Cloud Software Development Platforms: A Comparative Overview

Kyle Schutt
Department of Computer Science
635 McBryde Hall
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Osman Balci
Department of Computer Science
638 McBryde Hall
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

***Abstract*—Doing things "in the cloud" has become ubiquitous, and the cloud has become a rich environment for the use of technology anywhere and anytime to solve problems, connect people, and improve lives. Software engineering paradigms have been shifting during the last decade from "Software-as-a-Product (SaaP)" to "Software-as-a-Service (SaaS)" provided "in the cloud". The SaaS software paradigm comes with new capabilities and technical challenges for cloud software development, as compared to mobile and stand-alone software development. Distributed multitiered architecting, design, and programming for cloud software development require new strategies specifically motivated by the SaaS paradigm. This paper aims to compare and contrast cloud software development platforms and frameworks, namely, Java platform, Enterprise Edition (Java EE); Microsoft platform, .NET framework; Ruby on Rails framework; Zend framework; Node.js framework; and Django (Python) framework. A comparative overview is presented to help cloud software engineers select an appropriate platform / framework to solve a complex problem.**

Keywords—cloud architecture, software engineering, Java EE, .NET, Ruby-on-Rails, Django, Node.js, Zend

I. INTRODUCTION

The emergence of consumer-accessible and economically viable web-based virtual machines that can partially, or wholly, replace infrastructure requirements has ushered in a new era of software engineering. This new era of net-centric web-based applications redefines how software is delivered to a customer, and how the customer uses the delivered software. This new framework of software delivery and deployment has been colloquially defined as *cloud computing*. While the concept of net-centric applications has been around for decades, the barriers to entry into this space were limiting. However, the reduction of hardware costs, increased availability of resources, and the maturation of virtual machine technology have led to a new paradigm in software engineering: cloud software engineering.

While the technology behind cloud-based applications was the catalyst for the new paradigm, software engineers began to realize a need for specific platforms to develop a new style of web-based applications. This requires the platforms to be maintainable, extendable, configurable, testable, and feasible in the cloud environment. Over the years, various terms have been applied to these platforms such as web-based, full-stack, net-centric, cloud-based, client-server, Software-as-a-Service (SaaS), and cyber infrastructure frameworks. Each of these terms is indicative of varying types of web-based software with differing features and objectives. That being said, each term satisfies the same core objective of providing services over the Internet to be ingested by a client, in which the client can be multimodal.

In this comparative overview, several major platforms are analyzed to give the reader an idea of the underlying technology for each of the platforms. This is done based on the major language used within a given system. This paper aims to compare and contrast Cloud Software Development (CSD) platforms and frameworks that employ a number of major programming languages such as Java, .NET (C# or ASP), Ruby, PHP, JavaScript, and Python. This overview is by no means a comprehensive list of available platforms / frameworks for each of these languages since new ones are developed and released quite frequently. For each of these programming languages, this paper aims to discuss the major platforms / frameworks that are available to the cloud software engineer: Java EE, .NET, Ruby-on-Rails, Zend, Node.js, and Django, respectively. This overview aims to identify and analyze the five different tiers in a client-server architecture with respect to these frameworks: the Client, Web, Business, Data Mapping, and Data Sources tiers.

Java Enterprise Edition (Java EE) is a platform that has redefined how Java is used in the web-environment. Prior to the introduction of Java EE, Java was seen as a workhorse language for desktop computers. However, the thought of using Java in a web-based environment was unthinkable. There were multiple attempts to kick start Java Applets as a way to run Java on the client machine through a browser, but each attempt was met with resistance because of existing limitations of browsers and security issues with Java Applets.

978-1-5090-0809-4/16/$31.00 copyright 2016 IEEE
SERA 2016, June 8-10, 2016, Baltimore, USA

Another major drawback was the reliance on Swing to generate graphical user interfaces (GUIs), which would work well in a desktop environment but suffered in a browser-based environment due to resource limitations. This paper discusses the benefits of Java EE and how it overcame some of these issues by incorporating new UI designs to become the powerhouse in web-based applications that is seen today.

Microsoft's .NET framework was developed with network-centric architectures in mind from the beginning, hence the .NET moniker applied to all web-based technologies during development. With .NET, C# and ASP (and to an extent, F#) are the languages of choice when developing a .NET application. These applications have the benefit of being easily and readably deployable into a Microsoft environment that is scalable and easy to maintain using Internet Information Services (IIS). Additionally, .NET has a massive developer network that makes engineering .NET applications significantly easier.

Ruby-on-Rails (Rails) is a relative newcomer to the cloud software platform game, but it has matured alongside cloud computing since its inception. Rails has become a *defacto* standard for lean startups which are looking for a scalable and maintainable platform for new products and services. Under the hood, it relies on Ruby, which is a dynamically typed object-oriented programming language with the design goal of being accessible and "fun" to use for a programmer. Rails is an extension of Ruby to create a web-application framework that values convention-over-configuration, and follows the Model-View-Controller (MVC) architecture paradigm. This paper discusses the Rails architecture in more detail in Section 4.

Zend is the Rails equivalent for PHP. PHP has been a web-programming language since the early 1990's and is a server-side scripting language that is still widely used by some of the most popular web-based applications, such as Facebook, which has even developed a custom implementation of PHP. In the past, PHP was simply mixed in with HTML, which is then preprocessed on the server before being sent to the client's browser. Today, PHP has been extended and embedded into various templating engines and web frameworks. This paper focuses on one such framework called Zend. Zend provides an object-oriented extension of PHP that loosely follows the MVC architecture paradigm.

Node.js is quite notably the most unique web-based framework that is currently available: the server- and client-side are both written in JavaScript. Node.js is relatively young but provides an interesting model for how a web server should behave, that is an event-driven framework instead of a multi-threaded framework. Unlike other platforms, Node.js can handle thousands of requests easily by leveraging event-driven programming.

