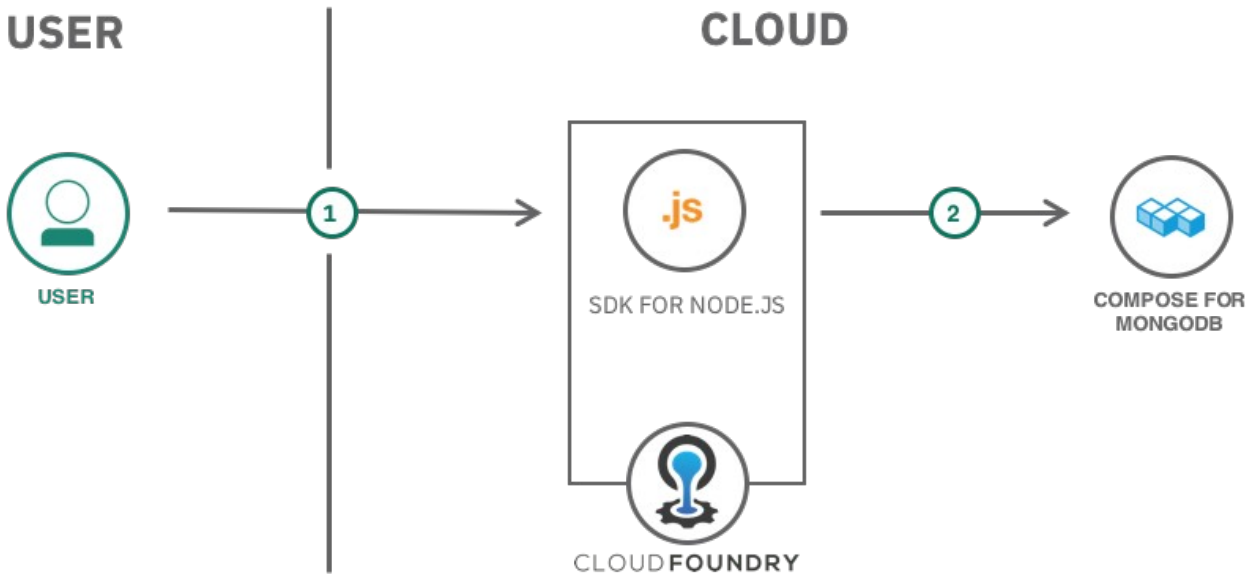
MEAN:-
MEAN stands for MongoDB, Express.js, AngularJS and Node.js. MEAN is an end-to-end JavaScript stack largely used for cloud-ready applications. Understanding why you might use it, identifying examples of when to employ it and diving deeper into the individual components can help you maximize the value of MEAN for software development.



An example architecture for a Node.js runtime with MongoDB on a MEAN stack

Why use MEAN?

MEAN is an open source web stack that is mainly used to create cloud-hosted applications. MEAN stack applications are flexible, scalable and extensible, making them the perfect candidate for cloud hosting. The stack includes its own web server so it can be deployed easily, and the database can be scaled on demand to accommodate temporary usage spikes. A MEAN application enters the world optimized to take advantage of all the cost savings and performance improvements of the cloud.

JavaScript has long been a popular language for front-end web development — it's flexible, dynamic and easy to use. But it has been an option only for back-end and database development for a few years, allowing developers to create applications using end-to-end JavaScript. Because every part of MEAN uses the same language, you can streamline your development teams. MEAN removes the need to hire different specialists to develop each part of an application. Instead, you can use a single pool of JavaScript developers to work adaptively, where and when needed. Standardizing on JavaScript also provides an opportunity to reuse code across the entire application, reducing unnecessary reinvention.

When should you use MEAN?

While the MEAN stack isn't perfect for every application, there are many uses where it excels. It's a strong choice for developing cloud-native applications because of its scalability and its ability to manage concurrent users. The AngularJS front-end framework also makes it ideal for developing single-page applications (SPAs) that serve all information and functionality on a single page. Here are a few examples for using MEAN:

- Calendars
- Expense tracking
- News aggregation sites
- Mapping and location finding

What does each component contribute to MEAN?

MongoDB:-

MongoDB is an open source, NoSQL database designed for cloud applications. It uses object-oriented organization instead of a relational model.

In the MEAN stack, MongoDB stores the application's data. Because both the application and the database use JavaScript, there's no need to translate the object as it journeys from the application to the database and back. The application can push and pull objects between the back end and the database without missing a beat.

MongoDB is touted for its scalability in both storage and performance. You can add fields to the database without reloading the entire table; and MongoDB is well known for its ability to manage large amounts of data without compromising on data access. With just a few clicks, you can expand the resources available to your database, making it perfect for applications with occasional periods of increased activity.

Express:-

Express is a web application framework for Node.js. It balances ease of use and a full feature set.

Forming the back end of the MEAN stack, Express handles all the interactions between the front end and the database, ensuring a smooth transfer of data to the end

user. It’s designed to be used with Node.js and so continues the consistent use of JavaScript throughout the stack.

Express is minimalist — it’s designed to efficiently handle processes without cluttering your application. But don’t confuse minimalist with featureless. Express offers excellent error handling and templating functionality to aid your development.

Express can also protect you from yourself because it uses the CommonJS module standard to prevent inadvertent overwriting of variables within the shared namespace. You can’t accidentally redefine a variable that you previously created. This enforcement of JavaScript closures can help prevent a time-consuming and costly error.

