# Spis treści

# 1 Introduction

An application server is a framework that provides developers with facilities to create web applications and a server on which we can run them. Application Server Frameworks contain a detailed service layer model where the application server acts as a group of components accessible to the code developer through a regular API outlined for the platform itself. For web applications, these parts are sometimes performed within the same running environment as their web server(s), and their main job is to support the development of dynamic pages. However, several application servers target way more than simply web page generation: they implement services like clustering, fail-over, and load-balancing, thus developers can concentrate on implementing the business logic.

Application servers are platforms where web applications or desktop applications run. Application server also refer to the computer hardware on which the services run. Application servers comprise of web server connectors, PC programming languages, runtime libraries, database connectors, and the organization code expected to be deployed, design, oversee, and connect these parts on a web host. An application server keeps running behind a web server (e.g. Apache or Microsoft Internet Information Services (IIS)) and quite often before a SQL database (e.g. PostgreSQL, MySQL, or Oracle). Web applications are codes which keep running on application servers and are composed in the language(s) the application server supports and call the runtime libraries and parts the application server offers.

Numerous application servers exist. The decision of choosing a server architecture impacts the cost, execution, reliability, adaptability, and viability of a web application. Exclusive application servers give system benefits in an all-around characterized yet proprietary manner. The application engineers create programs as per the specification of the application server. This project aims to compare the performance of a selected set of web server technologies and analyse the results based on various circumstances and conclude with which web server architecture should be used under what situations. The benchmarking of these various web technologies will be done using two popular benchmarking tools called ĄpacheBenchąnd "JMeter". The project also aims to compare the two benchmarking tools, analysing their features and results.

## 1.1 Purpose of the project

In the rapid development of Web today, many sites are faced with new problems, such as the problem of multiuser requests and high concurrency. The dynamic scripting language JavaScript has become enormously popular for client and is widely used in Web development. Node stands for one new technology in JavaScript. Node is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications [6]. Node.js uses an event-driven, non- blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [6]. Node.js popularity surveys performed by official website indicate that the average downloads are over 35,000 since the version 0.10 released in March 2013. Corporations are quickly realizing the importance of Node.js and five major PAAS providers have supported Node.js [4]. Nowadays, JavaScript has been the first popular language in GitHub with 177,352 repositories and growing [5].

Not long ago even a 2-second page response time was considered as an acceptable one. However, web users have become increasingly impatient when it comes to speed these days. Earlier, speed was considered a feature and now it is deemed a necessity. Additionally, technological innovation in mobile space has raised the bar for speed. Hence, speed makes a lot of economic sense now. A recent research found that 250-450 milliseconds are the numbers that decide the winner in the race for web speed. Research also indicates that the slower the site, the lesser would be the number of clicks and transactions performed on the site which would eventually result in the loss of users.

In order to perform these tests on the various Web technologies we will make use of the testing tools such as JMeter and ApacheBench. In addition to testing the various Web technologies, this project will also focus on the quality of the testing tools, the differences between them and if they produce different results under certain circumstances. Complex systems make increasing demands on web servers, high volumes can overwhelm systems if they are not scaled correctly. Multiple objects can interfere when one process is handling a request for a specific user. Fixes need to be identified early in the project so that server crashes and vulnerabilities can be caught in advance Clients have scalability concerns and we must warranty some level of scalability with industry accepted metrics. In order to achieve this, our web servers.

This paper focuses on the impact on Web performance from three different Web technologies: Node, PHP and Python-Django. The security and scalability issues are beyond the scope of the thesis. We mainly use the benchmark tests to analyse the web technologies. The main contributions of this thesis are listed as follows.

1. We consider web technologies such as Node, PHP and Django in our experiment and analyse their performance based on certain performance tests. Then we compare them making a

conclusion of which situation they ought to be used in and in which situation they outperform the other.

2. By means of benchmark tests we evaluate performance from an objective point of view. There is often a dual impact on Web server performance, from the CPU intensive requests, and from the number of users making requests. The research herein has taken each of these effects in account.

## 1.2 Goals of the project

The goal of this project is to determine under what circumstances do different web technologies perform better than the other and based upon the results making a conclusion in what situations they should be used. With the number of different web technologies available today, it becomes difficult to make a choice on what web technology should be used for a specific application. Whether it is a static web application, dynamic web application, web applications with content management systems such as WordPress and Drupal, social networking application, or CPU intensive web applications such as file compression, audio/video trans-coding. This project aims to determine if the different web technologies perform differently when used for some of the above purposes and find the best one for each case.

The rest of this research is organized as follows - Chapter 2 discusses the technologies and tools used in this project. Chapter 3 describes the test bed and environment configurations. Chapter 4 details the methodology and experimental design of tests. Chapter 5 presents and analyses the results of all tests. Chapter 6 makes a conclusion of the paper with a summary of study and a future direction.

# 2 Technologies and Toolset

## 2.1 Node

The official website (www.nodejs.org) defines Node as ą platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [1]."

Regardless, JavaScript is the world's most famous programming languages. If you have done any programming for the web, it's unavoidable. JavaScript, in view of the sheer reach of the web, has satisfied the ćompose once, run anywhere"dream that Java had back in the 1990s. Around the season of the Ajax insurgency in 2005, JavaScript went from being a toy language to something individuals wrote genuine and noteworthy projects with. A portion of the eminent firsts were Google Maps and Gmail, yet today there are a large group of web applications from Twitter to Facebook to GitHub.

JavaScript has for some time been the true standard for frontend side web development. While about all frontend code is composed in JavaScript, server-side development is a variety of choices between PHP, Java, and various different technologies. Life as a web engineer would be substantially more straightforward if a single language was utilized all around. Since JavaScript overwhelms in the browser, it bodes well to utilize it on the server too.

The idea of server-side JavaScript is not a new one. Netscape initially introduced JavaScript into the server world in 1994. Since that time, a lot of projects have endeavoured, and failed, to advance JavaScript as a server-side language. Execution, or scarcity thereof, restricted JavaScript from picking up a genuine a dependable balance in the server space.

Throughout the years, JavaScript has seen gigantic upgrades in performance. Because of its pertinence in the program, enormous players like Google have contributed a considerable measure of time and cash to make JavaScript as fast as possible. In 2009, Ryan Dahl of Joyent, put the large part of that recently discovered execution to great use on the server when he made the Node.js structure. Dahl assembled Node.js over Google's V8 JavaScript engine. V8 is a similar engine that has given Google Chrome its astounding JavaScript performance, and helped it turn into the most well-known browser on the planet.

# 2. TECHNOLOGIES AND TOOLSET

## Technical Details

- **Threading**

  Node.js works in a single threaded environment, utilizing non-blocking I/O calls, enabling it to support a huge number of simultaneous connections without incurring the cost of thread context switching [17]. The idea of sharing a single thread among every one of the requests that utilization the observer design is for building exceptionally concurrent applications, where any function performing I/O must utilize a callback. With a specific end goal to accommodate the single-threaded event loop, Node.js uses the libuv library that, utilizes a fixed-sized thread pool that is in charge of a portion of the non-blocking async I/O operations [18].

  A drawback of this single-threaded approach is that Node.js does not permit vertical scaling by increasing the amount of CPU cores of the machine it's running on while not using a further module, like cluster, StrongLoop process Manager, or pm2. However, developers can increase the default range of threads within the libuv thread pool; these threads are probably to be distributed across multiple cores by the server software system [19].

  Execution of concurrent tasks in Node.js is handled by a thread pool. the main thread decision functions post tasks to the shared task queue that threads within the thread pool pull and execute. Inherently non-blocking system functions like networking interprets to kernel-side non-blocking sockets, whereas inherently blocking system functions like file I/O run in an exceedingly block method on its own thread When a thread in the thread pool completes a task, it informs the main thread of this, which wakes up and execute the callback. As callbacks are handled synchronously on the main thread, long lasting computations and other CPU-bound tasks will freeze the whole event-loop till completion.

