**Launch a Linux Virtual Machine - EC2**

* Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.
* EC2 also allows users to build apps to automate scaling according to changing needs and peak periods, and makes it simple to deploy virtual servers and manage storage, lessening the need to invest in hardware and helping streamline development processes.
* EC2 setup involves creating an Amazon Machine Image (AMI), which includes an operating system, apps, and configurations. That AMI is loaded to the Amazon Simple Storage Service (S3), and it’s registered with EC2, at which point users can launch virtual machines as needed.
* Amazon offers different instance types of EC2 for different requirements and budgets, including hourly, reserved, and spot rates.
* A number of benefits and features draw developers to EC2 for cloud computing. Chief among these are:
  + Responsiveness
  + Flexibility in configurations
  + **Integration**
  + **Precise control**
  + Security
  + Cost

**QwikLab: Intro to S3**

* Amazon Simple Storage Service (Amazon S3) is a scalable, high-speed, web-based cloud storage service designed for online backup and archiving of data and applications on Amazon Web Services. Amazon S3 was designed with a minimal feature set and created to make web-scale computing easier for developers.
* Amazon S3 is an object storage service, which differs from block and file cloud storage. Each object is stored as a file with its metadata included and is given an ID number. Applications use this ID number to access an object. Unlike file and block cloud storage, a developer can access an object via a REST API.
* S3 enables customers to upload, store and download practically any file or object that is up to five terabytes (TB) in size, with the largest single upload capped at five gigabytes (GB).
* S3 provides 99.999999999% durability for objects stored in the service and supports multiple security and compliance certifications.
* Amazon S3 comes in three storage classes: S3 Standard, S3 Infrequent Access and Amazon Glacier.
  + S3 Standard is suitable for frequently accessed data that needs to be delivered with low latency and high throughput. S3 Standard targets applications, dynamic websites, content distribution and big data workloads.
  + S3 Infrequent Access offers a lower storage price for data that's needed less often, but that must be quickly accessible. This tier can be used for backups, disaster recovery and long-term data storage.
  + Amazon Glacier is the least expensive storage option in S3, but it is strictly designed for archival storage because it takes longer to access the data. Glacier offers variable retrieval rates that range from minutes to hours.
* A user can also implement lifecycle management policies to curate data and move it to the most appropriate tier over time.
* User data is stored on redundant servers in multiple data centers. S3 uses a simple web-based interface -- the Amazon S3 console -- and encryption for user authentication
* S3 buckets are kept private by default, but an admin can choose to make them publicly accessible. A user can also encrypt data prior to storage.

**Video: Virtualization**

* The Cloud might seem like a modern-day marvel, but in fact it relies on software old and new. That's why now, more than ever, Virtual Machines are used to run legacy software and power the Cloud that we don't think twice about.
* The video explains how virtual machines work and how they run everything from aging software to cutting-edge code.
* OS runs on Ring 0 and Applications on Ring 3.
* Virtualizations runs on Ring 1
* Xen is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently
* Guest virtual machines running on a Xen Project Hypervisor are known as “domains”.
* The hypervisor supports two primary types of virtualization: paravirtualization (PV) and hardware virtualized machine (HVM) also known as “full virtualization”.
* Today’s hardware allow support for hardware virtualized machine (HVM).
* Virtualization allows to use legacy application on modern hardware
* Virtualization allows to allocate ideal resources as required for the Virtual Machine
* Virtualization helps to allocate a new server in a minute in the cloud

