deploy those changes to an application already running in production. Amazon API Gateway allows *you* to cut those corners and deliver faster because the service has already implemented them for you.

The overall lifetime of an application might be unknown, or it might be known to be short-lived. Creating a business case for a new multi-tier application can be difficult for these reasons. It can be made easier when your starting point already includes the managed features that Amazon API Gateway provides, and where you only begin to incur infrastructure costs after your APIs begin receiving requests. For more information, see Amazon API Gateway Pricing.[6]

Iterate Rapidly, Stay Agile

With new applications, the user base may still be poorly defined (size, usage patterns, etc.). The logic tier must remain agile while the user base takes shape. Your application and business should be able to shift and accommodate the changing expectations of your early adopters. Amazon API Gateway reduces the number of development cycles required to take an API from inception to deployment. Amazon API Gateway provides the ability to create Mock Integrations[7] that allow you to generate API responses directly from API Gateway that client applications can develop against while, in parallel, full backend logic is being developed. This benefit applies not only at an API's first deployment, but also after the business has decided that the application (and existing API) must pivot quickly in response to your users. API Gateway and AWS Lambda enable versioning so that existing functionality and client dependencies can continue undisturbed while new functionality is released as a separate API/function version.

Security

Implementing the logic tier of a public three-tier web application as a web service immediately elevates the topic of security. The application needs to ensure that only authorized clients have access to your logic tier (which is exposed over the network). The Amazon API Gateway addresses the topic of security through ways that can give you confidence that your backend is secure. For access control, do not rely on providing your client applications with static API key strings; these can be extracted from clients and used elsewhere. You can take advantage of several different ways in which Amazon API Gateway contributes to securing your logic tier:

- All requests to your APIs can be made via HTTPS to enable encryption in transit.
- Your AWS Lambda functions can restrict access so that there is a trust relationship only between a particular API within Amazon API Gateway and a particular function in AWS Lambda. There will be no other way to invoke that Lambda function except by using the API through which you've chosen to expose it.
- The Amazon API Gateway allows you to generate client SDKs to integrate with your APIs. That SDK also manages the signing of requests when APIs

require authentication. Those API credentials used on the client side for authentication are passed directly to your AWS Lambda function – where further authentication can occur within code that you own and write, if needed.

- Each resource/method combination that you create as part of your API is granted its own specific Amazon Resource Name (ARN) that can be referenced in AWS Identity and Access Management (IAM)[8] policies.
  - This means your APIs are treated as first class citizens along with the other AWS-owned APIs. IAM policies can be fine-grained; they can reference specific resources/methods of an API created using Amazon API Gateway.
  - API access is enforced by the IAM policies that you create outside the context of your application code. This means that you do not have to write any code to be aware of or enforce those access levels. Code cannot contain bugs or be exploited if it does not exist.
  - Authorizing clients using AWS Signature version 4 (SigV4)[9] authorization and IAM policies for API access allows those same credentials to restrict or permit access to other AWS services and resources as needed (for example, Amazon S3 buckets or Amazon DynamoDB tables).

AWS Lambda

At its core, AWS Lambda allows arbitrary code written in any of the supported languages (Node, JVM based, and Python as of November 2015) to be triggered in response to an event. That event can be one of several programmatic triggers that AWS makes available, called an **event source** (see currently supported event sources here[10]). Many popular use cases for AWS Lambda revolve around event-driven data processing workflows, such as processing files stored in Amazon Simple Storage Service (Amazon S3)[11] or streaming data records from Amazon Kinesis[12].

When used in conjunction with Amazon API Gateway, an AWS Lambda function can exist within the context of a typical web service, and it can be triggered directly by an HTTPS request. Amazon API Gateway acts as the front door for your logic tier, but now you need to execute the logic behind those APIs. That's where AWS Lambda comes in.

Your Business Logic Goes Here

AWS Lambda allows you to write code functions, called **handlers**, which will execute when triggered by an event. For example, you can write a handler that will trigger when an event such as an HTTPS request to your API occurs. Lambda allows you to create modular handlers at your chosen level of granularity (one per API or one per API method) that can be updated, invoked, and changed independently. The handler is then free to reach out to any other dependencies it has (such as other functions you've uploaded with your code, libraries, native binaries, or even external web services). Lambda allows you to package all of your required dependencies in your function definition during creation. When you create your function, you specify which method inside your deployment package will act as the request handler. You are free to reuse the same deployment package for multiple Lambda function definitions, where each Lambda function may have a unique handler within the same deployment package. In the server-less multi-tier architecture pattern, each one of the APIs you create in Amazon API Gateway will integrate with a Lambda function (and the handler within) that executes the business logic required.