- **V8 Engine**

  V8 is the JavaScript execution engine built for Google Chrome and open-sourced by Google in 2008. Written in C++, V8 compiles JavaScript source code to native machine code instead of interpreting it in real time [18].

  Node.js makes use of libuv to handle asynchronous events. Libuv is an abstraction layer file system and network functionality on each Windows and POSIX-based systems like UNIX operating system, macOS, OSS on NonStop, and Unix.

  The core functionality of Node.js resides in a JavaScript library. The Node.js bindings, written

in C++, connect these technologies to each other and to the operating system.

- **Package Management**

  npm is an in-built package manager for the Node.js servers. It is utilized to install Node.js programs from the npm registry, sorting out the installation and administration of third-party Node.js programs. npm isn't to be mistaken for the CommonJS 'require()' statement. It is not utilized to load code; rather, it is utilized to install code and manage dependencies from the command line. The bundles found in the npm registry can extend from basic helper libraries, for example, Lodash to task runners suck as Gulp.

- **Unified API**

  Node.js may be combined with a browser, a database supporting JSON information (such as Postgres, MongoDB, or CouchDB) and JSON for a unified JavaScript development stack. With the difference of what were basically server-side development patterns like MVC, MVP, MVVM, etc., Node.js permits the reuse of the same model and service interface between client-side and server-side.

- **Event Loop**

  Node.js registers itself with the software system so as to be notified once a connection is created, and also the OS can issue a callback. inside the Node.js runtime, every connection could be a small heap allocation. historically, comparatively heavyweight OS processes or threads handled every connection. Node.js uses an event loop for scalability, rather than processes or threads [74]. In distinction to different event-driven servers, Node.js's event loop doesn't need to be known as explicitly. Instead callbacks are outlined, and also the server mechanically enters the event loop at the end of the callback definition. Node.js exits the event loop once there are not any any callbacks to be performed.

- **Non-blocking**

  The question of whether an operation is blocking or non-blocking refers to the fact that it must finish before the next operation begins. Non-blocking operations are said to be asynchronous and blocking operations are said to be synchronous. Node is non-blocking i.e. operations don't have to happen consecutively.

- **Scalability**

  Node.js utilizes a single threaded model with event callbacks. Events encourage the server to react in a non-blocking way and makes the server scalable compared to traditional servers which give restricted access to threads to deal with requests. Node utilizes a single threaded program to provide services to a substantially bigger number of requests than traditional servers like Apache HTTP Server.

- **Memory Management**

  Since Node is single-threaded, that implies that every one of your users will be sharing the same memory allocation. At the end of the day, unlike to in the browser, you must be mindful so as not to store user-specific information in closures where different connections can affect it.

### Middleware

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express provides a thin layer of fundamental web application features, without obscuring Node.js features [17].

Express is the most prominent framework for Node applications, and it highlights middleware utilizing continuation passing. When you need to run a similar code for conceivably a wide range of routes, the perfect place for that code is likely middleware. Middleware is a function that gets passed the request and response objects, alongside a continuation function to call, called next(). Envision that you need to add a requestId to each request/response pair with the goal that you can follow them back to the individual request when you're troubleshooting or debugging your logs for something.

You can write some middleware like this:

```
require('dotenv').config();
const express = require('express');
const cuid = require('cuid');

const app = express();

// request id middleware
const requestId = (req, res, next) => {
  const requestId = cuid();
```

```
  req.id = requestId;
  res.id = requestId;

  // pass continuation to next middleware
  next();
};

app.use(requestId);

app.get('/', (req, res) => {
  res.send('\n\nHello, world!\n\n');
});

module.exports = app;
```

## 2.2 PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML [18].

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does śomething"(in this case, output "Hi, I'm a PHP script!"). The PHP code is enclosed in special start and end processing instructions <?php and ?> that allow you to jump into and out of "PHP mode."

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve [18].

PHP began as a small open source venture that advanced as an ever-increasing number of people discovered how valuable it was. Rasmus Lerdorf released the primary rendition of PHP back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor"
- PHP is a server-side scripting language that is embedded with HTML. It is utilized to manage dynamic content, session tracking, databases even build whole web based e-commerce websites.
- It is coordinated with various well-known databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly fast in its execution, particularly when compiled as an Apache module on the Unix side. The MySQL server, once running, executes even extremely complex queries with tremendous results returned in record-setting time.
- PHP supports countless protocols, for example, POP3, IMAP, and LDAP. PHP included support for Java and distributed object architectures (COM and CORBA), making n-level improvement a plausibility for the first time.
- PHP syntax is C-Like.

PHP is essentially centred around server-side scripting, so you can do anything some other CGI program can do, for example, gather form information, produce dynamic page content, or send and

get cookies. In any case, PHP can do significantly more.

There are three main areas where PHP scripts are used -

- Server-side scripting - This is the most traditional and fundamental target field for PHP. In order to make this work you require three things: the PHP parser (CGI or server module), a web server and a web browser. You have to run the web server, with an associated PHP installation [19]. You can get to the PHP program output with a web browser, seeing the PHP page through the server. All these can keep running on your home machine if that you are simply experimenting with PHP.

- Command line scripting - You can make a PHP script to run with no server or program. You just need the PHP parser to utilize it in the appropriate way. This kind of use is perfect for scripts consistently executed utilizing cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can likewise be used for straightforward script processing tasks.

- Writing desktop applications - PHP is presumably not the absolute best language to make a desktop application with a graphical UI, yet if you know PHP exceptionally well, and might want to utilize some advanced PHP includes in your client-side applications you can utilize PHP-GTK to compose such programs. Additionally, you can build cross-platform applications along these lines. PHP-GTK is an expansion to PHP, not accessible in the fundamental distribution.

PHP can be used on all major operating systems, including Microsoft Windows, Mac OS X, Linux, many Unix variants, RISC OS, and probably others. PHP has support for almost all of the web servers today. This includes IIS, Apache, and many others. And this consists of any internet server which could make use of the FastCGI PHP binary, like lighttpd and nginx. PHP works as both a module, or as a CGI processor.

So, with PHP, you have the liberty of selecting an operating system and a web server. Furthermore, you also have the choice of using procedural programming or object-orientated programming (OOP), or a combination of them both.

With PHP, you aren't confined to output HTML. PHP's power includes outputting pictures, PDF documents or even Flash movies (with the usage of libswf and Ming) generated at the fly. You can additionally output easily any text, together with XHTML and another XML document. PHP can autogenerate these documents, and save them within the system, as opposed to printing it out, forming a server-side cache to your dynamic content material.

One of the strongest features in PHP is its support for a huge range of databases. Writing a database-enabled application is pretty easy with the help of one of the database specific languages (e.g. mysql), or using an abstraction layer like PDO, or connect to any database that supports the Open Database Connection standard popular via the ODBC extension. Other databases may additionally utilize cURL or sockets, like CouchDB.

PHP also has support for communicating with other services using protocols inclusive of IMAP, HTTP, LDAP, POP3, SNMP, NNTP, COM (on windows) and endless others. You may also open raw network connections and interact using every other protocol. PHP has aid for the WDDX complex information exchange among truly all web programming languages. When it comes to interconnection, PHP has support for instantiation of Java objects and the use of them transparently as PHP objects.

PHP has a powerful text processing feature, which incorporates the Perl compatible regular expressions (PCRE), and numerous expansions and functions to parse and get XML records. PHP standardises a large part of the XML augmentations on the strong base of libxml2, and expands the list of capabilities by including SimpleXML, XMLReader and XMLWriter support.

