



Component Stacks for Enterprise Applications

Panos Louridas

Mobile enterprise application development is transitioning from SQL-based traditional servers toward JavaScript for server and client. MEAN as a component stack is replacing LAMP, with a growing number of adopters and Web applications. As the world turns faster, we'll see increasingly flexible and evolving component stacks. In this instalment of the Software Technology department, Panos Louridas gives an introduction to LAMP and MEAN and shows how to practically use MEAN. I look forward to hearing from both readers and prospective column authors. —*Christof Ebert*



WE INTERACT WITH Web applications to carry out most of our everyday tasks. From paying taxes to booking vacations, our basic means of interaction is a Web browser that communicates with an application running somewhere on a server. Consequently, Web development is a primary source of employment for programmers. Programming languages such as Ruby, PHP, and JavaScript, which are used primarily for Web development, feature prominently in the top 10 popular languages in yearly rankings, such as those issued by *IEEE Spectrum*.¹

Until relatively recently, the tools used to develop Web applications followed a well-established architecture called the LAMP stack. Recently, the MEAN stack has taken the Web developer world by storm and is replacing LAMP.

LAMP

The LAMP stack has been around for many years and is the established way to

develop Web applications. Originally it stood for “Linux, Apache, MySQL, and PHP,” but its meaning is now broader. Basically, developers build Web applications using specific software components at different levels. At the base level, the OS of choice is Linux. With the OS in place, a Web application needs a webserver. Traditionally, this has been the Apache webserver. The data the Web application uses is stored in a relational database; the most popular one is MySQL. The application itself is written in a programming language offering good support for Web programming; this was initially PHP.

Things have changed considerably over time. PHP is popular for Web development, but many programmers prefer Java or Python, which are still compatible with the LAMP stack. And it would be ludicrous to say that LAMP doesn't cover Ruby on Rails. In the same vein, MySQL remains the most popular database, but PostgreSQL powers many Web

THE RISE OF MEAN



In a way, MEAN (for an explanation of this acronym, see the main article) is an offshoot of JavaScript's popularity as a general-purpose programming language, instead of its more traditional role as a language embedded in webpages. This became possible with Node.js, a framework for building server-side JavaScript applications. Programmers can now use JavaScript for the same tasks for which they've been using Java, Python, or Ruby on the server.

That's a challenge and an opportunity. JavaScript uses a specific programming model based on events. To make programs that perform well, programmers must be careful to use asynchronous programming patterns, which often requires a change in thinking. At the same time, Web developers already know JavaScript because they use it to add interactive components on their webpages. Instead of having to work with two programming languages in a single Web application—for example, Python on the server-side components and JavaScript on the front-end, browser-side components—developers can program everything in the same language.

Modern Web applications might deviate somewhat from that cycle. The webpages might include parts that are updated dynamically, through asynchronous JavaScript (Ajax) calls, without requiring the user to reload or navigate to a new page. However, the basic vehicle of interaction in the LAMP stack is through HTTP request and response pairs. This principle is relinquished in MEAN, and a new Web-programming paradigm emerges.

MEAN

Like LAMP, MEAN is an acronym but comprises more than the acronym components. M is for MongoDB, a popular NoSQL database that powers MEAN applications. E is for Express, a Web application framework built on Node.js. A is for AngularJS, a JavaScript framework for endowing Web applications with modern interactive features. N is for Node.js, the basic computational engine for MEAN applications. The MEAN stack is a server-side JavaScript execution environment, letting developers write generic JavaScript applications that are executed by Node.js, and not just traditional Web JavaScript code that executes in a browser. (For a brief look at the rise of MEAN, see the related sidebar.)

Node.js is pretty low-level; although you can write webserver applications directly on it, it's much better to rely on frameworks that let you access higher-level abstractions. Express is such a framework, letting you focus more on the logic of your applications than on the nitty-gritty.

MEAN asks developers to stop relying on relational databases and make the jump to the NoSQL world. That doesn't mean you can't use relational databases alongside the MEAN stack. NoSQL databases,

applications. And although nobody can doubt the Apache webserver's importance, the nginx webserver's popularity has been increasing. With the LAMP acronym blurred, we had better find a better way to identify what LAMP development is.

Defining LAMP Development

LAMP has four main components. First, the OS is open source—in most cases, a Linux distribution, although nobody would say that a Web application running on FreeBSD is automatically denied the LAMP moniker.

Second, the Web application is based on user interaction with HTML pages that, after being prepared, are sent by a webserver in response to HTTP protocol requests.