Django is the Zend equivalent for Python. Python is an interesting language that has gained popularity in recent years for its simple syntax and pervasiveness in non-technical classrooms. There has been an explosion of Python modules and wrapper for various complex and scientific applications that have decreased the barrier to entry for new and non-programmers alike. Django is one such platform that provides software engineers with a framework that follows convention-over-configuration and the MVC architecture pattern in such a way that allows informally trained developers to readily understand the concepts. This paper focuses on Django, but a number of other frameworks and extensions exist for Python providing easy access to various components in a client-server architecture.

The remainder of this paper is organized as follows: Sections 2 through 7 provide an overview of each of the major platforms / frameworks and discuss the five tiers of a client-server architecture. Section 8 provides a comparative overview of all of the platforms / frameworks mentioned above. Finally, concluding remarks are given in Section 9.

II. Java Platform, Enterprise Edition (Java EE)

A *Client-Server Architecture* (CSA) based on the Java platform, Enterprise Edition (Java EE) [2,13] is shown in Figure 1 with five tiers: (1) client tier, (2) web tier, (3) business tier, (4) data mapping tier, and (5) data source tier. Java EE is a powerful web technology that builds on the success and popularity of Java to provide developers with the tools required to build large, robust web-applications. When a user initiates a request to a Java EE web-application, the web server which is typically IIS or Apache analyze the request to determine which servlet or which function within a servlet must be executed. These requests might refer to one or more Java Server Faces pages (JSF) that leverage technology like XML User Interface Language (XUL) to generate web pages.

While the base Java EE favors convention over configuration, it does not inherently force the developer to use the MVC architectural pattern as in other architectures. Therefore, the incoming requests can also be processed by the JAX-RS RESTful web services, which in turn execute some functionality to produce the desired result. If the system requires data from a data source, the system can leverage Persistence API Entities that represent data in the data source. From here, the application can perform all necessary computation and instantiate the JSF page. The JSF page renders itself and returns to the client the HTML, CS, and JavaScript that it requires. These are only a small sampling of technologies available to the Java EE platform.

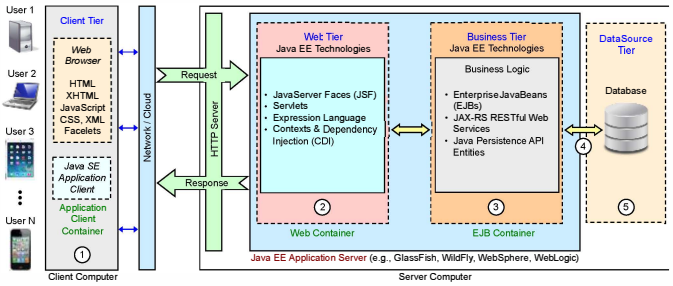

Figure 1: Java EE-based Client-Server Architecture

A. Tier 1: Client

Java EE applications are unique in that not only do they support the typical web protocols like HTML, but they can also work in a dedicated client container to run Java code. Java EE can also be used as a RESTful API by returning data in XML or JSON, or any format for that matter, to the native client application to be displayed. Many of the libraries required to render JSF pages are loaded remotely as JavaScript components when the page is displayed on a browser. These libraries can either be hosted by the Java EE server or on a content delivery network (CDN) to minimize code reuse and load on a developer's infrastructure. A browser is not the only method of ingesting a Java EE web application as it can server as the backend implementation for any types of clients from smartphones to other web servers.

B. Tier 2: Web

The web tier for a Java EE web-application consists mainly of JSF pages (as define by the expression language) and Context & Dependency Injection (CDI). CDI provided developers the ability to dynamically request modules as needed based on the incoming request. While JSF is not required by Java EE to serve content over the web, it is highly recommended as the standard method for generating HTML and JavaScript based on the incoming request since it abstracts data bindings and application logic by with expression language. The design of the JSF framework also values component reuse and extensibility which greatly reduces overall development work for the software engineer.

C. Tier 3: Business

The business tier contains all logic pertaining to a specific business need by implementing various Enterprise Java Beans (EJBs). These EJBs can then respond to specific requests to JAX-RS RESTful calls that perform some predefine function on the models. EJBs can also respond to specific events that originate from a JSF page or component. The Java Persistence API is also listed here because it is used to define the models that are represented in the database. The business tier in Java EE applications is responsible for the security and the state of the current application. These sessions are managed by the application on a client-by-client basis to improve the security and continuity of the business logic components. Finally, while Java EE is mostly a request-driven framework, it also contains modules that allow for event-driven processing and messaging between the server and the client called the Java Messaging Service [14].

D. Tier 4: Data Mapping

Java EE provides several different options for data mapping technologies including Java Database Connectivity (JDBC), Java Persistence API, Connector Architecture, and Transaction API. While each of these options relies on the base Java EE APIs for connecting to a data source, each of them play a unique role in the ecosystem. JDBC and the Java Persistence API are mainly used in the management of relational data objects by defining entities and generating SQL queries based on the class definition. The Connector Architecture is primarily geared toward enterprise customers looking to integrate legacy Java applications into a Java EE web application. Transaction API is a more fine-grained approach to JDBC by managing individual methods within an entity.

E. Tier 5: Data Sources

Java EE provides support for all major database management systems such as Oracle, DB2, SQLServer, MySQL, and PostgreSQL. Additionally, Java EE supports NoSQL data sources like mongodb. Like other web-application frameworks, Java EE is fairly agnostic to the systems implemented as the data source. Modern data mapping technologies, like Java Persistence API, are generic enough in definition to be extended to work with any kind of data source, object- or document-based. The type of data source utilized by a web-application is entirely dependent on the requirements and expertise of the developers involved. The choice of relational or NoSQL is left to the software engineer since there has yet to be a unified consensus on the appropriate usage of data sources for a given application.