Amazon VPC Integration

AWS Lambda, the core of your logic tier, will be the component directly integrating with the data tier. Because the data tier will often contain sensitive business or user information, the data tier should be tightly secure. For AWS services with which you can integrate from a Lambda function, you can manage access control using IAM policies. These services include Amazon S3, Amazon DynamoDB, Amazon Kinesis, Amazon Simple Queue Service (Amazon SQS), Amazon Simple Notification Service (Amazon SNS), other AWS Lambda functions, and more. However, you might have a component that governs its own access control, such as a relational database. With components such as this you could achieve better security by deploying them within a private networking environment—an Amazon Virtual Private Cloud (Amazon VPC)[13].

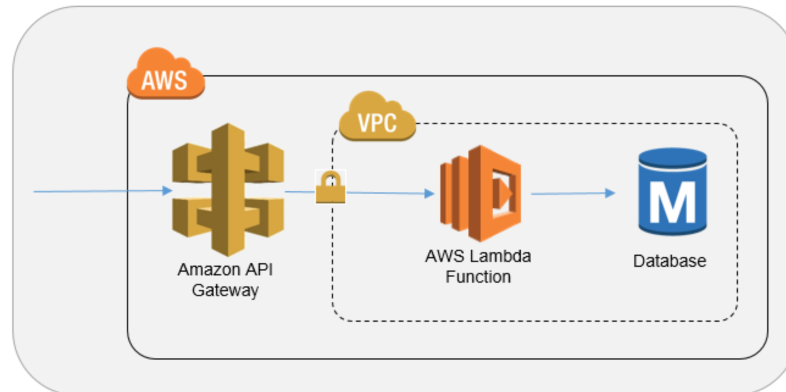

Figure 2: Architectural pattern using a VPC

The use of a VPC means the databases and other storage media that your business logic depends on can be made inaccessible over the Internet. The VPC also ensures that the *only* way to interact with your data from the Internet will be through the APIs that you've defined and the Lambda code functions that you've written.

Security

To execute a Lambda function, it must be triggered by an event or service that has been permitted to do so via an IAM policy. It is possible to create a Lambda function that cannot be executed *at all* unless it is invoked by an API Gateway request that you define. Your code will only process as part of your valid use case; defined by the API you've created.

Each Lambda function itself assumes an IAM role, an ability that must be granted via an IAM trust relationship. That IAM role defines the other AWS services/resources your Lambda function will be able to interact with (such as an Amazon DynamoDB table or an Amazon S3 bucket). The services your function has access to will be defined and controlled from outside of the function itself. This is subtle, but powerful. It allows the code you write to be free from storing or retrieving AWS credentials: This means you don't have to hard code API keys, and you don't have to write code to retrieve them and store them in memory. Enabling your Lambda function to call the services it's allowed to, as defined by its IAM role, is managed for you by the service itself.

The Data Tier

By using AWS Lambda as your logic tier, you have a wide number of data storage options for your data tier. These options fall into two broad categories: Amazon VPC hosted data stores and IAM-enabled data stores. AWS Lambda has the ability to securely integrate with both.

Amazon VPC Hosted Data Stores

The integration of AWS Lambda with Amazon VPC enables functions to integrate with a variety of data storage technologies in a private and secure manner.

- Amazon RDS[14]

  Use any of the engines that Amazon Relational Database Service (Amazon RDS) makes available. Connect to Amazon RDS directly from the code you've written in Lambda just as you would outside of Lambda, but with the advantage of simple integration with the AWS Key Management Service (AWS KMS) for database credential encryption.

- Amazon ElastiCache[15]

  Integrate your Lambda functions with a managed in-memory cache to boost the performance of your application.

- Amazon RedShift[16]

  You can build functions that securely query an enterprise data warehouse for the purpose of building reports, dashboards, or retrieving ad-hoc query results.