### 2.3 Python - Django

Django is a high level Python Web Framework enabling developers to write clean and pragmatic web applications. Django supports templating, routing, authentication, routing, basic database administation and many other features. Although Django can be used without a database, it comes packed with an object-relational mapper which describes the database layout in simple Python code. Models can be represented in many different ways using the data-model syntax. Django also has a clean and elegant URL scheme and does not put any ambiguities in URLS.

Django is written in the Python programming language. Python is arguably one of the easiest programming languages to read and understand. It uses natural language constructs like paragraph-like layout and indentation along with easy to learn syntax. It makes understanding the structure of the program and flow significantly easy compared to some other programming languages.

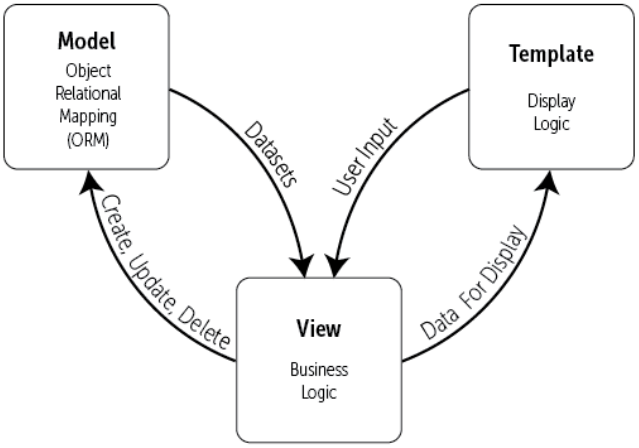
Django follows the Model-View-Controller approach. MVC is a design pattern aiming to separate web application into mainly three parts -

- **Model** - The model provides an interface between the database and containing application data. Django's Object-Relational Mapping provides the interface to the application database.

- **View/Template** - The view decides the information that should be presented to the user and also collects the information from the user. It acts as the interface between the client and the server. The template provides display logic and is the interface between the user and your

Django application

- **Controller/View** - The controller manages the business logic for the application by acting as an information broker between the view and the model. The view manages the bulk of the applications data processing, application logic and messaging.




This design philosophy allows developers to -

- Run servers separately for the database, media and applications.
- Easily have media served from a CDN (Content Delivery Network)
- Cache content at multiple scopes and levels
- For large web applications, it enables clustering and load-balancing in order to distribute the website across multiple servers

Web programming and web design are two very different ideas. For everything but the small prkects, the programming and design is not done by the same person, in many situations it is not even done by the same company. Django's template system makes it clear that Django programmers and websites must be able to work independent from each other. The profit is a plain text scripting language which makes use of tags to provide the presentation logic that decides what content needs to be displayed in the template.

DRY (Don't Repeat Yourself) is a term that often comes up, and it is one of Django's core principles. The DRY principle is evident in how Django makes use of template inheritance and helps reduce repetition and redundant code. Django comes with a large set of in-built tools that can be

used out of the box without prior setup requirement. It provides some common, however complex processes in the form of wrappers. Some of these packages include administration, authorization, database specific features, session management, syndication, etc. Django provides high security by disallowing code execution inside a webpage. This simple approach provides high security compared to other languages such as Javascript which can be executed within the browser and hence keeps the door open to various kind of vulnerabilities.

### 2.4 .NET

.NET Core is a general purpose, modular, cross-platform and open source implementation of the .NET Standard. It contains many of the same APIs as the .NET Framework (but .NET Core is a smaller set) and includes runtime, framework, compiler and tools components that support a variety of operating systems and chip targets. The .NET Core implementation was primarily driven by the ASP.NET Core workloads but also by the need and desire to have a more modern implementation. It can be used in device, cloud and embedded/IoT scenarios. [7].

.NET Core is supported by Microsoft on Windows, macOS and Linux. On Linux, Microsoft primarily supports .NET Core running on Red Hat Enterprise Linux (RHEL) and Debian distribution families. .NET Core currently supports X64 CPUs. On Windows, X86 is also supported. ARM64 and ARM32 are in progress.

Here are the main characteristics of .NET Core:

- **Cross-platform** - .NET Core provides key functionality to implement the app features you need and reuse this code regardless of your platform target. It currently supports three main operating systems (OS): Windows, Linux and macOS. You can write apps and libraries that run unmodified across supported operating systems. To see the list of supported operating systems, visit .NET Core roadmap.

- **Open source** - .NET Core is one of the many projects under the stewardship of the .NET Foundation and is available on GitHub. Having .NET Core as an open source project promotes a more transparent development process and promotes an active and engaged community.

- **Flexible deployment** - there are two main ways to deploy your app: framework-dependent deployment or self-contained deployment. With framework-dependent deployment, only your app and third-party dependencies are installed and your app depends on a system-wide version of .NET Core to be present. With self-contained deployment, the .NET Core version used to build your application is also deployed along with your app and third-party dependencies and can run side-by-side with other versions. For more information, see .NET

Core Application Deployment.

- **Modular** - .NET Core is modular because it's released through NuGet in smaller assembly packages. Rather than one large assembly that contains most of the core functionality, .NET Core is made available as smaller feature-centric packages. This enables a more agile development model for us and allows you to optimize your app to include just the NuGet packages you need. The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs in a pay-for-what-you-use model.

.NET Core is composed of the following parts:

- A .NET runtime, which provides a type system, assembly loading, a garbage collector, native interop and other basic services.

- A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities.

- A set of SDK tools and language compilers that enable the base developer experience, available in the .NET Core SDK.

- The 'dotnet' app host, which is used to launch .NET Core apps. It selects the runtime and hosts the runtime, provides an assembly loading policy and launches the app. The same host is also used to launch SDK tools in much the same way.

## 2.5 Database - PostgreSQL

PostgreSQL is an object-relational database management system that emphasizes on the principles of extensibility and standards compliance. The primary function is to store data in a secure manner and return the data as response when requests are made to the database from software applications. It is able to handle high workloads that range from small single-machine applications to very large-scale applications for purposes such as data warehousing that have many concurrent users. PostgreSQL requires mininum maintenance because it is very stable. Therefore applications developed on PostgreSQL have low total cost of ownership compared to other database management systems.

Some of the important features of PostgreSQL are as follows -

- User-defined types

- Table inheritance
- Sophisticated locking mechanism
- Views, rules, subquery
- Nested transactions
- Asynchronous replication
- Foreign key referential integrity
- Multi version concurrency control

PostgreSQL is standards compliant. Its implementation of SQL strongly conforms to ANSI-SQL:2008 standard. It supports all subqueries including subselects, it is read-committed and serializes transaction isolation levels. The data integrity features included in postgreSQL are primary keys, foreign keys with cascading updates and deletes as well as restrictions, check constraints, not null constraints and unique constraints. Some of the advantages of using PostgreSQL are -

- **Immunity to over-deployment** Over-deployment is a major problem in most proprietary database vendors such as Oracle SQL. But with PostgreSQL, since it has no associated licensing cost for using the software, there are no legal restrictions or compliances to follow.
- **Extensible** The source code for PostgreSQL is open-source. If a database administrator needs to customise or extend PostgreSQL, they are free to do so and require minimum effort with no added costs.
- **Cross platform** PostgreSQL is available on all the major UNIX operating systems as well as Windows and macOS via the cygwin framework.
- **Low staffing and maintenance cost** PostgreSQL is designed to have minimum maintenance and tuning requirements as compared to other proprietary databases, while providing all the features, performance and stability.