AngularJS:-

AngularJS — Google’s JavaScript front-end framework — isn’t the only front-end framework in use, but it’s exceedingly popular. It is effectively the default for front-end JavaScript development. If you’re developing a web application in JavaScript, you’re using AngularJS.

The MEAN stack includes AngularJS to help developers build the user-facing side of the application. Because the back end, front end and database are all built on JavaScript, there’s a smooth flow of information between all parts of your application.

AngularJS didn’t become the most popular JavaScript front-end framework by mistake. Its ability to simultaneously develop for desktop and mobile use, its well-tuned performance and its easy-to-use templates make it the ideal front end to build cloud-native applications.

Node.js:-

Node.js is an open source JavaScript framework that uses asynchronous events to process multiple connections simultaneously. It is an ideal framework for a cloud-based application, as it can effortlessly scale requests on demand. You’re likely to find Node.js behind most well-known web presences.

Node.js is the backbone of the MEAN stack. Express is purpose-built to work on top of Node.js, and AngularJS connects seamlessly to Node.js for fast data serving. Node.js comes complete with an integrated web server, making it easy to deploy your MongoDB database and application to the cloud.

The greatest strength of Node.js is its scalability. Cloud applications are best when they can respond quickly to usage spikes. What good is virtually unlimited processing power if it's only available after your users time out? By expanding your resources as they're needed, you're able to serve more users while the framework's single-thread architecture allows the application to effectively provide a smooth user experience across numerous connections. Node.js can support as many as a million simultaneous connections.

Remember, Node.js works best with many low-resource requests as opposed to resource-intensive requests. While a single thread protects against process deadlocks, it's not immune to a large process freezing the system for all clients.

Node.js - Utility Modules:-

1. OS Module:-Provides basic operating-system related utility functions.
2. Path Module :-Provides utilities for handling and transforming file paths.
3. Net Module :-Provides both servers and clients as streams. Acts as a network wrapper.
4. DNS Module :-Provides functions to do actual DNS lookup as well as to use underlying operating system name resolution functionalities.
5. Domain Module :-Provides ways to handle multiple different I/O operations as a single group.

Installation of Node.js and npm on Ubuntu 18.04:-

Start by updating the packages list by typing:
```
$sudo apt update
```

Install `nodejs` using the `apt` package manager:
```
$sudo apt install nodejs npm
```

To verify the installation execute the following command:
```
$nodejs –version
```

To be able to download npm packages, you also need to install `npm`, the Node.js package manager. To do do type:

```
$sudo apt install npm
```

Connect Node.js to MySQL:-

Install MySQL Driver:-

To access a MySQL database with Node.js, you need a MySQL driver.

```
$npm install mysql
```

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "wadah",
password: "wadah"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
});
```

run command to check connections

```
$ node demo_db_connection.js
$ Connected!
```

Smiple Node.js with MySql DB Application:-

simple example which passes two values using HTML FORM POST method. And use **process_get** router inside server.js to handle this input.

index.html page:-

```
<html>
<body>
<form action="http://127.0.0.1:6020/insert_user" style="border:1px solid #ccc"
method="post">
<div>
<h1>Sign Up</h1>
<p>Please fill in this form to create an account.</p>
<hr>
<label for="Name"><b>Name</b></label>
<input type="text" placeholder="Enter Name" id="name" name="name" required>
<label for="email"><b>Email</b></label>
<input type="text" placeholder="Enter Email" name="email" required>
<label for="psw"><b>Password</b></label>
<input type="password" placeholder="Enter Password" name="password" required>
<hr>
<label for="address"><b>Address</b></label>
<input type="text" placeholder="House no and Street" name="address" required>
<label for="mobile"><b>Phone</b></label>
```

```
<input type="number" placeholder="Enter Phone Number" name="phone" required>

<div class="clearfix">
<button type="submit" >Sign Up</button>
</div>
</div>
</form>

</body>
</html>
```

insert_user.js page:-

```
var express = require("express");
var bodyParser = require("body-parser");
var app = express();
var mysql = require('mysql');


app.use( express.static( "public" ) );
app.use( bodyParser.json() ); // to support JSON-encoded bodies
app.use(bodyParser.urlencoded({ // to support URL-encoded bodies
extended: true
}));


var connection = mysql.createConnection({
host: "localhost",
user:"wadah",
password:"wadah",
database:"users"
});

app.use(bodyParser.urlencoded({ extended: true }));
app.set("view engine", "ejs");

app.post("/insert_user", function(req, res) {

var sql = "INSERT INTO user_info SET ? ";
var post ={name :req.body.name , email: req.body.email, password: req.body.password
, address: req.body.address, phone: req.body.phone };

console.log(post);
var query = connection.query(sql, post, function (err, data) {
if (err) {
console.log("Err inserting");

} else {
// successfully inserted into db
console.log("Inserted Successfully ");
}
});
```

```
console.log(query.sql);


});
app.listen(6020, function() {
console.log('Server running at http://127.0.0.1:6019/');
});
```

Sources:-

https://www.w3schools.com/nodejs/nodejs_mysql.asp

–

https://www.ibm.com/cloud/learn/mean-stack-explained

–

https://www.tutorialspoint.com/nodejs