- Private web service hosted by Amazon Elastic Compute Cloud (Amazon EC2)[17]

  You might have existing applications running as a web service privately within a VPC. Make HTTP requests over your logically private VPC network from a Lambda function.

IAM-Enabled Data Stores

Because AWS Lambda is integrated with IAM, it can use IAM for securing integration with any AWS service that can be leveraged directly using the AWS APIs.

- Amazon DynamoDB[18]

  Amazon DynamoDB is the AWS infinitely scalable NoSQL database. Consider Amazon DynamoDB when you want to retrieve data records (400KB or smaller as of this writing) with single-digit millisecond performance, regardless of scale. Using Amazon DynamoDB fine-grained access control your Lambda functions can follow the best practice of least privilege when querying specific data in DynamoDB

- Amazon S3[19]

  Amazon Simple Storage Service (Amazon S3) provides Internet-scale object storage. Amazon S3 is designed for durability of 99.999999999% of objects, so consider using it when your application needs cheap, highly durable storage. In addition, Amazon S3 is designed for up to 99.99% availability of objects over a given year, so consider using it when your application requires highly available storage. Objects stored in Amazon S3 (files, images, logs, any binary data) can be accessed directly via HTTP. Lambda functions can communicate securely with Amazon S3 via virtual private endpoints, and data within S3 can be restricted to only the IAM policy associated with the Lambda function.

- Amazon Elasticsearch Service[20]

  Amazon Elasticsearch Service (Amazon ES) is a managed version of the popular search and analytics engine, Elasticsearch. Amazon ES provides managed provisioning of clusters, failure detection, and replacement of nodes; you can restrict access to the Amazon ES API by using IAM policies.

The Presentation Tier

Amazon API Gateway opens up a variety of presentation tier possibilities. An Internet-accessible HTTPS API can be consumed by any client capable of HTTPS communication. The following list contains common examples that you could consider using for your application's presentation tier:

- Mobile App: In addition to integrating with custom business logic via Amazon API Gateway and AWS Lambda, you could use Amazon Cognito[21] as a mechanism to create and manage user identities.
- Static website content (such as files hosted in Amazon S3): You can enable your Amazon API Gateway APIs to be cross-origin resource sharing (CORS)-compliant. This allows web browsers to directly invoke your APIs from within the static web pages.
- Any other HTTPS-enabled client device: Many connected devices are capable of communicating via HTTPS. There is nothing unique or proprietary about how clients communicate with the APIs you create using Amazon API Gateway; it is pure HTTPS. No specific client software or licenses are required.

Sample Architecture Patterns

You can implement the following popular architecture patterns using Amazon API Gateway and AWS Lambda as the glue that forms your logic tier. For each example, we will only use AWS services that do not require users to manage their own infrastructure.