**Tutorial: Install a LAMP Web Server on Amazon Linux 2**

* Amazon EC2 provides the virtual application servers, known as instances, to run your web application on the platform you choose. Amazon EC2 allows you to configure and scale your compute capacity easily to meet changing requirements and demand. It is integrated with Amazon’s proven computing environment, allowing you to leverage the AWS suite of services.
* Let's say you decide to host your LAMP website on two Linux t2.micro EC2 instance in the US East region. With an on-demand pricing model, your monthly charge for your virtual machines will be $19.04.
* A LAMP stack that is not highly available would cost less than $1 per month if you qualify for the AWS Free Tier and are within its limits.
* LAMP is an acronym, and these stacks typically consist of the Linux operating system, the Apache HTTP Server, the MySQL relational database management system, and the PHP programming language.
* To install LAMP stack on Amazon Linux, make sure one has SSH access and admin access to the EC2
* Before installing LAMP, it is always recommended to update the underlying install repos and applications.
* In the given tutorial, Apache 2.4, PHP 7.2, Maria DB 10.2 is installed. However, one can install any different combination of versions and alternatives.
* Once installed, the security groups must be updated to allow incoming port connections from the internet to the EC2 instance.
* You must allow the ec2-user account to manipulate files in this directory, you must modify the ownership and permissions of the directory.
* You can test your LAMP server by create a sample phpinfo.php which will display the PHP info. Make use you can set appropriate permissions and the php file is deleted once deleted.
* Make sure you secure your Database
* To access database from http, install phpMyAdmin
* Now that you have a LAMP stack installed, you have many choices for what to do next. Basically, you've installed a platform that will allow you to install most kinds of websites and web software on your server.
* As an immediate next step, you should ensure that connections to your web server are secured, by serving them via HTTPS.

**Compare the performance, functionality, and price when serving web content from S3 versus an EC2 VM**

**EBS is:**

* A block storage (so you need to format it). This means you are able to choose which type of file system you want.
* Block storage stores files in multiple volumes called blocks, which act as separate hard drives; block storage devices are more flexible and offer higher performance than regular file storage.
* As it’s a block storage, you can use Raid 1 (or 0 or 10) with multiple block storages
* It is really fast
* It is relatively cheap
* With the new announcements from Amazon, you can store up to 16TB data per storage on SSD-s.
* You can snapshot an EBS (while it’s still running) for backup reasons
* But it only exists in a particular region. Although you can migrate it to another region, you cannot just access it across regions (only if you share it via the EC2; but that means you have a file server)
* You need an EC2 instance to attach it to
* You can now increase volume size, adjust performance, or change the volume type while the volume is in use. You can continue to use your application while the change takes effect.

**S3 is:**

* An object store (not a file system).
* You can store files and “folders” but can’t have locks, permissions etc like you would with a traditional file system
* This means, by default you can’t just mount S3 and use it as your webserver
* There is no hierarchy of relations between files with object storage — data objects can be distributed across several machines.
* You can access the S3 service from anywhere on the internet.
* But it’s perfect for storing your images and videos for your website
* Great for short term archiving (e.g. a few weeks). It’s good for long term archiving too, but Glacier is more cost efficient.
* Great for storing logs
* You can access the data from every region (extra costs may apply)
* Highly Available, Redundant. Basically data loss is not possible (99.999999999% durability, 99.9 uptime SLA)
* Much cheaper than EBS.
* You can serve the content directly to the internet, you can even have a full (static) website working direct from S3, without an EC2 instance

|  |  |  |
| --- | --- | --- |
|  | **AWS S3** | **AWS EBS** |
| Performance | Support 100 PUT/LIST/DELETE Requests per second  Scalable upto 300 request per second | Provisioned IOPS delivers upto 10000 input/output operations per second |
| Cost | First 50TB/Month - $0.0245 per GB  Next 450TB/Month - $0.0235 per GB  Over 500TB/Month - $0.0225 per GB | Around 4-5 times higher than S3  Varies as per Region and type of EBS provisioned |
| Availability and Accessibility | Upto 99.9999999 percent  Accessible by APIs | 99.99 percent  Accessible by AWS EBS |
| Access Control | Access based on IAM, bucket policies and user policies | Access based on Secuirty Groups and IAM |
| Storage Limit | No limit on number of objects  Each object restricted to 5TB | Each EBS disk size is restricted upto 16TB |