Third, the application needs to keep and process some data. The LAMP stack assumes that the data can and should be described using the relational-database model. The actual database of choice isn't of

much consequence; what's important is that it should be relational.

Fourth, the application must present views that depend on the user's input. That means that not all the HTML pages the webserver returns can be static. Most of the HTML pages the user sees are prepared dynamically, on the basis of the user's input. The chosen programming language—for example, PHP, Python, Ruby, Java, or Perl—takes that input and presents a new HTML page on the fly that the webserver sends back to the user.

Deviating from Tradition

Traditionally, a MEAP (mobile enterprise application platform) application works with the cycle of HTTP request, process, and HTTP response. That is, the user, through the browser, interacts with an HTML page and issues a request to the server. The server processes the request, prepares a new HTML page, and sends it to the user.

though, are a better fit, especially MongoDB, which uses JavaScript as its primary interface. In this way you can use the same language for front-end interaction programming, server-side logic processing, and database access and manipulation.

MongoDB organizes data into *collections* instead of tables. A collection contains *documents*, which are key-value pairs. A value can be a key-value pair itself, so that documents can represent complex nested structures. Documents are essentially JSON (JavaScript Object Notation) objects, a JavaScript subset that can describe nested data structures. MongoDB converts the JSON objects to BSON (Binary JSON) format and places them in persistent storage.

MongoDB collections are schemaless; that is, the documents don't adhere to a predefined schema (as do rows in an SQL table). A collection might contain objects with different structures. That makes MongoDB a good choice for storing data that might have a changing structure and that can't be converted to standard relational formats.

The schemaless nature of MongoDB databases has several consequences. MongoDB collections tend to eat up a lot of space. In a typical relational database, the data type definitions are stored only once in the database dictionary. In a schemaless database, you must specify the structure of each document in the collection, even if all documents have a similar structure. SQL databases have benefitted from years of research and practice in query optimization and execution, nothing of which transfers to schemaless databases. In practice, most queries that are practical in schemaless databases are simple key-value lookups. If you want to run queries that combine

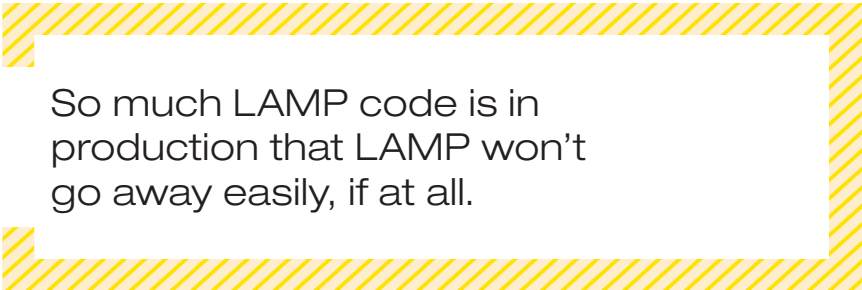
different documents or calculate aggregate values, you might find that performance grinds down to a halt. Moreover, MongoDB doesn't offer transactions like the traditional transactional models of relational databases—although other NoSQL databases offer more transaction support.

At the same time, if relatively simple queries will meet your data needs, MongoDB can be blazingly fast. It can distribute data among different MongoDB instances (this process is called sharding). This makes it well suited for very big amounts of data (“mongo” comes from “humongous”) that can be stored in a distributed fashion.

AngularJS is a fully fledged development framework for webpage applications, structuring them along the Model-View-Controller design pattern. With AngularJS, developers can describe how data from the server links to events and GUI components on the webpage and how events and input from the user alter the GUI components' state. AngularJS extends HTML with a template

application (the HTML and templates for everything the user will see and all associated JavaScript) is downloaded at once. Then, the pages the user sees change continuously depending on the user events, without any page reloading from the server. That entails a significant departure from MEAP applications' HTTP request-response sequence. A MEAN application can issue a single HTTP request and get a bundle of HTML, templates, and JavaScript in a single HTTP response. Then, all the other data is exchanged through REST (Representational State Transfer) calls from the browser to the server. The REST calls typically pass data from the browser to function callbacks running in Node.js and return the results. Data and results are passed as a JSON object payload in the REST calls and responses.

A good way to think about this design is that the front end that the browser gets is an independent client application, which stands on its own and interacts with a Web service application in the server. The client application interacts with the server



So much LAMP code is in production that LAMP won't go away easily, if at all.

syntax that allows easy manipulation of page elements and lets developers specify how, in response to user events, data can travel to and from the server asynchronously via Ajax.