III. Microsoft Platform, .NET Framework

A client-server architecture based on the Microsoft platform, .NET framework [7] is shown in Figure 2 with five tiers. A .NET web application can be developed using a variety of languages such as C# or Visual Basic (VB) to create code-behind web pages and web services. It differs from scripting languages like Python in that it compiles down to Microsoft Intermediate Language to be run on the common language runtime (CLR) engine that can provide additional performance benefits. A request to a .NET web application is first received by the site hosted on an Internet Information Services web server (IIS) before being forwarded to the .NET code. In general, the URL paths match up directly to a specific webpage that is defined in the site, such as "/Admin.aspx".

However, in ASP.NET MVC, dynamic routing can occur that allow for the developer to create data-driven web pages based on the incoming request. Once the ASP.NET page has been initiated, the business logic is executed. This can be defined in the XHTML template itself as embedded dynamic code or in the code-behind class of the web page. Here managed components and classes from the .NET library can be initiated and utilized to perform the appropriate functions. If necessary, the model may need to retrieve data from a data source and, therefore, initialize a data persistence layer to obtain the data. Once the retrieval is complete, the server-side webpages are generated and compiled down to HTML or XHTML with CSS and JavaScript before being sent back to the user.

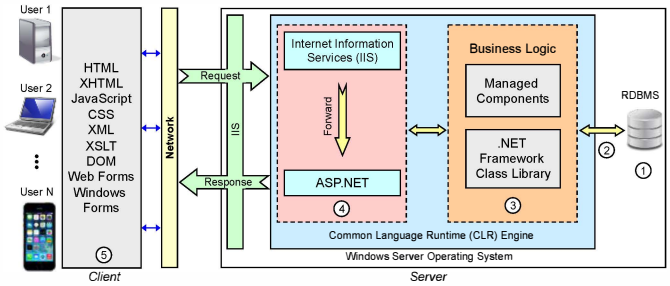

Figure 2: .NET-based Client-Server Architecture

A. Tier 1: Client

Similar to most other web application frameworks, .NET supports HTML, CSS and JavaScript to properly display the content to the user. Similar to Java EE, .NET applications also support XHTML, which is a well-formed XML-based markup language. Depending on the implementation of a .NET application, the output data format can vary depending on the client. A .NET application can output HTML/XHTML formatted data to be displayed by a browser, or, as a .NET web service that outputs XML formatted data to be ingested by a client application. A .NET web service can follow either the RESTful or SOAP paradigm depending on its implementation.

B. Tier 2: Web

For .NET applications, the de facto webserver is IIS. It provides out-of-the-box support for all types of .NET web applications and provides the developer with a powerful tool to manage multiple sites, permissions, SSL, and other features. However, it is also possible to host .NET web applications under Apache and nginx with mod_mono and FastCGI, respectively [5, 11].

Regardless of the web server, the second part of the web tier is routing the request to the correct web page for processing. In .NET applications, there are two different methods that handle routing depending on the web application and version of .NET being used. The first method is a direct request for a web page with the URL. The URL path defines a specific page that is fetched by web server and rendered to the user. If a page does not exist, the user gets a 404 page not found error from the web server. The second method uses dynamic routing defined in a .NET MVC web application. This allows the developer to define data-driven web pages based on the URL path being processed by the web server. It also allows the developer to create a web page that handles errors more gracefully then a harsh 404 message.

The web tier is also where the dynamic web pages are compiled into HTML or XHTML depending on the implementation. The web pages can be designed and implemented using a variety of languages. The design of the page is developed with XAML-defined user controls (for Windows Presentation Foundation applications), Web Forms, XSLT styling, CSS styling, and others. The code-behind for the webpages can also be implemented in a number of languages like Visual Basic, C#, or the more esoteric F#.

C. Tier 3: Business

The Business Tier in .NET applications contains a wealth of components, extensions, and objects that can be used and implemented in the design code or the code-behind files. This can include simple UI components to complex application-specific logic. Additionally, modules and components can be added to a web-application to minimize code reuse and increase modularity for both UI controls and application logic. While there is no central repository for .NET modules and components, many can be found through Google search or on the Microsoft Developer Network (MSDN).

D. Tier 4: Data Mapping

The primary data persistence layer for .NET web applications is ADO.NET, but Microsoft recommends Entity Framework for applications. ADO.NET is not entirely an object-relation mapper (ORM) because it only utilizes certain data structures and still requires the developer to configure much of the SQL and data access [8]. There are several other ORMs that are available to RDBMS that are more powerful and easier to maintain such as Entity Framework and Fluent NHibernate [6]. Each of these extensions has its own tradeoffs, but are far superior in providing a convention-based implementation of an object that represents relational data. These ORMs also provide support for LINQ to improve readability when working with database objects [9].

E. Tier 5: Data Sources

The primary data source for .NET web applications is SQL Server, but other supported relational databases include MS Access, SQLite, MySQL, DB2 and Oracle (using the Oracle Client data provider). NoSQL support is also available for data sources such as mongodb. Additionally, Microsoft provides other adapters and drivers for non-conventional data sources with the Object Database Connectivity standard (ODBC) such as spreadsheets, object databases, and Microsoft Access.

IV. RUBY ON RAILS FRAMEWORK

A client-server architecture based on the *Ruby on Rails* framework [15] is shown in Figure 3 with five tiers. A Ruby-on-Rails (RoR/Rails) web application is a solution wherein all tiers are managed by the Rails architecture. When a user initiates a well-formatted URL, the Rails Router verifies the URL path and dispatches the request to the correct controller and action [16]. Depending on the implementation of the action within the controller, various models may be instantiated or fetched from the database. The object-oriented models are mapping to objects in the database that can be retrieved and used to accomplish business-specific logic.

Additionally, various RubyGems can be bundled with the application to provide additional operations not provided in the standard Ruby libraries. Once the business-specific logic has completed successfully, Rails generates the response

required by the request. The response is generated based on the template ERBs that match the action, and then sent back to the user. The user ingests this response and appropriately displays it as necessary.