Mobile Back End

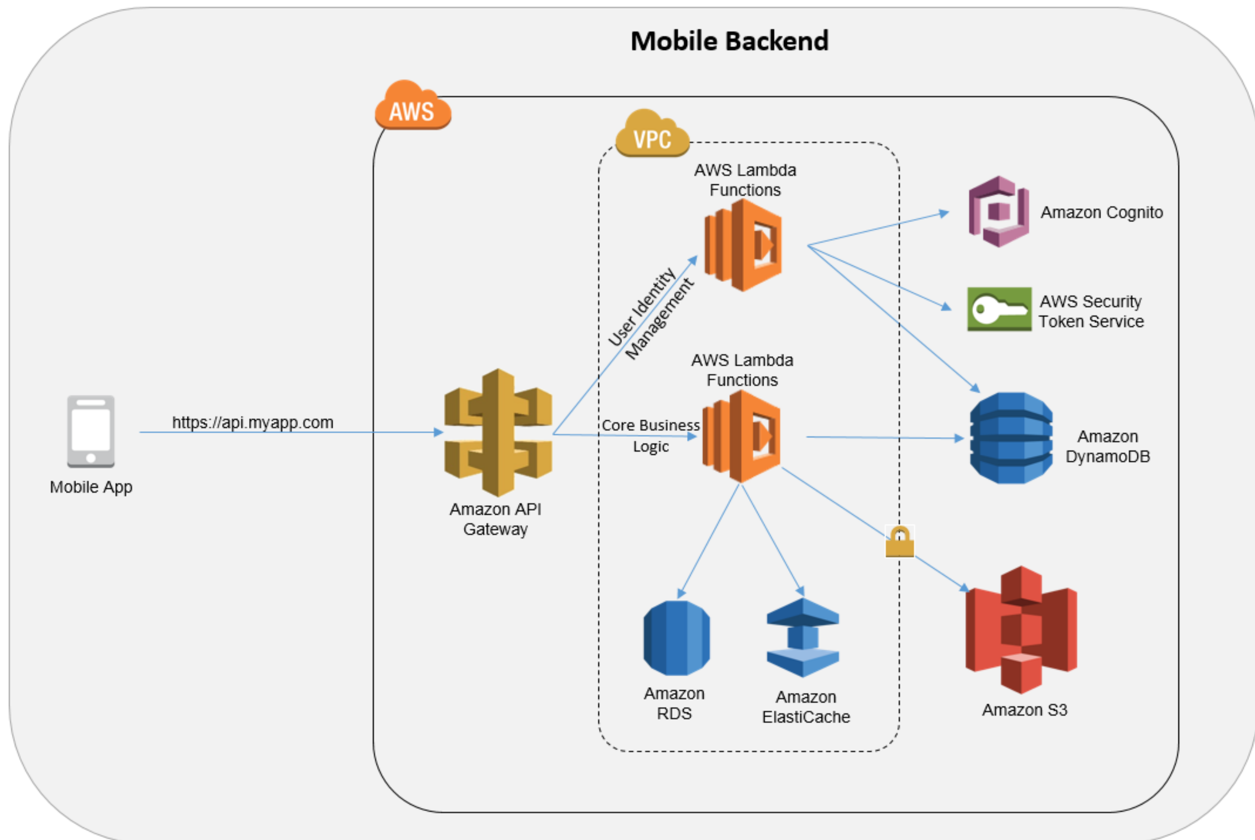

Figure 3: Architectural pattern for mobile back end

- **Presentation Tier:** A mobile application running on each user's smartphone.
- **Logic Tier:** Amazon API Gateway and AWS Lambda. The logic tier is globally distributed by the Amazon CloudFront distribution created as part of each Amazon API Gateway API. A set of Lambda functions can be specific to user/device identity management and authentication, and managed by Amazon Cognito, which provides integration with IAM for temporary user access credentials as well as with popular third party identity providers. Other Lambda functions can define the core business logic for your mobile back end.
- **Data Tier:** The various data storage services can be leveraged as needed; options are discussed earlier in this paper.