**QwikLab: Intro to DynamoDB**

* DynamoDB is a powerful, fully managed, low latency, NoSQL database service provided by Amazon.
* DynamoDB allows you to pay for dedicated throughput, with predictable performance for "any level of request traffic".
* Scalability is handled for you, and data is replicated across multiple availability zones automatically.
* Amazon DynamoDB can store the documents in JSON format. It stores the data in tabular format encapsulating items in each table.
* Amazon DynamoDB can also integrate with non-AWS services and can be monitored through CloudWatch, and the control access can be maintained through IAM. With DynamoDB, being a key-value store as well, you can only perform operations on indexed fields.
* There are two types of Primary Keys:
  + Partition Key
  + Partition and Sort Key (Aka Composite Primary Key)
* Beside Primary Keys there are Secondary Indexes. There are Global Secondary Indexes, which contain a Partition and a Sort Key and Local Secondary Indexes which have the same partition the Primary Key has, but a different Sort Key. When using a Partition and Sort Key it's more likely you get a higher throughput due to higher distribution of your data over the partitions.
* There is different type of operations one can do in DynamoDB
  + Put- and GetItem operations need the full Table Primary Key, which can consist of two fields, if it is a Composite Key. You cannot use Index Primary Keys (aka Secondary Indexes) for those operations.
  + Query is more flexible then GetItem. You can query your database by the Table or Index Partition Key. You can set KeyConditionExpression to define a subset of all data you want to receive.
  + Scan is the most flexible read operation. Your database does not use the advantages of hashing and indexing, but filters your data during an expensive full table scan based on a FilterExpression.
  + To use Update- and DeleteItem you need a Table Primary Key as well.
  + There are more operations for doing batch operations
* DynamoDB is a NoSQL database and is schemaless. With the Primary Key being an exception, you have to specify data types. DynamoDB supports scalar types, document types and set types.

**Deploy a Node.js Web App**

* This project allows one to deploy a high-availability Node.js web app using AWS Elastic Beanstalk and Amazon DynamoDB.
* IAM takes care of the authorization from outside your code
* You'll create a DynamoDB table that is external to the AWS Elastic Beanstalk environment, and configure the application to use this external table instead of creating one in the environment.
* Following is the step to deploy Node.js Web App
  + Launch an Elastic Beanstalk Environment
  + Add Permissions to Your Environment's Instances
  + Deploy the Sample Application
  + Create a DynamoDB Table
  + Update the Application's Configuration Files
  + Configure Your Environment for High Availability
* To make the Node.js app ready to scale:
  + Decouple application instances from DB
  + Be stateless
  + Stateless authentication with JWT
  + Storage on S3
  + Properly configure WebSockets
* The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

**QwikLab: Intro to AWS Lambda**

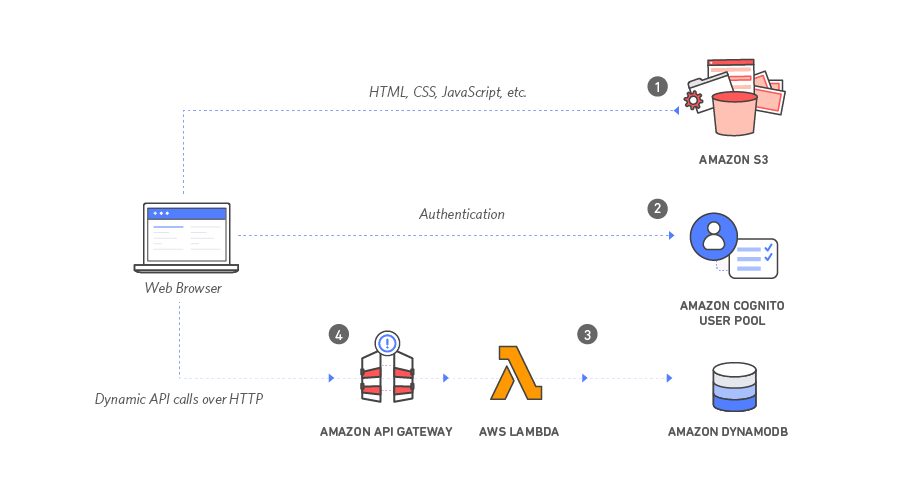
* This lab will give you the basic understanding of AWS Lambda.
* It will demonstrate the basic steps required to get started to create and deploy a Lambda function in an event-driven environment.
* AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running. AWS Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.
* You upload your code to AWS Lambda or write code in Lambda’s editor
* You setup your code to trigger from other AWS services, HTTP Endpoints or in-app activity
* This is when the Lambda runs your code using the compute resources what is needed
* AWS Lambda automatically monitors Lambda functions on your behalf, reporting real-time metrics through Amazon CloudWatch, including total requests, account-level and function-level concurrency usage, latency, error rates, and throttled requests. You can view statistics for each of your Lambda functions via the Amazon CloudWatch console or through the AWS Lambda console.
* AWS Lambda functions can be configured to run up to 15 minutes per execution. You can set the timeout to any value between 1 second and 15 minutes.
* AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a Runtime API which allows you to use any additional programming languages to author your functions.
* You can easily list, delete, update, and monitor your Lambda functions using the dashboard in the AWS Lambda console. You can also use the AWS CLI and AWS SDK to manage your Lambda functions.
* Each AWS Lambda function has a single, current version of the code. Clients of your Lambda function can call a specific version or get the latest implementation.
* Events are passed to a Lambda function as an event input parameter. For event sources where events arrive in batches, such as Amazon SQS, Amazon Kinesis, and Amazon DynamoDB Streams, the event parameter may contain multiple events in a single call, based on the batch size you request.