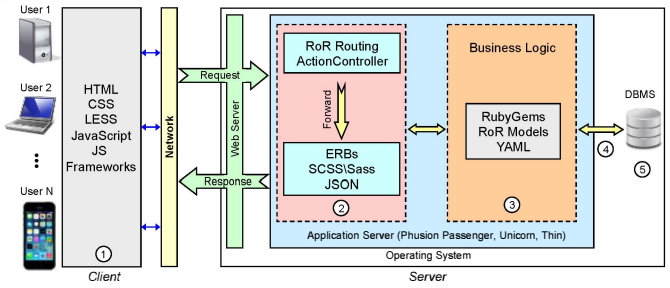

Figure 3: Ruby on Rails-based Client-Server Architecture

A. Tier 1: Client

Like all web application frameworks, the client side primarily utilizes HTML, CSS, and JavaScript to display a web page that is generated from the Rails application. Rails also supports various languages that transcompile into CSS or JavaScript, such as: CoffeeScript for JavaScript; or LESS, SASS or SCSS for CSS. Additionally, Rails works with client-side JavaScript-frameworks (such as Angular.js) that follow the MVP pattern since Rails is architected directly on the MVC pattern.

Depending on the implementation of the Rails application, any client that can ingest XML or JSON from an API can connect to the Rails application. This includes native iOS and Android applications that can utilize the API to obtain data and then display it directly to the user. By default, Rails generates a JSON API of all resources that are created with the Rails Generator. Additionally, software engineers can create customized Rails generators to improve workflow and canonize automatically generated code to meet the specific requirements of an application [18].

B. Tier 2: Web

For Rails applications, the Web Tier handles incoming requests and outgoing responses differently. As requests come through the web server, the standard router is the Rails Router that dispatches requests to an action in the specified controller. The Rails Router also handles the generation of the acceptable URL paths based on the resource definitions, dynamic segments, helper functions, namespaces, and other useful options for use within the application.

For outgoing responses, the web tier in Rails applications is responsible for generating the correct data based on the format specified by the incoming request. For example, the controller is responsible for responding with HTML or JSON. If the request requires a JSON format, the controller can respond with JSON-formatted data from an action in the specific controller. Otherwise, the Rails application responds with an HTML document that is defined by an ERB (Embedded Ruby) file that includes HTML with Ruby mixins.

C. Tier 3: Business

The Business Tier of a Rails application is where all of the computation occurs. This is where application-specific logic is defined as Rails models or in reusable helper modules. Additionally, this is where any configuration, internationalization, and initialization are defined for the application. The one major component in the Business Tier is RubyGems. In a Rails application, a *gem* is a reusable package of code that provides specific functionality. These are essentially modular plugins that allow the developer to provide certain functionality in their applications. Gems can range from very complex scheduling expansions to simple sortable models. Currently, there are over 6,000 official RubyGems and even more are available in the open source marketplace.

D. Tier 4: Data Mapping

The Data Mapping tier in a Rails application varies depending on the data source. Relational database management systems (RDBMS) are defined by ActiveRecord with a specific adapter based on the type of database being used—a comprehensive list of adapters can be found at the Ruby Toolbox [19]. As for document-based databases (NoSQL), there are various data persistence implementations based on the type of database employed. For example, mongodb has a data-mapping layer called mongoid that can be used in Rails applications [10].

E. Tier 5: Data Sources

As mentioned in Tier 4, the data sources for rails can encompass both SQL-based and NoSQL databases. This can also include graph-based and XML-based data sources provided the developer is able to implement the proper ActiveResource to read and write from an API defined by the data source. For example, a popular XML-based data source is eXistdb; while no gem exists that provides an adapter to eXistdb, the functionality can be defined in an ActiveResource that maps and creates queries to access RESTful resources in the database [4, 17].

V. ZEND FRAMEWORK

A client-server architecture based on the Zend framework [21] is shown in Figure 4 with five tiers. The Zend web application framework provides a solution to web-based application written in PHP. Similar to Rails, it is built using the MVC design pattern to favor convention-over-configuration, loosely coupled components, and to be lightweight. Zend is just one MVC-based framework, CakePHP is another—they generally have the same functionality and it is really a religious decision which one to utilize. When the user initiates a request to a PHP web application, a web server first fields the request. This webserver can be any of the major players such as IIS,

Apache, or nginx. The Zend Application Server that manages all server-side components supports all of these web servers.

Once the request is processed and passed to the application, the Zend Router and Zend Controller process the request and pass the request to the appropriate action based on the matching controller. This functionality is nearly identical to Rails Routing. From here the request is processed in the action of a controller, this action may utilize various Zend plugins and Zend models to perform a specified business function. These models can be representative of data stored in a data source and extracted via a persistence layer that is compatible with that data source. Once all the processing is complete, Zend generates the view through View Scripts, the PHPrenderer, or Zend_View. Each of these generators performs different functions that are discussed below, but all have the same end result of producing an HTML document embedded with CSS and JS to be displayed on the user's device.

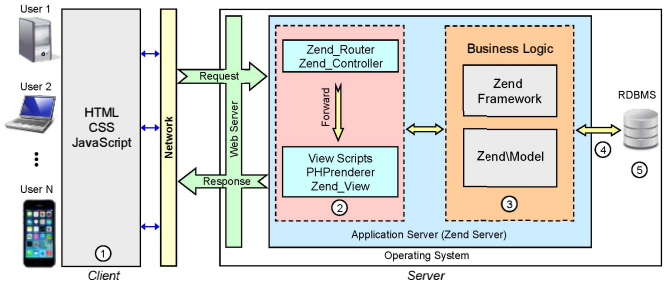

Figure 4: Zend-based Client-Server Architecture

A. *Tier 1: Client*

Like the other web application framework discussed, the client-side code generated by Zend and other PHP-based frameworks all support HTML, JavaScript, and CSS. Additionally, Zend has the ability to act as a RESTful endpoint for native applications that require an API.