Amazon S3 Hosted Website

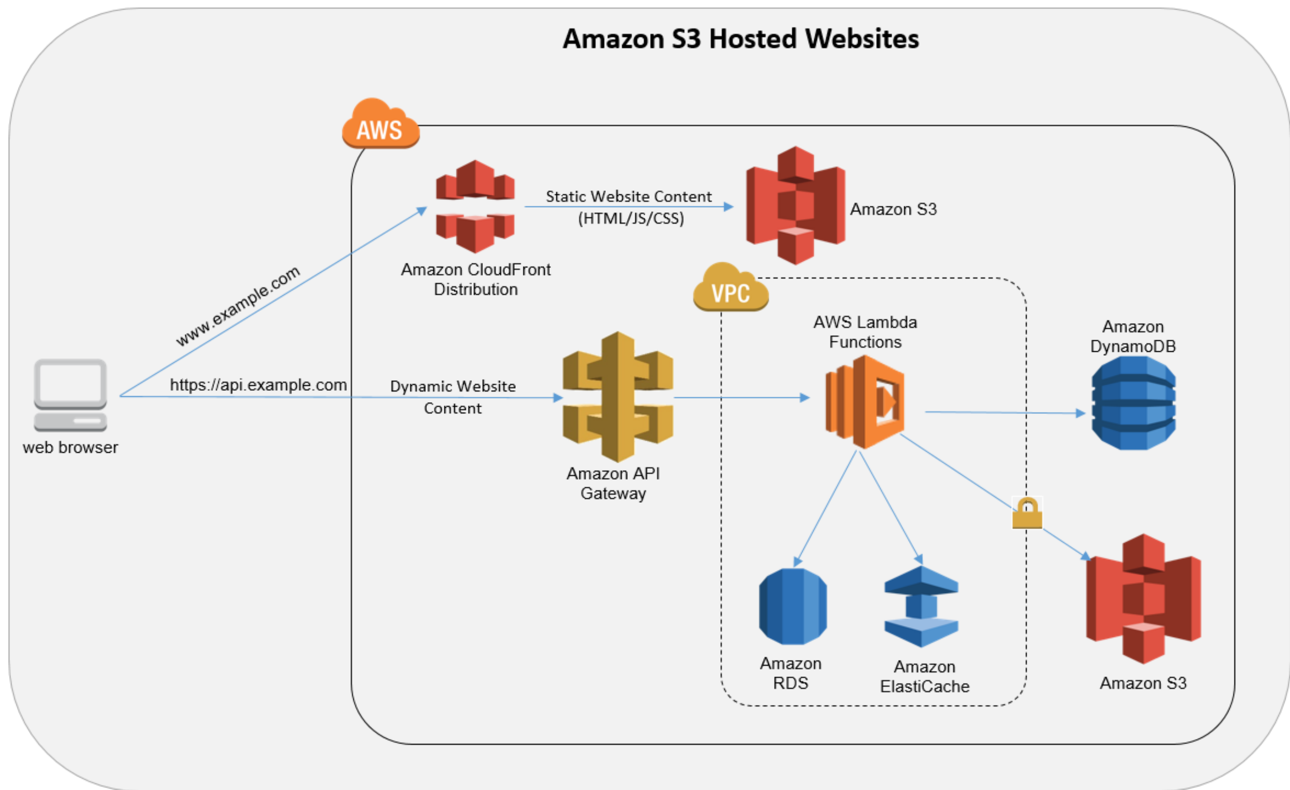

Figure 4: Architectural pattern for a static website hosted on Amazon S3

- **Presentation Tier:** Static website content hosted in Amazon S3, distributed by Amazon CloudFront. Hosting static website content on Amazon S3 is a cost-effective alternative to hosting content on server-based infrastructure. However, for a website to contain rich features, the static content often must integrate with a dynamic back end.
- **Logic Tier:** Amazon API Gateway and AWS Lambda. Static web content hosted in Amazon S3 can directly integrate with Amazon API Gateway, which can be CORS compliant.
- **Data Tier:** The various data storage services can be leveraged as needed. These options are discussed earlier in this paper.