**QwikLab: Intro to Amazon API Gateway**

* Amazon API Gateway is an AWS service that enables developers to create, publish, maintain, monitor, and secure APIs at any scale
* This lab invokes the knowledge of previous lab – Intro to Lambda by creating a Lambda function.
* In this lab, you create a simple FAQ micro-service. The micro-service will return a JSON object containing a random question and answer pair using an Amazon API Gateway endpoint that invokes an AWS Lambda function.
* Amazon API Gateway and AWS Lambda provide the perfect combination of web services to effortlessly build, deliver and maintain a suite of microservices that can be the foundation of complex software systems.
* In this lab, you will learn how to develop, deploy and debug a simple microservice that represents one part of a much larger system. It will consist of two pieces: the RESTful API and the function that is executed when a user hits the endpoint.
* A microservice using Amazon API Gateway consists of a defined resource and associated methods (GET, POST, PUT, etc.) in API Gateway as well as the backend target.
* Logging is an essential part of building backends and it is no different for a serverless API. In the serverless environment, we have lesser control over the underlying infrastructure, logging is the only way to acquire knowledge on how the application is performing. Amazon CloudWatch is a monitoring service to help you collect and track metrics for your resources. Using the analogy of server logs and application logs, you can roughly think of the API Gateway logs as your server logs and Lambda logs as your application logs.
* To enable API Gateway CloudWatch Logs is a two step process. First, we need to create an IAM role that allows API Gateway to write logs in CloudWatch. Then we need to turn on logging for our API project.
* Lambda CloudWatch logs are enabled by default. It tracks the duration and max memory usage for each execution. You can write additional information to CloudWatch via console.log
* CloudWatch groups log entries into Log Groups and then further into Log Streams. Log Groups and Log Streams can mean different things for different AWS services. For API Gateway, when logging is first enabled in an API project’s stage, API Gateway creates 1 log group for the stage, and 300 log streams in the group ready to store log entries. API Gateway picks one of these streams when there is an incoming request.
* For Lambda, each function has its own log group. And the log stream rotates if a new version of the Lambda function has been deployed or if it has been idle for some time.