## 2.6 ApacheBench

ApacheBench (ab) is a single-threaded command line computer program for measuring the performance of HTTP web servers. Originally designed to test the Apache HTTP Server, it is generic enough to test any web server. The ab tool comes bundled with the standard Apache source distribution, and like the Apache web server itself, is free, open source software and distributed under the terms of the Apache License. [1] Apachebench generates a flood of queries to the specified URL and returns a variety of metrics. These metrics can be used to compare the performance of

various web technologies in our project. Some of the important metrics are requests/second, time per request, and transfer rate. The tool allows testers to load test a URL or server by replicating number of users and number of concurrent requests per second.

## 2.7 JMeter

JMeter is an Open Source testing software. It is 100% pure Java application for load and performance testing. jMeter is designed to cover categories of tests like load, functional, performance, regression, etc., and it requires JDK 5 or higher. This tutorial will give you great understanding on jMeter framework needed to test an enterprise level application to deliver it with robustness and reliability. [2]

JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance/functionality of the target server/application via tables, graphs, etc.

The protocols supported by JMeter are -

- **Web** - HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
- **Web Services** - SOAP / XML-RPC
- **Directory** - LDAP
- **Service** - POP3, IMAP, SMTP
- Database via JDBC drivers
- Messaging Oriented service via JMS
- FTP Service

### Features

- Being an open source software, it is freely available.
- It has a simple and intuitive GUI.
- JMeter can conduct load and performance test for many different server types: Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3, etc.
- It is a platform-independent tool. On Linux/Unix, JMeter can be invoked by clicking on JMeter shell script. On Windows, it can be invoked by starting the jmeter.bat file.
- It has full Swing and lightweight component support (precompiled JAR uses packages javax.swing.* ).

- JMeter store its test plans in XML format. This means you can generate a test plan using a text editor.
- Its full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- It is highly extensible.
- It can also be used to perform automated and functional testing of the applications.

### 2.8 DigitalOcean

DigitalOcean is an IaaS (Infrastructure as a Service) company that delivers fast and easy solutions for developers and bsuinesses to deploy their applications on the cloud. It accelerates software development so that the developers and programmers can spend more time working on the feature building and less tiem and focus on managing the infrastructure.

DigitalOcean virtual servers (VPS) also known as "droplets"use KVM (Kernel-based Virtual Machine) as hypervisor and so they can be created in a variety of different sizes. DigitalOcean has their data center in eight different regions around the world, and provides 6 GNU/Linux distributions and dozens of on-click setup applications. DigitalOcean also offers an additional feature of load balancing on their cloud servers. They provide a scalable infrastructure for programmers to build applications with fast development times.

DigitalOcean’s services are specifically created for all scales of applications, from individual based to enterprise based. They offer serveral different levels of cloud based hosting considering different requirements such as storage capacity, volume needs, and also billed either on a monthly or hourly basis. The more the features, will determite the amount of memory, disk space, transfer limit and core processors provided by the server in your chosen plan.

Setting up a DigitalOcean virtual server or "dropletćan be done in under a minute. With each droplet, the developer gets full root access to the server, with the ability to customize the server setup and choose the desired operating system.

All of DigitalsOcean’s plans include the following -

- Solid state drives (SSD)
- Simple account control panel
- DNS management
- Global image transfer (gives the developers the ability to load droplets in different datacenters)
- Private networking (different droplets belonging to you can communicate with each other with no restrictions on bandwidth limits)

### Technology and Infrastructure

DigitalOcean has a datacenter each in San Francisco, London, Singapore and three each in Amsterdam and New York. The datacenter in Singapore supports IPv6. Cloud servers are build using the KVM virtualization built on Intel's Hex-Core CPUs that have a dedicated ECC RAM and RAID SSD storage. However, the developer can use the control panel or DigitalOcean's name-spaced API to design their own. The control panel also provides a one-click install of common apps such as Linux distributions on the droplets, such as FreeBSD, CentOS, Ubuntu, Drupal or Wordpress. Security is one of the most important concerns when it comes to web hosting, but it becomes even more important on cloud hosting. DigitalOcean restricts access to the most critical systems, so the technical staff does not have any access to backend hypervisors, so only the engineering team can access snapshots of the storage systems as well as backups. The datacenters are protected physically by security staff that are working round-the-clock. Biometric readers and two factor authentication are also provided as a safety measure. DigitalOcean promises a 99.99% uptime, and offers credit if the downtime is below that level.

### Features

- **Teams** - This features allows the developers to share resources and accounts between multiple teams and projects. By inviting teams members to streamline a lot of the manual work. The team members can have various roles such as some responsible for managing resources and billing across multiple sites or apps, provide an overview to make sure that all the team members are using the best practices for security, etc.

- **Block Storage** - Whenever more storage is required, it is easy to extend additional storage to the droplets as per our requirement. The storage blocks are separate from the actual server droplet and have multiple copies that are spreat out throughout the network of servers to make sure that the data is protected in rare situations of hardware failure.

- **Resilient Network** - By making use of Tier - 1 bandwidth, DigitalOcean's worldwide network is optimized so that it delivers data wherever it has to go, with high speed and high security. It is possible to instantly deploy by using load balancers which improve the reliability of the droplet server.

- **Spaces** - In addition to block storage, DigitalOcean provides a cloud storage solutions called Spaces. It is used as a store for important data.

- **One-click Installs** - When setting up a droplet, it is possible to imstall many different types of applications, scripts or CMS systems with just one click instead of logging in to the console

and do it all from scratch.

# 3 Experimental Environment

## 3.1 Local Server Environment

The local servers are setup using the three different web technologies, namely Node, PHP and Django. Each of the servers are able to run simultaneously on the same machine on different ports. Below is the setup and configuration for each of the servers. The servers are setup using the http server module that comes bundled with each of the web technologies.

In order to keep the readings consistent, the local servers are setup on the same machine and are connected to the same network. The operating system on the local server machines is Windows 10, CPU speed 2.6 GHz and 16 GB RAM.

### Node Server

In order to setup the Node server, the following pre-requisites are required -

- NPM - Node package manager
- Node
- Express framework

Node package manager gives us access to open-source packages that aid in the development of Node applications. It comes bundled with the installation of Node.

Once we choose a folder in which the server files will reside, we run the terminal and execute the following command - ‘npm install express‘

This installs the express middleware framework that provides us with the necessary modules to develop a Node server. After the installation completes we observe a folder structure as follows -

```
|-node_modules
|-src
|---index.html
|---index.js
|-package.json
```

The node_modules folder contains all the necessary packages that express requires, some of the notable packages are the ’http-server’, ’babel’ etc. Te src files contains all the server related files, index.html is the homepage of the server that is returned when a request is made to the server. index.js is where the server code lives. The simplest method to fire a web server using Node and express is shown below -

```
var express = require('express');
var app = express();

app.get("/*", (req, res) => {
res.sendFile(path.join(__dirname, "index.html"));
});

app.listen(9000, () => console.log("Running on 9000"));
```

The above code fires a server that runs on localhost port 9000. When a request is made to the home page http://localhost:9000/ ('/' indicates homepage) the server receives the request and the app.get("/") function fires and sends a response with index.html file.

index.html is a simple static HTML file that gets served as the homepage when a request is made. Depending on the server and application the homepage can be configured as required.

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
</head>

<body>
<h1>This is API file</h1>
</body>

</html>
```

The homepage here simply displays the title 'This is API file' on the browser when a request is made for it.

The package.json file is where the configurations for the server and dependencies are configured. For the purpose of this project we have utilized the following dependencies - ćors", ęxpress", "pg"these are application specific dependencies.

- **cors** is a package that enables cross-origin resource sharing, which makes use of additional HTTP headers in order to let a user gain permission of resources that are on a different server, in our case to gain access to resources in the remote server configuration.

- **express** as discussed previously, is a Node.js framework for server configuration.
- **pg** is the package that enables connection to the postgreSQL database.

Additionally, Node also supports packages that are only used in the development phase, they are listed under the devDependencies object, some of the important devDependencies used in this project are -

- **body-parser** is a middleware which parses the incoming request's body header in req.body property.
- **babel-cli** used for compilation of different Javascript files written in the latest ECMAScript 6 in order to transpile them to ECMAScript 5 which is understandable by the browsers.
- **nodemon** watches the server files for any changes and restarts the server automatically when changes are made, hence it gets rid of the manual restarting of the server needed to be done by the developer.

The server is fired by running the script in the scripts object present in the package.json file.