Microservices Environment

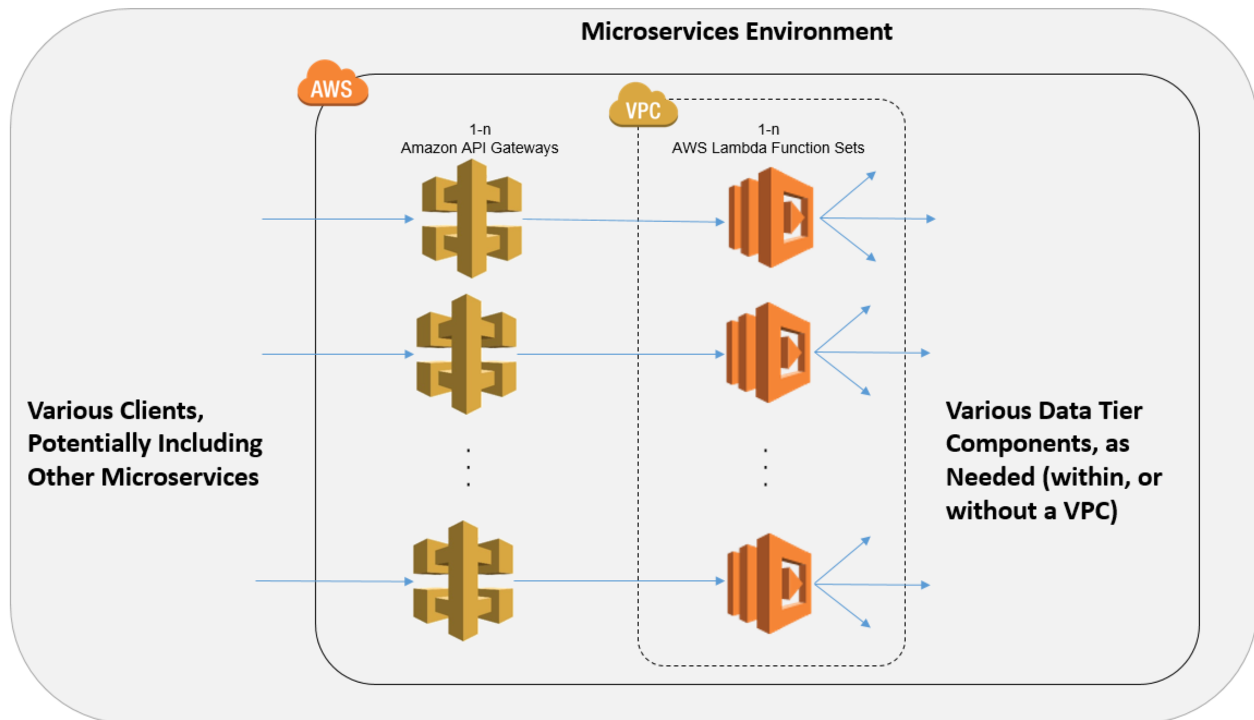

Figure 5: Architectural pattern for a microservices environment

The **Microservices** architecture pattern is not bound to the typical three-tier architecture we have covered in this whitepaper. In a microservices architecture, there is massive decoupling of software components so the benefits of the multi-tier architecture are amplified throughout. An API created with Amazon API Gateway, and functions subsequently executed by AWS Lambda, is all that you need to build a microservice. Your team is free to use these services to decouple and fragment your environment to the level of granularity desired.

In general, a microservices environment can introduce the following difficulties: repeated overhead for creating each new microservice, issues with optimizing server density/utilization, complexity of running multiple versions of multiple microservices simultaneously, and proliferation of client-side code requirements to integrate with many separate services.

However, when you create microservices using the AWS serverless pattern these problems become simpler to solve and, in some cases, simply outright disappear. The AWS microservices pattern lower the barrier for the creation of each subsequent microservice (Amazon API Gateway even allows for the cloning of existing APIs). Optimizing server utilization is no longer relevant with this pattern. Both API Gateway and Lambda enable simple versioning capabilities. Finally, Amazon API Gateway provides programmatically generated client SDKs in a number of popular languages to reduce integration overhead.

Conclusion

The multi-tier architecture pattern encourages the best practice of creating application components that are easy to maintain, decoupled, and scalable. When you create a logic tier where integration occurs via Amazon API Gateway and computation occurs within AWS Lambda, you are on your way to realizing those goals while reducing the amount of effort to achieve them. Together, these services provide a HTTPS API front end for your clients and a secure environment within your VPC to execute business logic. This allows you to take advantage of many popular scenarios in which you can use these managed services instead of managing typical server-based infrastructure yourself.

Contributors

The following individuals and organizations contributed to this document:

Andrew Baird, AWS Solutions Architect

Stefano Buliani, Senior Product Manager, Tech, AWS Mobile

Vyom Nagrani, Senior Product Manager, AWS Mobile

Ajay Nair, Senior Product Manager, AWS Mobile

amazon webservices

Notes

[1] http://aws.amazon.com/api-gateway/

[2] http://aws.amazon.com/lambda/

[3] https://aws.amazon.com/vpc/

[4] https://aws.amazon.com/cloudfront/

[5] https://d0.awsstatic.com/whitepapers/DDoS_White_Paper_June2015.pdf

[6] https://aws.amazon.com/api-gateway/pricing/

[7] http://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-mock-integration.html

[8] http://aws.amazon.com/iam/

[9] http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html

[10] http://docs.aws.amazon.com/lambda/latest/dg/intro-core-components.html#intro-core-components-event-sources

[11] https://aws.amazon.com/s3/

[12] https://aws.amazon.com/kinesis/

[13] https://aws.amazon.com/vpc/

[14] https://aws.amazon.com/rds/

[15] https://aws.amazon.com/elasticache/

[16] https://aws.amazon.com/redshift/

[17] https://aws.amazon.com/ec2/

[18] https://aws.amazon.com/dynamodb/

[19] https://aws.amazon.com/s3/storage-classes/

[20] https://aws.amazon.com/elasticsearch-service/

[21] https://aws.amazon.com/cognito/