**AWS Tutorial: Build a Serverless Web Application**

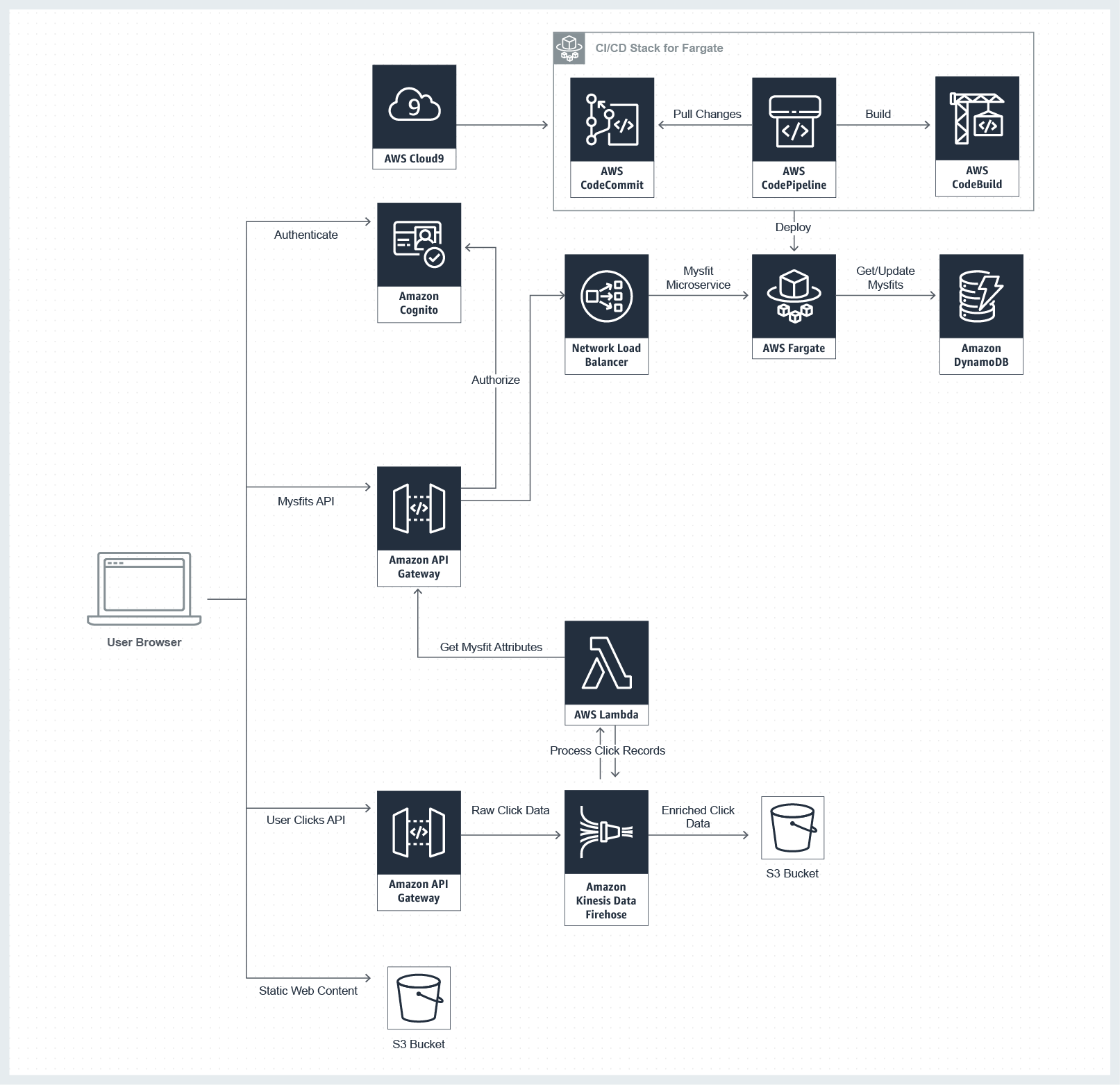
* This Lab uses following AWS Services - AWS Lambda, Amazon API Gateway, Amazon S3, Amazon DynamoDB, and Amazon Cognito
* Building a serverless application allows you to focus on your application code instead of managing and operating infrastructure. You do not have to think about provisioning or configuring servers since AWS handles all of this for you. This reduces your infrastructure management burden and helps you get faster time-to-market.
* Following are the advantages of Serverless computing:
  + Cheaper – Pay as you go
  + Scalable – Automatic Scaling
  + Focus on the application rather than underlying resources.



* Amazon S3 hosts static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser. Your end users will then access your site using the public website URL exposed by Amazon S3. You don't need to run any web servers or use other services in order to make your site available. This is done by Files being uploaded in the S3 bucket and enabling Website Hosting. Also Change the Bucket Policy to allow Public Reads which is off by default.
* Amazon Cognito provides user management and authentication functions to secure the backend API. You'll create an Amazon Cognito user pool to manage your users' accounts. You'll deploy pages that enable customers to register as a new user, verify their email address, and sign into the site. You can use Cognito User Pools to add sign-up and sign-in functionality to your application or use Cognito Identity Pools to authenticate users through social identity providers such as Facebook, Twitter, or Amazon, with SAML identity solutions, or by using your own identity system. For this module you'll use a user pool as the backend for the provided registration and sign-in pages. Once Pool is created, Associate it with the application. All new users signed in through the Webpage will have an entry in the Cognito Pool
* Amazon DynamoDB provides a persistence layer where data can be stored by the API's Lambda function. One can use AWS Lambda and Amazon DynamoDB to build a backend process for handling requests for your web application. A Lambda function will be invoked each time a user. The function will fetch data and record the request from a DynamoDB table and then respond to the front-end application.
* JavaScript executed in the browser sends and receives data from a public backend API built using Lambda and API Gateway. One can use Amazon API Gateway to expose the Lambda function you built in the previous module as a RESTful API. This API will be accessible on the public Internet. It will be secured using the Amazon Cognito user pool you created in the previous module.
* This Lab merely skims the surface of what serverless computing is about. It gives you a real fundamental understanding of what serverless computing is about so that you have a foundation of knowledge that you can build on.