```
"scripts": {
"start": "nodemon --exec babel-node -- src/index.js"
},
```

nodemon is set to watch the index.js file which is where our server code lives and updates the server if any changes are made to this file. babel-node executes the ECMAScript 6 code in index.js and transpiles it to ECMAScript 5 which is understandable to the web browser. When the server is ready we get the following message in the console if everything worked -

```
[nodemon] 1.12.1
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching: *.*
[nodemon] starting 'babel-node src/index.js'
Running on 9000
```

Therefore the server is now running and listening to requests on port 9000.

### PHP Server

Local PHP servers can be set up relatively easily in comparison to the other two web technologies considered in this project. XAMPP for Windows makes setting up a local server on the machine relatively easy. It provides a WAMP stack (Windows, Apache, MySQL and PHP). The steps involved in setting up a PHP server are as follows -

- **Creating Virtual Hosts**

  There are a number of ways to serve static PHP files from the server. One way is to simply keep all server related files within the 'htdocs' folder in XAMPP. However, the drawback of this is that we have to enter long URLs for every page that we want to visit. Instead we can create a virtual host for each of our sites which have different URLs. The virtual hosts set up is located in the following file:

  ```
  /xampp/apache/conf/extra/httpd-vhost.conf
  ```

  At the bottom of the file append the following:

  ```
  NameVirtualHost 127.0.0.1:80

  <VirtualHost 127.0.0.1:80>
  DocumentRoot "C:/xampp/htdocs/"
  ServerName localhost
  </VirtualHost>
  ```

  This sets the root of localhost to 'C:/xampp/htdocs'. So all the server calls made to localhost will look for the url in this folder.

- Now all we have to do is run the XAMPP control panel and start the Apache server. This fires the PHP server on port 80 and serves the PHP files located inside the 'C:/xampp/htdocs' folder. If we navigate to 'http://localhost/' in our web browser, the server returns the index.php file since it is the homepage. The xampp/htdocs/ folder structure is as follows -

  ```
  |-img
  |-xampp
  |-index.php
  |-favicon.ico
  ```

  The 'img' folder contains all the static images that are used by the web application, favicon.ico is the icon that is visible on the title bar of the browser when we navigate to the website. index.php is the homepage of the website and is returned when a request is made from the browser.

### Django Server

Django is extremely versatile in terms of how it can be installed and configured. Django can be:

- Installed from source, PyPi (Python Package Index) or by using package manager applications in UNIX operating systems.
- Installed on various operating systems.
- Configured to use several of the popular databases.
- Run within a Python virtual environment or within the system environment.

In order to install and setup Django, we need the following -

- **Python** - The Python version used in this project is version 3.6
- **pip** - Python package manager used to install Django.

Once we have these two requirements fulfilled, we simply need to install Django. It is done in the command prompt (on Windows) or terminal (on Linux) by running the following command -

```
pip install django
```

The django version used in this project is 2.0.2. Once everything is installed, we need to create our server, this is done in the terminal by running -

```
django-admin startproject projectname
```

Where project name is the name of the project. This creates a folder with the name provided as the projectname, with the following structure:

```
|-projectname
|---settings.py
|---urls.py
|-applicationname
|---migrations
|---views.py
|---urls.py
|---models.py
|-manage.py
```

The manage.py is the main file through which the server is built. It puts our project's package into the sys.path and also sets the DJANGO_SETTINGS_MODULE environment variable to make it point to our project's settings configuration file.

```
#!/usr/bin/env python
import os
import sys
```

```
if __name__ == "__main__":
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "projectname.settings")
try:
from django.core.management import execute_from_command_line
try:
import django
execute_from_command_line(sys.argv)
```

The projectname/settings.py file contains all the configurations for our server, such as database settings, application specific settings, middleware configurations, template routing, authentication handlers etc. projectname/urls.py manages the routing of the website. The ‘urlpatterns‘ list routes URLs to views.

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
url(r'^admin/', admin.site.urls),
url(r'^$', include('fibonacci.urls')),
url(r'^api/fibonacci/', include('fibonacci.urls')),
url(r'^api/books/', include('books.urls'))
]
```

Different applications can be run on the same server. Django provides separation of each application into it’s own environment which is then managed by the main server that sends the correct response as per the incoming requests. To create an application we run the following command in terminal -

```
python manage.py startapp applicationname
```

This creates a new application in our Django environment. The application has it’s own views, models and urls for routing purposes.

```
|-applicationname
|---migrations
|---views.py
|---urls.py
|---models.py
```

The urls.py file contains the routing for the respective application. Thus when it gets a request from the user, it checks the url pattern and routes the appropriate view to be sent as a response.

```
from django.conf.urls import url
```

```
from . import views

urlpatterns = [
url(r'^$', views.index, name='index'),
]
```

Django processes an incoming request in the following way -

- Django determines which root URLconf package to use. Usually, the value is of the ROOT_URLCONF setting but when the incoming HttpRequest object already has a urlconf which would be set by a middleware, it will be replaced and it's value will be used instead of the ROOT_URLCONF setting.

- Django loads the Python module and looks for the variable called 'urlpatterns'. This is a Python list of django.urls.path() or django.urls.re.path() instance.

- Django checks through each URL pattern in order, and when it finds the first one that matches the requested URL it stops.

- Once a URL pattern match is found, Django imports the given view declared in the url. The view is simply a Python function. It gets the following arguments -

  - The HttpRequest instance

  - A match of the regular expression if the marched URL pattern returns no named groups

A view is a Python function that takes the web request from the client and returns a response. The contents of the response can be HTML, a redirect, an error 404, image, or any other piece of information. It contains the arbritrary logic necessary to return a response. The views.py file is where the logic for each route is written. Each function inside the views.py file represents a route and is executed when the urls.py file finds a matching route to execute and send as a response to the requesting client.

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
now = datetime.datetime.now()
html = "<html><body>It is now %s.</body></html>" % now
return HttpResponse(html)
```

- First, we import the HttpResponse class from the django.http package, and Python's datetime package

- We define a function which acts as the view function (current_datetime in this case). Each view function take a request object as its first parameter.
- The view returns a response in the form of HttpResponse object which contains the response. Every view function must return an HttpResponse object.

A model in Django is a single source of information about the data in our database. It contains the information such as fields and behaviour of the data. Usually each model maps to a single database table.

- Each model is a Python class of django.db.models.Model
- Each attribute in the model represents a data field in the database
- Enables automatically generated database access API

Below is an example of a model called 'Books' which has values first_name and last_name

```
from django.db import models

# Create your models here.
class Books(models.Model):
name = models.CharField(max_length=255)
```

name is the field in the Books model. Each field specifies as a class attribute. They are attributed as database columns.

The Books model above would translate to the following SQL query:

```
CREATE TABLE myapp_books (
"id" serial NOT NULL PRIMARY KEY,
"name" varchar(255) NOT NULL,
);
```

### 3.2 Remote Server Environment

The remote server setup is done using a cloud hosting platform DigitalOcean. As discussed in the previous chapter, DigitalOcean provides it's subscribed clients with virtual private servers (also known as droplets). In this section, each of the remote servers are set up on different virtual private networks located in the same data center in order to minimalize the effect of distance between client and server.

The remote servers are all set up on 64-bit Ubuntu 16.04 machines, with 20 GB SSD drive, 1 CPU, 512 MB RAM and 512 Mb/s bandwidth connection.

The servers are hosted on the following IP addresses -

- **Node** - 46.101.216.250
- **PHP** - 46.101.172.41
- **Django** - 46.101.169.55

Connection to the remote servers is done via SSH in the terminal in the following way -

- In the terminal we establish an SSH connection to the remote server through 'root' user which is predefined when the VPS is created

```
Dennis@DESKTOP-6DGK559 MINGW64 ~
$ ssh root@46.101.172.41
```

- It will prompt for the password

```
Dennis@DESKTOP-6DGK559 MINGW64 ~
$ ssh root@46.101.172.41
root@46.101.172.41's password:
```

On successful login, connection to the remote server will be established and we can configure the server as per our requirements.

### Transferring files to remote servers

There are two ways in which we can transfer our files to the remote servers. One is by using Git and the other is a file transfer software called Filezilla.

### Git

Git, a distributed version control system that syncs repositories across different machines and Github to track code remotely. Creating a git repository is relatively easy. In this project, the Node remote server is setup using Git. In order to transfer files from a local machine, it needs to be located inside a directory with git initialized and synced with Github. This can be done by running the following command in the terminal -

```
git init
```

This command creates a git repository, with a .git directory with subdirectories for objects, refs/tags, refs/heads, template files etc. Now, any change that is made inside the directory is watched by git. It maintains a compared list between new versions and old version of the directory. The command 'git status' gives us the current status of the directory compared to the previous saved state.

In order to sync new changes to the repository, the command 'git commit' is used. A commit is a record of what files have been changed since the last commit or initial commit(git init). Commits help us to control the state of our code allowing us to rollback to a certain commit if required.

For the purpose of sharing code between the local and remote servers, Github enables developers to do so. By uploading the local git repository to Github we can share our code. This is done in the following way -

- First we add a remote git repository, this is done in the following way -

  ```
  git remote add origin https://github.com/username/repositoryname.git
  ```

  This tells the local git repository to sync with the remote repository on Github located in the specified URL.

- Once the remote repository is saved, push the current local repository and all it's files to the Github repository -

  ```
  git push -u origin branchname
  ```

  origin indicates the remote repository to push to and branchname is the branch we wish to upload.

- The code is now available on the remote Github repository and can be shared with any other computer.

Once the code is uploaded to the remote Github repository, it is time to download the code into the remote server.

- Login to the remote server via SSH using the appropriate IP address, username and password.

- In order to clone the repository into our remote server machine, we use the 'git clone' command

  ```
  git clone https://github.com/username/repositoryname.git
  ```

  This clones the git repository into the directory in the remote server. Now we can use the code the same way we did in the local machine.

### Filezilla

Filezilla is a cross-platform FTP, SFTP and FTPS client with an intuitive graphical user interface. It allows transfer of files from one machine to another view the file transfer protocol. The server files for PHP and Django are transferred to their respective remote servers using the Filezilla client.

The main features of Filezilla are:

- Allows transfer of files in FTP, SFTP and also encrypted protocols of FTPS and SFTP
- Supports IPv6, the latest internet protocol version
- Directory comparison to compare local files and server files within the same directory. It also provides highlighting of missing or mismatched files
- Ability to configure network settings
- Support for remote file editing
- HTTP/1.1, SOCKS5 and FTP-Proxy support

Compared to git, transferring files via Filezilla is relatively easier and quicker to achieve. The user can directly login to the remote server within Filezilla using the IP address, username and password of the remote server, on success browse the directories in the server and make file transfers.

## 3.3 Testing tool

Apachebench is used to load test both the local and the remote servers. It provides most of the key metrics required to make our comparison of the performance of the various web technologies discussed in this project. The nature of benchmark testing is http based and Apachebench provides the functionalities of making requests with multiple users as well as with concurrency. Apachebench generates a flood of queries to the specified URL and returns a variety of metrics. These metrics can be used to compare the performance of various web technologies in our project. Some of the important metrics are requests/second, time per request, and transfer rate. The tool allows testers to load test a URL or server by replicating number of users and number of concurrent requests per second.

Apachebench comes bundled with the XAMPP installation required for PHP. In order to run Apachebench we need to navigate to the folder /xampp/apache/bin where the ab.exe file is located. Next, we load up the terminal and execute the command