The culmination of this programming model is single-page applications (SPAs). In SPAs, all the Web ap-

plication with an agreed REST API. The client application handles user interaction; the server application handles the business logic.

AngularJS isn't the only choice. The Ember.js development framework has goals similar to those of AngularJS and a strong, devoted

A MEAN EXAMPLE

Consider an online catalog that presents various services (such as cloud storage or online applications available as software as a service) from third-party providers. For each service, the catalog includes basic information and ratings on a set of criteria. The criteria might differ depending on the nature of the service, and it's not clear that a single schema could fit them all. The nature of the service description might also differ among services.

MongoDB is a good fit for this catalog. Most queries are simple reads and some updates; no queries contain complex calculations. You only need to record the information and the criteria ratings for each service, which you can do with JSON (JavaScript Object Notation) documents.

All of the catalog's functionality is through a REST (Representational State Transfer) API, through which JSON documents are updated, written, or read. That API defines a set of actions available through specific URLs (for example, `/service/amazing_cloud_storage/15/put`). The actions are implemented through Express routing rules calling server-side JavaScript; Node.js is the underlying engine.

The user's browser interacts with the REST API, sending and receiving JSON documents. The browser renders the documents as HTML, using a Model-View-Controller framework such as AngularJS. Figure A shows the setup.

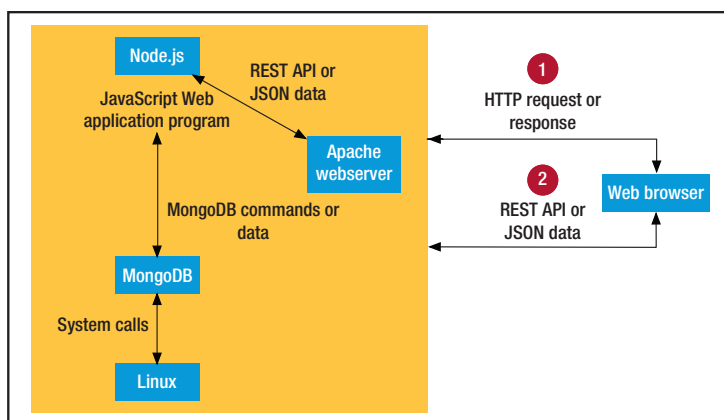


FIGURE A. A typical MEAN setup, for an online catalog that presents various services from third-party providers. REST stands for Representational State Transfer; JSON stands for JavaScript Object Notation.

clear-cut that MEAN applications can't have features from LAMP applications, or vice versa.

The platonic ideal of a MEAN application would have the front end served in one go at the start, but in reality that might not be practical. You might need to strike a balance in which some parts of the application are served in stages through successive HTTP requests and responses. Also, you can't rely on the client side to enforce access controls: with a modicum of JavaScript, someone could bypass all security features of such an application. The controls must be enforced on the server side as well.

It's conceptually appealing to regiment development stacks into those that can rely on NoSQL databases versus those that require SQL databases. However, you might encounter situations in which an otherwise MEAN application needs an SQL database or in which a LAMP application would be much better off with a NoSQL database.

Table 1 compares LAMP and MEAN.

Now, Where?

Where should you go now—down the well-trodden path of LAMP or the trail blazed by MEAN?

This question has no single answer. If a development team has achieved excellence with LAMP, there's no reason to dump it and jump on the MEAN train just to be in fashion. So much LAMP code is in production that LAMP won't go away easily, if at all. MEAN moves fast—perhaps too fast for some people's taste. For example, an announcement from AngularJS's developers that AngularJS 2.x would have breaking changes from AngularJS 1.x raised a wave of protest. At other times, things that developers take

community. It provides different programming abstractions, but the basic idea of having the front end as a client application communicating with a REST API in the back end remains the same.

For an example of the practical

use of MEAN, see the sidebar, "A MEAN Example."