**Bring it together: AWS Tutorial: Build a Modern Web Application**

* You will build a sample website called Mythical Mysfits that enables visitors to adopt a fantasy creature (mysfit) as pet.
* This Lab emphasis on creating a modern well-architected architecture. Following is application architecture diagrams provide a structural representation of the services that make up Mythical Mysfits and how these services interact with each other:



* Following is the implementation of the **Mythical Mysfits**
  + Create Static Website Build a static website, using Amazon Simple Storage Service (S3) that serves static content (images, static text, etc.) for your website.
  + Build Dynamic Website Host your application logic on a web server, using an API backend microservice deployed as a container through AWS Fargate.
  + Store Mysfit Data Externalize all of the mysfit data and persist it with a managed NoSQL database provided by Amazon DynamoDB.
  + Add User Registration Enable users to registration, authentication, and authorization so that Mythical Mysfits visitors can like and adopt myfits, enabled through AWS API Gateway and its integration with Amazon Cognito.
  + Capture User Clicks Capture user behavior with a clickstream analysis microservice that will record and analyze clicks on the website using AWS Lambda and Amazon Kinesis Firehose.
* AWS provides cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser called AWS Cloud9. We create a new environment with default minimal settings in Cloud9 for your application and can clone existing github projects through CLI. Cloud9 is helpful in following ways:
  + Code with Just a Browser
  + Code Together in Real Time
  + Build Serverless Applications with Ease
  + Direct Terminal Access to AWS
  + Start New Projects Quickly
* All Static contents of our Applications will be served through AWS S3. Hence, we create a S3 bucket with appropriate Read permissions and enabling Website Hosting.
* For Dynamic Content, we can use AWS Lambda. However, Lambda has some disadvantages specially to run long running process. Hence, we use AWS Fargate. AWS Fargate is a compute engine for Amazon ECS that allows you to run containers without having to manage servers or clusters. Thus, it is highly useful for deploying modern applications. In our application, AWS Fargate creates a PHP container behind AWS Load Balancer.
* However, this will be provisioned not through the console/CLI but through AWS CloudFormation. AWS CloudFormation allows one to script infrastructure as a code. AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment. CloudFormation allows you to use a simple text file to model and provision, in an automated and secure manner, all the resources needed for your applications across all regions and accounts. This can be automated and deploy with versioning just as a code.
* The first code deployment can be manual in AWS Fargate. However, future deployment has to be automated. This is done through AWS Code Services. It allows Continuous Integration and Continuous Delivery of our application. A stack needs to be created which will manage the CI/CD.
* The Backend of the applications uses AWS DnamoDB which is a NoSQL database and is schemaless. The DynamoDB table can be accessed by AWS Cloud9 CLI terminal. This allows one to execute the queries on AWS DynamoDB table.
* Once the backend is complete and integrated, final commit needs to be pushed through the AWS Code Services into AWS Fargate container and static contents in AWS S3 bucket create earlier.
* To enable registration and authentication of website users, we will create a User Pool in AWS Cognito - a fully managed user identity management service for the reasons described in earlier modules.
* The register Users can acess authorized section through REST API with AWS Amazon API Gateway. Amazon API Gateway is also a managed service, and provides commonly required REST API capabilities out of the box like SSL termination, request authorization, throttling, API stages and versioning, and much more.
* Thus we have fully running applications running on AWS.
* To capture User behavior, we need to understand how users are interacting with the application.
* Kinesis Firehose enables all of the records received by the stream to be automatically delivered to a serverless function created with AWS Lambda .
* AWS Kinesis Firehose provides a service API just like other AWS services, and in this case we are using its PutRecord operation to put user click event records into the delivery stream. We use Amazon API Gateway to create an AWS Service Proxy to the PutRecord API of Kinesis Firehose. This allows us to craft our own public RESTful endpoint that does not require AWS credential management on the frontend for requests.
* Lambda is well-suited for these applications because it is integrated with data stores such as Kinesis Data Streams and Data Firehose, S3, CloudWatch Events, CloudWatch Logs, and DynamoDB. Once the Lambda function is created and the Kinesis Firehose delivery stream is configured as an event source for the function, the delivery stream will automatically deliver click records as events to code function we've created, receive the responses that our code returns, and deliver the updated records to the configured Amazon S3 bucket.