```
ab -n numberOfUsers -c concurrency http://siteToBenchmark.com/
```

where numberOfUsers is an integer value which takes the number of users to benchmark the URL with and concurrency indicates the number of simultaneous users making the request to the server.

# 4 Test Methodology

The experiment evaluates the results of the server by performing benchmark tests. In all the tests, we follow the one-factor-at-a-time experimental design [3] to ensure the accuracy and effectiveness of tests.

## 4.1 Performance Testing

Performance Testing is a type of testing to ensure software applications will perform well under their expected workload. Features and Functionality supported by a software system is not the only concern. A software application's performance like its response time, reliability, resource usage and scalability do matter. The goal of Performance Testing is not to find bugs but to eliminate performance bottlenecks.

The focus of Performance Testing is checking a software program's:

- Speed - Determines whether the application responds quickly
- Scalability - Determines maximum user load the software application can handle.
- Stability - Determines if the application is stable under varying loads

Performance testing is popularly called as "Perf Testingąnd is a subset of performance engineering. Performance testing is done to provide stakeholders with information about their application regarding speed, stability and scalability. More importantly, performance testing uncovers what needs to be improved before the product goes to market. Without performance testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability. Performance testing will determine whether or not their software meets speed, scalability and stability requirements under expected workloads. Applications sent to market with poor performance metrics due to non existent or poor performance testing are likely to gain a bad reputation and fail to meet expected sales goals. Also, mission critical applications like space launch programs or life saving medical equipments should be performance tested to ensure that they run for a long period of time without deviations.

### Common Performance Problems

Most performance problems revolve around speed, response time, load time and poor scalability. Speed is often one of the most important attributes of an application. A slow running application will lose potential users. Performance testing is done to make sure an app runs fast enough to keep a user's attention and interest. Take a look at the following list of common performance problems and notice how speed is a common factor in many of them:

- Long load time - Load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.

- Poor response time - Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input. Generally this should be very quick. Again if a user has to wait too long, they lose interest.

- Poor scalability - A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users. Load Testing should be done to be certain the application can handle the anticipated number of users.

- Bottlenecking - Bottlenecks are obstructions in system which degrade overall system performance. Bottlenecking is when either coding errors or hardware issues cause a decrease of throughput under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is to find the section of code that is causing the slow down and try to fix it there. Bottle necking is generally fixed by either fixing poor running processes or adding additional Hardware. Some common performance bottlenecks are CPU utilization Memory utilization, Network utilization, Operating System limitations, Disk usage.

### Performance Testing Process

The methodology adopted for performance testing can vary widely but the objective for performance tests remain the same. It can help demonstrate that your software system meets certain pre-defined performance criteria. Or it can help compare performance of two software systems. It can also help identify parts of your software system which degrade its performance.

Below is a generic performance testing process -

- **Identify your testing environment** - Know your physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before you begin the testing process. It will help testers create more efficient tests. It will also help identify possible challenges that testers may encounter during the performance testing procedures.