B. *Tier 2: Web*

The Web Tier in Zend is split into different parts: the router and controllers, and the view rendering. The Zend Router handles matching URL requests with the appropriate actions. It then passes to the correct controller delegation based on the action specified. The Zend Router can only handle the requests that are defined in a last-in-first-out queue. Once the controller's action is executed, then the Business Tier objects are instantiated.

Additionally, the Web Tier is responsible for rendering the appropriate view based on the action and the controller. This can either be in simple View Scripts, the PHPrenderer, or a Zend View. View Scripts are the abstract object used to generate a user-defined Template based on PHPLIB, or utilize the Zend View Interface to generate their own customized view objects. Zend View allows the developer to incorporate a third party templating system to ease the complexity of a PHP view template. Finally, the PHPrenderer is the heavy-lifting rendering engine for PHP templates as defined by View Scripts or in a Zend View. It also acts as a "router" of sorts for mapping actions to their specific templates to be compiled and sent to the user.

C. *Tier 3: Business*

The Zend Business Tier is primarily made of Zend Models that are defined by a set of abstract classes. These abstract classes are used to represent the model in the data source, if necessary. If they are standalone models, they are identical to any other PHP class declaration. In this Tier, developers can leverage many of the plugins and objects offered by the Zend framework such as pagination or user authentication.

D. *Tier 4: Data Mapping*

For most applications built with Zend, the data persistence layer is defined within the Zend Models as a set of defined functions inside the model and an initialization to the data source. Additionally, there are specific ORMs and ODMs that can be used with Zend—the most well-known of which is Doctrine. Doctrine is similar to Hibernate for Java and NHibernate for C#. Doctrine allows the developer to generate PHP objects directly from database objects and vice versa.

E. *Tier 5: Data Sources*

Zend can work with a variety of databases that are either relational- or document-based provided that the developer utilizes the correct data mapper or implements the queries properly. Zend is similar to Rails in that it is completely agnostic to the type of data source being used provided that the developer properly generates the code needed for access to the backend services.

VI. NODE.JS FRAMEWORK

A client-server architecture based on the Node.js framework [12] is shown in Figure 5 with five tiers. Node.js is a web application framework developed entirely in JavaScript for both the server- and client-side. It is unique in this overview in that it is fully developed in the same language from frontend to backend. Other frameworks that have been discussed only use JavaScript on the client-side. Because of this design choice, the barrier to entry when creating a web-based application is quite low considering a developer only needs to spend time learning a single language.

Node.js can run as a standalone web-server or behind Apache or nginx. Upon initiating a properly formatted URL to a Node.js application, the request is filtered to the web server. The request is then passed to a router that is defined by the user. The router dispatches the request to a specified controller and action to perform a business operation. This operation may require access to the database through the data-mapping layer. The data-mapping layer for Node.js is generally written as a JavaScript object (the schema of which is dependent on the JavaScript data mapping framework employed by the user). Once the business operation is complete, the response is sent back to the web tier where a templating engine, like Jade, generates data-driven webpages to be displayed to the client.

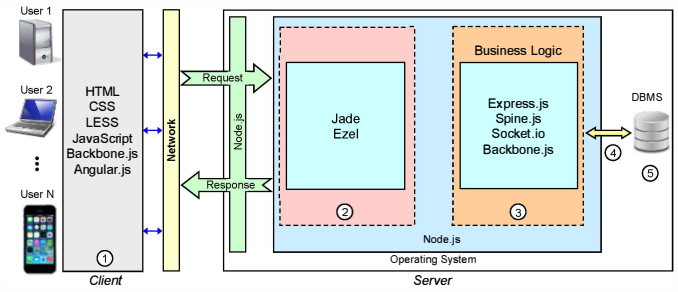

Figure 5: Node.js-based Client-Server Architecture

A. Tier 1: Client

The client-side code requires the use of HTML, CSS, or JavaScript to display a webpage. It is also possible that the client could be ingesting an API. Additionally, Node.js applications can leverage several client-side JavaScript frameworks to create dynamic webpages, such as those created with Angular.js or Backbone.js. Angular.js has the added benefit of being a client-side MVC/MVP framework that allows the web-application to be a simple API that can then be ingested by Angular.js to create dynamic webpages on the client-side. This alleviates much of the rendering that generally occurs in the web tier using server-side templates.

B. Tier 2: Web

The web tier for Node.js is generally handled by Node.js itself as the web server. It can run behind Apache or nginx, if necessary. Node.js is an event-driven web server built using Google V8 JavaScript engine that allows it to register with the operating system and field requests coming into the web server. Similar to other architectures, the Web tier performs double duty of routing incoming requests and generating template HTML documents to send to the user. For Node.js, routing can be defined by a myriad of different frameworks, the most robust of which is Express.js. For templating HTML documents, Jade is an example of a template engine that is similar to Zend Views. However, it is possible that no template framework needs to be installed if the developer utilizes a data-drive client-side template engine like Angular.js.

C. Tier 3: Business

The business tier in a Node.js application can leverage several different JavaScript frameworks to improve the development process of applications. Additionally, it is possible for the developer to create their own classes provided that they are written in JavaScript and can be executed by the Node.js web server. Several of the popular frameworks include Express.js, Spine.js, Socket.io, and Backbone.js. Each of these frameworks has different implementation strategies but each of them is geared toward providing a powerful sandbox to perform business logic. Many of these strategies are centered on asynchronous user interfaces (model- and data-driven web pages), and event-driven networking. Spine.js and Backbone.js are two examples of event-driven frameworks built on Node.js to provide a lightweight API to the web application. Express.js is similar in that it focuses on creating a thin, but powerful, HTTP server that allows the developer to seamlessly integrate various ORMs and template engines.

D. Tier 4: Data Mapping

In Node.js there are a large variety of data mapping frameworks depending on the type of database that is being utilized. These frameworks could be geared toward a SQL database using Sequelize, or mongodb using mongoose. Additionally, Node ORM works for both SQL and NoSQL database but it still quite young.