The Dividing Line

Although MEAN and LAMP employ different technologies and tools, the line dividing them often isn't so

TABLE 1

LAMP vs. MEAN.*

	LAMP	MEAN
Maturity	High	Still bleeding-edge in some parts
Adoption	High	High in new projects
Community	Very large	Large, very enthusiastic
Ecosystem	Huge (many programming languages, frameworks, and databases)	Homogeneous (one programming language, few frameworks and databases)
Availability	Both open- and closed-source solutions	Open source solutions
Hip factor	Low (the traditional way to do things)	High (the new way to do things)
Programming paradigm	Well established	Requires adopting new thinking (asynchronous, event-based programming)
Support	Good, from both established commercial vendors and online communities	Commercial support mostly through startups. Good online communities fill the gap.
Performance	Depends on the framework. Java-based frameworks are usually faster than those based on scripted languages. LAMP has plenty of experience with robust, performing enterprise applications.	Node.js is very fast. However, it's not clear whether this is enough to support high-performance enterprise applications.
Developer productivity	Depends on the framework. For instance, Ruby on Rails has been developed with that in mind.	Can be high if the developers are well versed in the JavaScript mind-set. Otherwise, it can be poor.
Security	Depends on the framework. Thanks to accumulated experience, a set of secure practices exists.	Must be considered on both the server and client sides. Users can always subvert the client-side JavaScript.
Usefulness for embedded systems and the Internet of Things	Depends on the framework. A full MEAN stack isn't designed to run on an embedded device, so developers must use MEAN technologies with small footprints.	Although it's too early to say with certainty, Node.js seems to be a good fit for embedded devices running Linux.

* For an explanation of these acronyms, see the main article.

for granted in LAMP, such as well-established interfaces for talking to SQL databases, don't yet exist in MEAN—and the argument that you don't need SQL in MEAN doesn't hold much water. That said, once you've seen what you can do with AngularJS or Ember.js, you won't likely want to go back.

As I indicated, though, there's no reason why the twain shall not meet. You can combine very well an SQL database and a Python development framework, or Ruby on Rails, with AngularJS in the front end. Or you can use both an SQL and a NoSQL

database in the same application, if that suits your needs.

Developers are in a lucky situation in which they're, in a sense, spoiled with choices. The best answer to my question is to use your best judgment and pick and choose as you need.

Neither LAMP nor MEAN will be the last word on developing Web applications. New ideas and approaches will emerge to respond to emerging needs. For example, the Internet of

Things will require rethinking how we develop our applications to make them fit in all sorts of small, embedded devices. The emergence of distributed applications built from autonomous nodes needs technologies appropriate for building networked services instead of shoehorning existing tools—the reception of the Go programming language might hint at that. Moreover, future applications will be built from many independent components. Those components won't all need to be deployed and configured in tandem, nor will they need to be updated together.

Frameworks will be judged by their flexibility and their facilities for handling component modules.

For links to more information on LAMP and MEAN, see the “For More Information” sidebar. 🔍

Reference

1. S. Cass, “The 2015 Top Ten Programming Languages,” *IEEE Spectrum*, 20 July 2015; <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>.

PANOS LOURIDAS is an associate professor at the Athens University of Economics and Business and a consultant at the Greek Research and Technology Network. Contact him at louridas@aueb.gr.



FOR MORE INFORMATION

If you're interested in LAMP frameworks, check out these sites:

- The Java Spring Framework, <http://projects.spring.io/spring-framework>;
- Ruby on Rails, <http://rubyonrails.org>;
- Python Django, www.djangoproject.com; and
- PHP, <https://secure.php.net>.

To explore MEAN, start with these sites:

- Node.js, <https://nodejs.org/en>;
- Meteor, www.meteor.com;
- Express, <http://expressjs.com>;
- AngularJS, <https://angularjs.org>; and
- Ember, <http://emberjs.com>.

Security is important in Web application development. A good starting point is the OWASP (Open Web Application Security Project) Top 10 Project (www.owasp.org/index.php/OWASP_Top_Ten_Project), which presents the most critical Web application flaws.

ADVERTISER INFORMATION • MARCH/APRIL 2016

Advertising Personnel

Debbie Sims: Advertising Coordinator

Email: dsims@computer.org

Phone: +1 714 816 2138 | Fax: +1 714 821 4010

Chris Ruoff: Senior Sales Manager

Email: cruoff@computer.org

Phone: +1 714 816 2168 | Fax: +1 714 821 4010

Advertising Sales Representatives (display)

Central, Northwest, Southeast, Far East:

Eric Kincaid

Email: e.kincaid@computer.org

Phone: +1 214 673 3742

Fax: +1 888 886 8599

Northeast, Midwest, Europe, Middle East:

David Schissler

Email: d.schissler@computer.org

Phone: +1 508 394 4026

Fax: +1 508 394 1707

Southwest, California:

Mike Hughes

Email: mikehughes@computer.org

Phone: +1 805 529 6790

Advertising Sales Representative (Classifieds & Jobs Board)

Heather Buonadies

Email: h.buonadies@computer.org

Phone: +1 201 887 1703