E. Tier 5: Data Sources

For Node.js, the standard data sources are NoSQL databases like mongodb or couchdb. With Sequelize, and other ORMs, the most popular data source like PostgreSQL and MySQL can also be utilizes. However, Node.js is a dynamically typed language and, therefore, benefits tremendously from unstructured data in NoSQL databases.

VII. Django (Python) Framework

A client-server architecture based on the Django (Python) framework [3] is shown in Figure 6 with five tiers. Python is quickly becoming a popular language for solving simple and complex problems in various industries such as Geographical Information Systems (GIS) and statistical analysis. Additionally, many web-application frameworks have been created using Python such as Django, Tornado, Flask, and many more. While Python is generally seen as a high-level procedural language, it has developed into a dynamically-typed multi-paradigm language that is capable of solving a variety of programming problems on all major platforms. For the sake of brevity, this paper provides an overview of a Python CSA using one of the more mature and feature-rich web-application frameworks for Python: Django. Like many of the other framework discussed, Django follows the MVC architectural pattern and provides many of the same components.

A web-application built with Django can be deployed in several ways; including GlassFish and WildFly (JBoss). When a user first initiates a request on a client device, the web server fields the request. Django provides a built-in web server for development and testing purposes, while deployments into a production environment generally require Apache or nginx with a Python extension. Once a request is received, it is sent to Django's URL *dispatcher*, which is highly configurable but has fewer features than that of a fully-fledged router. The dispatcher calls the function on the view, which is acting as a controller in this instance. The view leverages any included Python Modules and Django Models. The Django Models are then mapped to a DBMS that can either be relational or document-based. Once the business logic is complete, control returns to the view, which calls the template code that is

defined in a Django Template. These templates generate HTML, JavaScript, and CSS that are displayed to the user.

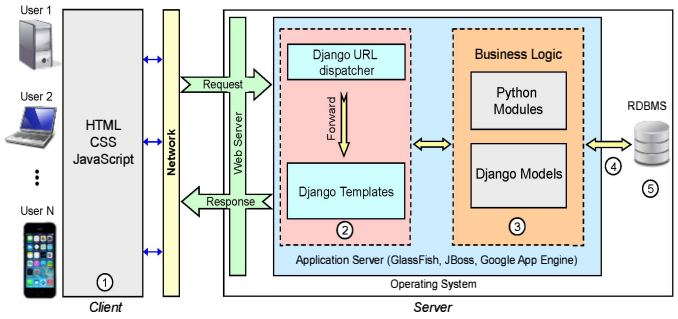

Figure 6: Django (Python)-based Client-Server Architecture

A. Tier 1: Client

The client-side code requires the use of HTML, CSS, or JavaScript to display a webpage. It is also possible that the client could be ingesting an API. Python web-frameworks are well-suited for data translation to JSON or XML that can be consumed by a native application on the client side.

B. Tier 2: Web

Django (and other Python frameworks) use a simple URL dispatcher to handle incoming requests. In some cases, the URL dispatcher is as simple as calling a specific function to generate a page. Unlike Node.js, Zend, or Rails, the URL dispatcher simply matches regular expressions instead of providing a full router to handle requests. Django Templates are views that use the Django Template Language to generate data-driven webpages. These templates are highly extensible and reusable once the syntax and the extension system is understood.

C. Tier 3: Business

The business tier for a Python-based framework is where the real power of Python shows through because of the growing number and maturation of many high profile and powerful modules built in Python (or, at least, has a Python wrapper). There is not much that Python cannot achieve based on its current state. Modules such as NumPy and SciPy have opened the door to complex numerical and statistical analysis, and to calculations that used to be handled singly by R or MatLab. Additionally, GIS has benefited greatly from the implementation of ArcPy by ESRI for its ArcGIS line of products.

The business tier for Django allows developers to include any and all Python modules and provides a data model schema that can be used to quickly access data in an object-oriented fashion. These modules and models can work independently of one another depending on the business requirement that is being addressed.

D. Tier 4: Data Mapping

As prolific as the business tier, the data mapping tier has no shortage of modules available to assist developers in mapping a model to a database entity whether it be relational- or document-based. For relational databases, Storm, SQLAlchemy, and Django's ORM are the standard mappers that are available. For mongo, PyMongo is used as the ODM. Additionally, Django can be run on Google App Engine and take advantage of the data mapping to Google Datastore. Finally, Python can also leverage JDBC with GlassFish to access data.

E. Tier 5: Data Sources

Similar to other web-application frameworks, Django supports almost every major relational- and document-based databases. For relational, this includes MySQL, SQLite, PostgreSQL, and others. For document-based and NoSQL, this includes mongodb and Google Datastore. Again, any database is supported provided that a developer is capable of creating the correct data mapper.

VIII. Comparative Overview

This section compares and contrasts the six Cloud Software Development (CSD) platforms and frameworks. Table 1 and Table 2 present the technologies used by each CSD platform or framework at each tier of the five-tier client server architecture.

The Client Tier, Tier 1, is almost entirely defined by the web standards currently in place. At the most basic, each of the platforms supports HTML, JavaScript (JS), and Cascading Style Sheets (CSS). Since most web-based applications are delivered directly to users over the Internet through a web browser, these are the standard technologies. However, the current landscape of cloud software engineering also includes mobile devices, web-connected devices, and other web applications. On the other hand, many of the clients leveraging a web-based application do not require user interface components and can, therefore, communicate by utilizing other data transfer standards such as XML or JavaScript Object Notation (JSON).

Most modern web browsers also support web standards that are less prevalent than others, but are essential variations of HTML, JS, or CSS such as XHTML, LESS (a stylesheet language), or Sassy CSS (SCSS). XHTML is a stricter version of HTML while LESS and SCSS are scripting languages for CSS that allow developers to create more complex rules for CSS. LESS and SCSS have the added bonus of either being compiled on the client-side, hence being placed in Tier 1, or on the server-side, which would place them in Tier 2.

Table 3 shows the client tier technologies used by the CSD platforms / frameworks. Table 3 shows that all of the frameworks support the standard web technologies, as expected, while a few frameworks leverage additional supported technologies like SCSS and XHTML, which are extensions to CSS and HTML, respectively.

The Web Tier, Tier 2, has two major roles in a web application. The first role is to analyze and act upon incoming requests from a client to properly dispatch the request to the appropriate controller for further execution. The second role is to create the response once the processing has been completed in Tier 3. This response is initially based on the type of format the client is requesting. As discussed in Tier 1, this could be a web interface, JSON, or XML. In all of these cases, the second role of Tier 2 is to generate the output data based on some type of template. Each platform / framework is equally capable of outputting any format because each platform has some version of a templating engine to generate the content.

Table 4 shows the component technologies of the CSD platforms / frameworks. Tier 2 fulfills several roles in a cloud application and within each of these roles contains various technologies. For example, in the web server role, Tier 2 includes the web server itself, a routing engine, and a controller. The web server can be embedded in the framework like Node.js, or as additional component like Apache. When these frameworks are deployed, many of them are hosted via an Application Server that provides a stable environment for production deployments. For example, a Java EE application can be hosted via Tomcat for testing purposes but it is recommended that the application be hosted within a Java EE server that has both the web and EJB containers [1, 13].

The web server then uses a router to dispatch execution to the appropriate controller based on the incoming request. In the simplest case, a router utilizes regular expressions to determine the controller and action based on the URL path and parameters. The router delegates execution and the controller executes the action. The action completes and the controller notifies the framework.

In the second role, where the controller notifies the framework, the framework generates the requested view. Depending on the request, the views can be quite different. For example, a simple API request that requires no HTML is rendered as an XML or JSON formatted response. In the case where a user has requested an HTML document, the renderer generates an HTML document based on data-driven templates as defined by the developer.

The Business Tier, Tier 3, is where all the heavy-duty processing and business logic occur in a web application. This is where the web application's purpose is defined and executed. For each of the platforms, there are a large number of additional tools, modules, extensions, and plugins that can be leveraged to achieve the desired output. Tier 3 is also where the web application accesses data objects through Tier 4 when performing the business logic.

The Data Mapping Tier, Tier 4, is a lightweight tier that allows the developer to define how data objects are stored in a data source. For example, this is where models defined in Tier 3 are translated into the data source's language for Create-Read-Update-Delete (CRUD) database operations. Some of the platforms discussed above provide this functionality out of the box, such as Rails, while others require the developer to define the data mapping directives in the model itself, such as .NET and Java EE. For Relational Data Base Management System (RDBMS) data sources, Tier 4 abstracts the required Structured Query Language (SQL) statements to perform operations on the model's representations in the database. This frees the developer from having to create the SQL statements themselves, and update those SQL statements when the model changes. For NoSQL data sources, Tier 4 abstracts away the functions required by those systems. For example, Tier 4 can abstract the mongo commands necessary to perform actions on a mongodb data source such that the developer does not have to duplicate effort in creating those commands.

Table 5 shows the various data mapping technologies for each of the platforms / frameworks based on the type of database systems. This is a small sampling of the available data mappers for each of these frameworks based on SQL or NoSQL database management systems.

The Data Sources Tier, Tier 5, is the most agnostic of all the tiers. For any given platform, if a data mapper exists for a specific data source, then that data source can be used by the platform. Even if a fully-fledged data mapper does not exist, it is still possible for any data source to be used by any of these platforms by leveraging specific base classes in each.

Table 6 shows data source technologies for the CSD platforms / frameworks. The Tier 4 technology is entirely dependent on the selected Tier 5 technology. Even if there are no available Tier 4 technologies for a selected database system, it is still possible to utilize the system within a specified framework. The frameworks are designed in such a way that they are agnostic to the data storage system. This means that a developer can leverage any database system that meets their needs for their selected platform / framework provided that an appropriate data mapper for Tier 4 is developed and implemented.

Each of these tables illustrates that selecting a specific technology to create a web-based, or cloud-enabled, application includes non-functional and qualitative requirements of a given problem domain. Furthermore, the technology selected is really a matter of preference to the software engineer. This notion plays readily into the overarching aspirations of cloud computing of providing scalable, affordable, and accessible resources to software engineers in such a way that allows them to truly architect, design, and develop complex applications easily.

Selecting a specific platform goes beyond analyzing the available technologies and involves more than just quantitative analysis of the problem, the cloud service provider, or the functional requirements. Selecting a platform can be boiled down to the expertise of those involved in the project. For example, a company investigating the development of a next generation social media platform may not only analyze the functional requirements, but also the collective experience of its employees. If the employees have more experience in JavaScript and SQL, and the problem requires high throughput processing, then it might make sense to utilize Node.js with a relational database to create the cloud-based application. This is an important point in cloud software engineering as it

includes more esoteric and non-functional requirements, along with functional requirements, when it comes to engineering a software system of systems.

Additionally, there are functional requirements that may trump some of the more "soft" requirements because of the need to provide specific functionality from a third-party vendor. This is less of a problem in the open source space since many of the components have been ported to multiple frameworks with only slight variations in functionality and, of course, documentation. For example, it is quite easy to find multiple variations for UI extensions for Django and Rails while Java EE and .NET have fewer options. This also extends to vendors who create specific extensions of their own products. A notable example of this is ArcPy that was created and is maintained by ESRI for Python.

ESRI has embraced Python as the language of choice to make scripting workflows accessible to as many practitioners as possible in the GIS industry. Python was an appropriate choice in that it balanced the need for a simple syntax, and extensible execution engine that could handle the massive GIS calculations. However, ArcPy is the only available package for backend processing of GIS data available from ESRI and, thus, it forces software engineers to utilize Python frameworks for GIS-enabled cloud applications. As a side note, ESRI also has the ArcGIS API for JavaScript that provides client-side GIS interface elements likes maps, tools, and other components.

While having to navigate third-party components can seem restrictive, it is not entirely a showstopper since it is possible to integrate components of different frameworks into a hybrid framework that improves the reliability and performability of a system. A key characteristic of cloud computing, and cloud software engineering, is the separation of concerns. To improve the reliability of a cloud application, various functionalities are split among multiple server computers and technologies to militate against the potential failure of a single component. In fact, many of the major cloud service providers provide a variety of web service for consumers that are narrow in scope specifically to alleviate the reliance on any singular system.

With this in mind, a major component of cloud software engineering is the discoverability of existing services. In any cloud software engineering project, analyzing the landscape and availability of services is of the utmost importance alongside governance and deployability. One part of engineering for the cloud includes selecting the appropriate technology, and, secondly, minimizing code reuse and development cost by discovering components that wholly, or partially, solve the problem at hand. Table 7 provides a breakdown of available repositories and activity on developer community sites. Governance is more directly linked with specific cloud service providers; unless the application requires third-party vendors that have predefined service level agreements. Finally, the deployability varies in each framework based on the availability of services and the components themselves. For example, Rails has the ability to be deployed quickly and efficiently to the cloud in a large number of ways by leveraging GitHub, Heroku, or Capistrano, to name a few.

In summary, the selection of a cloud-enabled framework for a specific problem domain is predominantly dependent on the specific preferences of the decision makers and stakeholders involved in the development of the solution. Each of the frameworks presented here, and others not covered, all provide the essential building blocks for creating a cloud-based application in terms of the five tiers: client, web, business, data mapping, and data sources. The selection of the appropriate framework is left to the reader and their colleagues.

IX. Concluding Remarks

Cloud software engineering has been known by many names. During 1980s, it was referred to as distributed software engineering. During 1990s, it was called network-centric, web-based, or Internet-based software engineering. The year 2000 was a turning point with the emergence of the new software paradigm called "Software-as-a-Service (SaaS)" and the creation of Application Servers [20]. Version 1.2 of Java EE, which was called J2EE, was released on December 12, 1999. Since then, Java EE has been a major platform for engineering large-scale and complex cloud software systems. Microsoft introduced the .NET framework as embedded system within the Windows Server 2003 operating system in 2003. Since then, it has been upgraded and improved every single year.

In recent years, we have seen the emergence of several new frameworks for cloud software development, namely, Ruby on Rails, Zend, Node.js, and Django (Python). These new frameworks aim to bring new levels of abstraction to make it easier for the developer to rapidly and easily create a cloud software system. These frameworks value convention-over-configuration and are architected based on the Model-View-Controller design pattern.

Selection of a cloud software development platform or framework remains to be an *ad hoc* process that is mostly influenced by personal choices rather than technical rationale. The comparative overview provided in this paper aims to assist cloud software engineers in selecting the appropriate technology to architect, design, and develop a complex system of systems for the cloud.

References

[1] Apache Tomcat (2015), "Apache Tomcat homepage," http://tomcat.apache.org/

[2] Chigani, A. and O. Balci (2012), "The Process of Architecting for Software / System Engineering," *International Journal of System of Systems Engineering 3*, 1, 1-23.

[3] Django (2015), "Django homepage," Django Software Foundation, https://www.djangoproject.com/

[4] eXistdb (2015), "eXistdb – The Open Source Native XML Database," eXist Solutions, http://exist-db.org

[5] FastCGI (2015), "FastCGI," FastCGI, http://www.fastcgi.com/drupal/

[6] Fluent NHibernate (2015), "Fluent NHibernate by jagregory", James Gregory, http://www.fluentnhibernate.org/

[7] Microsoft (2015), "Microsoft .NET Framework," Microsoft Corporation, Redmond, WA, http://www.microsoft.com/net

[8] Microsoft (2015), "ADO.NET," Microsoft Corporation, Redmond, WA, https://msdn.microsoft.com/en-us/library/aa286484.aspx

[9] Microsoft (2015), "LINQ: .NET Language Integrated Query," Microsoft Corporation, Redmond, WA, https://msdn.microsoft.com/en-us/library/bb308959.aspx

[10] Mongodb (2015), "Mongoid", MongoDB Incorporated, https://docs.mongodb.org/ecosystem/tutorial/ruby-mongoid-tutorial/

[11] Mono Project (2015), "mod_mono", The Mono Project, http://www.mono-project.com/docs/web/mod_mono/

[12] Node.js (2015), "Node.js homepage," Node.js Foundation, https://nodejs.org/

[13] Oracle (2015), "Java EE at a Glance," Oracle Corporation, Redwood Shores, CA, http://www.oracle.com/technetwork/java/javaee/overview/

[14] Oracle (2015), "Java Messaging Service Concepts", Oracle Corporation, Redwood Shores, CA, http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html

[15] Ruby on Rails (2015), "Ruby on Rails homepage," http://rubyonrails.org/

[16] Ruby on Rails (2015), "Rails Routing from the Outside In," http://guides.rubyonrails.org/routing.html

[17] Ruby on Rails (2015), "ActiveResource::Base," http://api.rubyonrails.org/v3.2.1/classes/ActiveResource/Base.html

[18] Ruby on Rails (2015), "Creating and Customizing Rails Generators and Templates," http://guides.rubyonrails.org/generators.html

[19] Ruby Toolbox (2015), "Active Record DB Adapters," https://www.ruby-toolbox.com/categories/Active_Record_DB_Adapters

[20] Tao, L. (2001), "Shifting Paradigms with the Application Service Provider Model," *IEEE Computer*, Vol. 34, No. 10 (Oct.), pp. 32-39.

[21] Zend (2015), "Zend Framework homepage," http://framework.zend.com/