- **Identify the performance acceptance criteria** - This includes goals and constraints for throughput, response times and resource allocation. It is also necessary to identify project success criteria outside of these goals and constraints. Testers should be empowered to set performance criteria and goals because often the project specifications will not include

a wide enough variety of performance benchmarks. Sometimes there may be none at all. When possible finding a similar application to compare to is a good way to set performance goals.

- **Plan and design performance tests** - Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
- **Configuring the test environment** - Prepare the testing environment before execution. Also, arrange tools and other resources.
- **Implement test design** - Create the performance tests according to your test design.
- **Run the tests** - Execute and monitor the tests.
- **Analyze, tune and retest** - Consolidate, analyze and share test results. Then fine tune and test again to see if there is an improvement or decrease in performance. Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU. Then you may have the consider option of increasing CPU power.

## 4.2 Benchmark Test Methodology

Benchmark Testing is a type of testing done to give repeatable set of quantifiable result from which present and future software releases for specific functionality can be baselined or compared. It is a process used to compare the performance of software or hardware system also known as SUT (System Under Test). An SUT can be a Web-based application.

A benchmark must be repeatable. For instance, with every iteration of load test, if the response times varies too much, system performance be benchmarked. Response time needs to be stable amongst different load conditions.

A benchmark must be quantifiable. For example, user experience cannot be quantified in numbers, but time a user spends on a webpage due to good UI can be quantified.

### Importance of benchmark testing

At business level, benchmark testing can be helpful in determining

- How well a web-based application is performing with respect to the competitors
- It ensures that websites complies with standards and best practices
- It enables to evaluate third- party service providers prior to making a contracting decision
- Allows to figure out the mistakes to be avoided

- How different types of customers experience the response time and availability of a site.

## 4.3 Phases of benchmark testing

### Planning Phase

- Identifying and prioritize standards and requirements
- Decide benchmark criteria
- Define benchmark test process

### Analysis Phase

- Identify root cause of error to improve quality
- Setting goals for test process

### Integration Phase

- Share outcomes with concerned person and get approval
- Establish functional goals
- Define benchmark test process

### Action Phase

- Develop test plan and documentation
- Implement actions specified in previous phases and monitor progress
- Run the process continuously

## 4.4 Test setup

The benchmark test will be performed in three separate scenarios - CPU intensive task on local server, CPU intensive task on remote server and a SELECT database query on local server with a local database setup.

The remote servers are hosted on DigitalOcean web server platform, with each technology having it's own dedicated server.

## 4.5 Testing the web technologies

According to one-factor-at-a-time experimental design, we make three fundamental tests - Calculate Value of Fibonacci and Fetch data from local DB and Fetch data from remote DB. Calculate

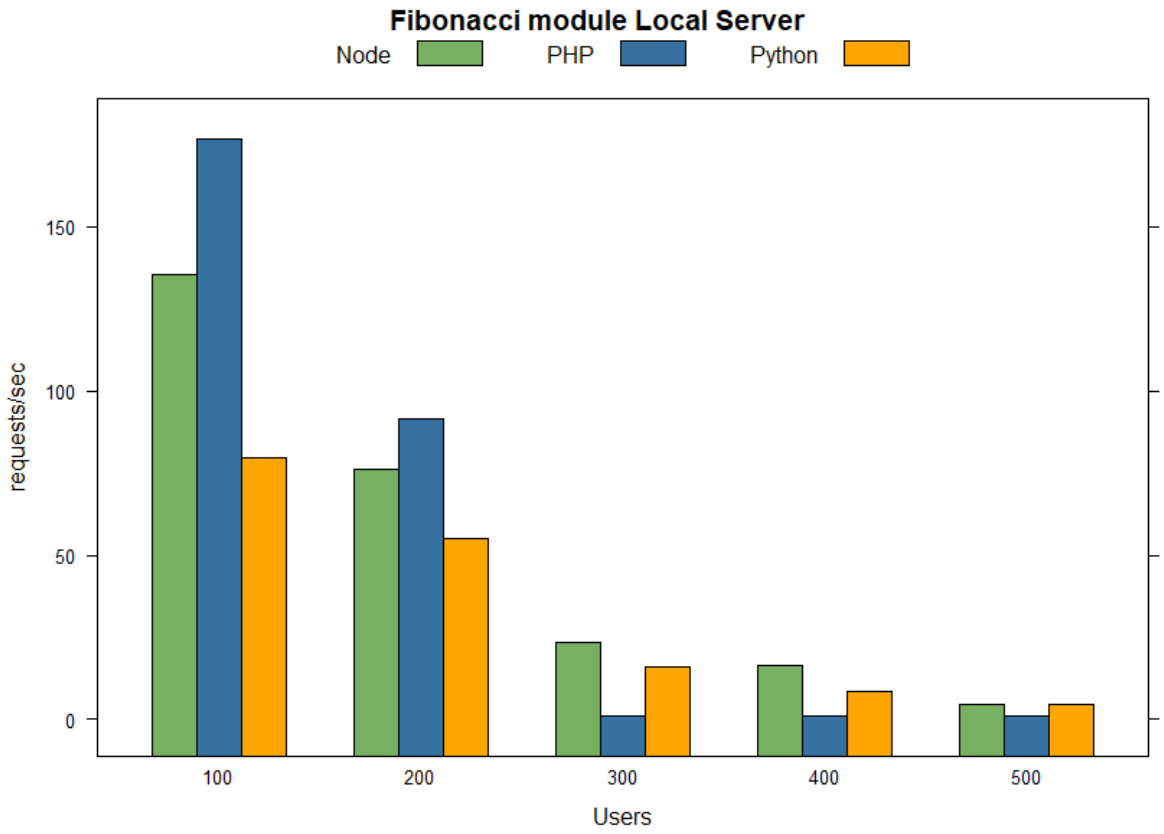
# 4. TEST METHODOLOGY

Value of Fibonacci module calculates some value of Fibonacci and evaluates the performance under compute-intensive tests. Select Operation from local DB module compares different performance through querying some value of DB in an IO-intensive situation. Under all benchmark tests, we keep requests 1000 and vary users from 10 to 1000. TABLE 1 summarizes the factors in our experiments.

**Table 1:** Benchmark test factors

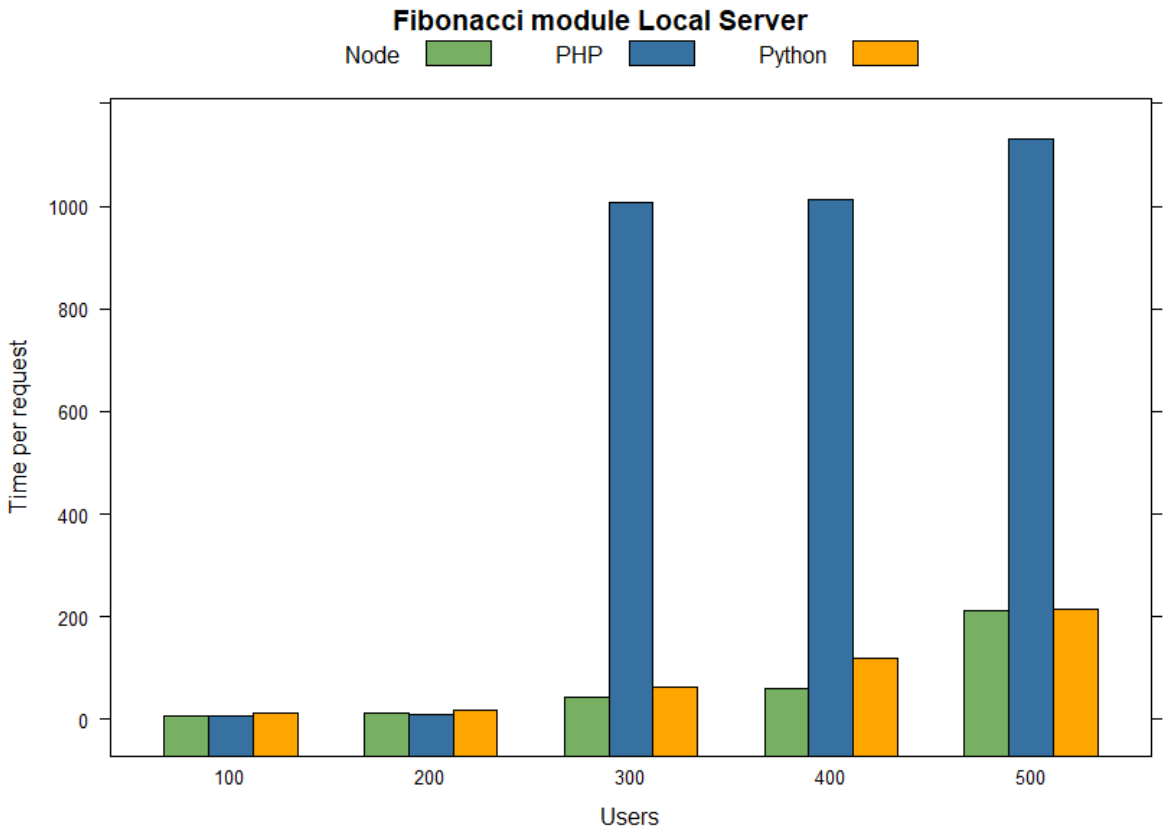
| | |
|---|---|
| Users | 10, 100, 200, 300, 400, 500 |
| Requests | 200, 500, 700, 1000 |
| Benchmark test module | Calculate fibonacci, SELECT local DB, remote DB |
| Web technologies tested | Node, PHP, Python-Django |

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

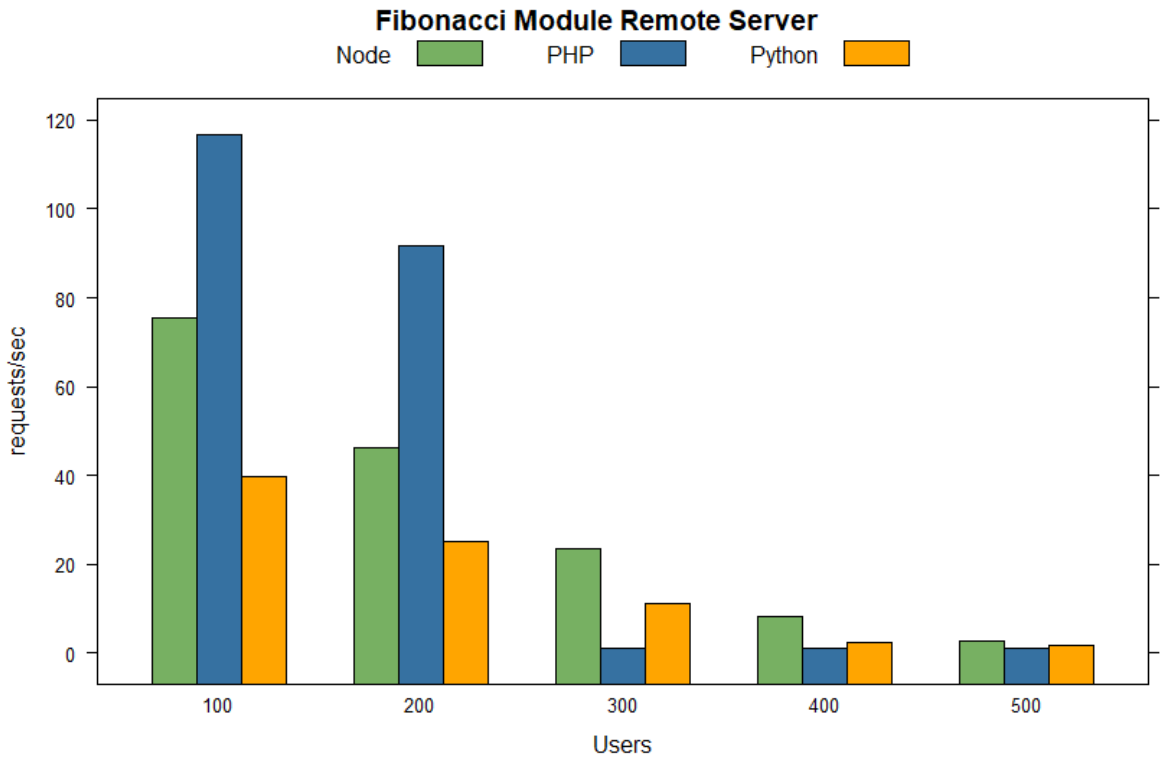



**Rysunek 1:** Requests per second, Fibonacci module on local server

# 4. TEST METHODOLOGY

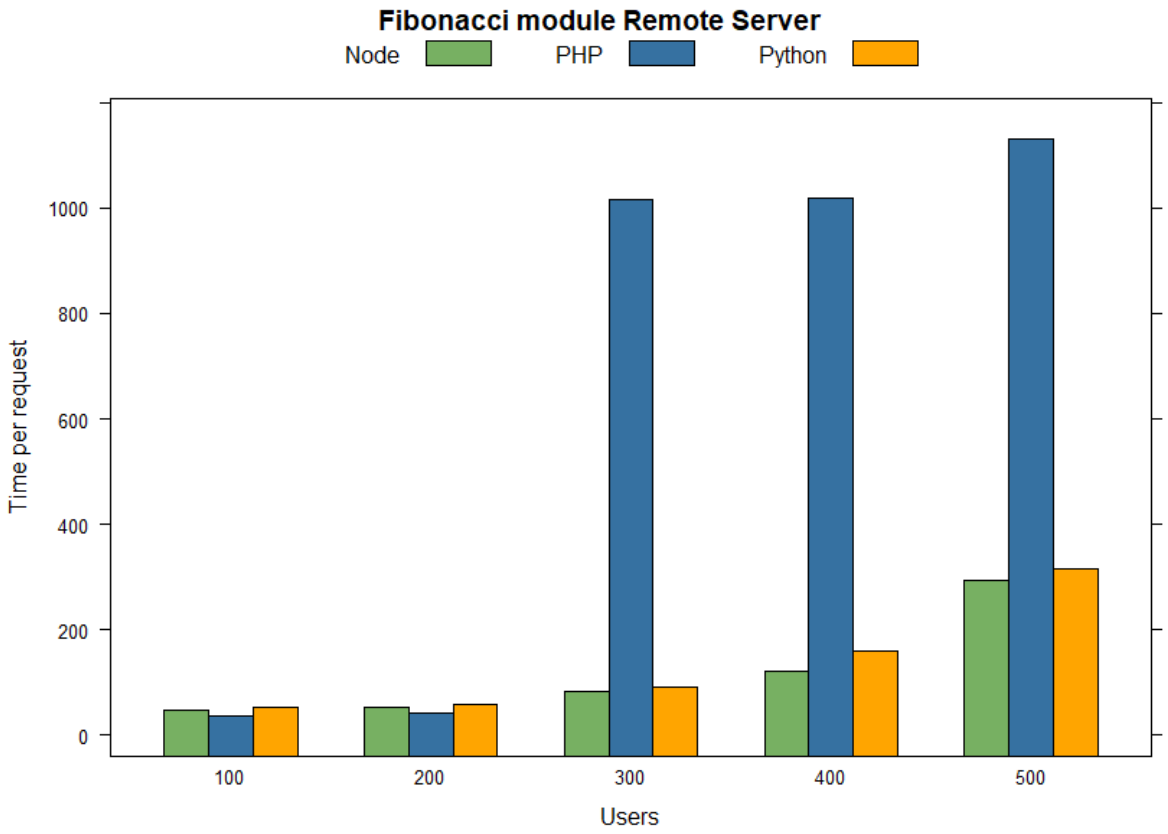



**Rysunek 2:** Time per request, Fibonacci module on local server

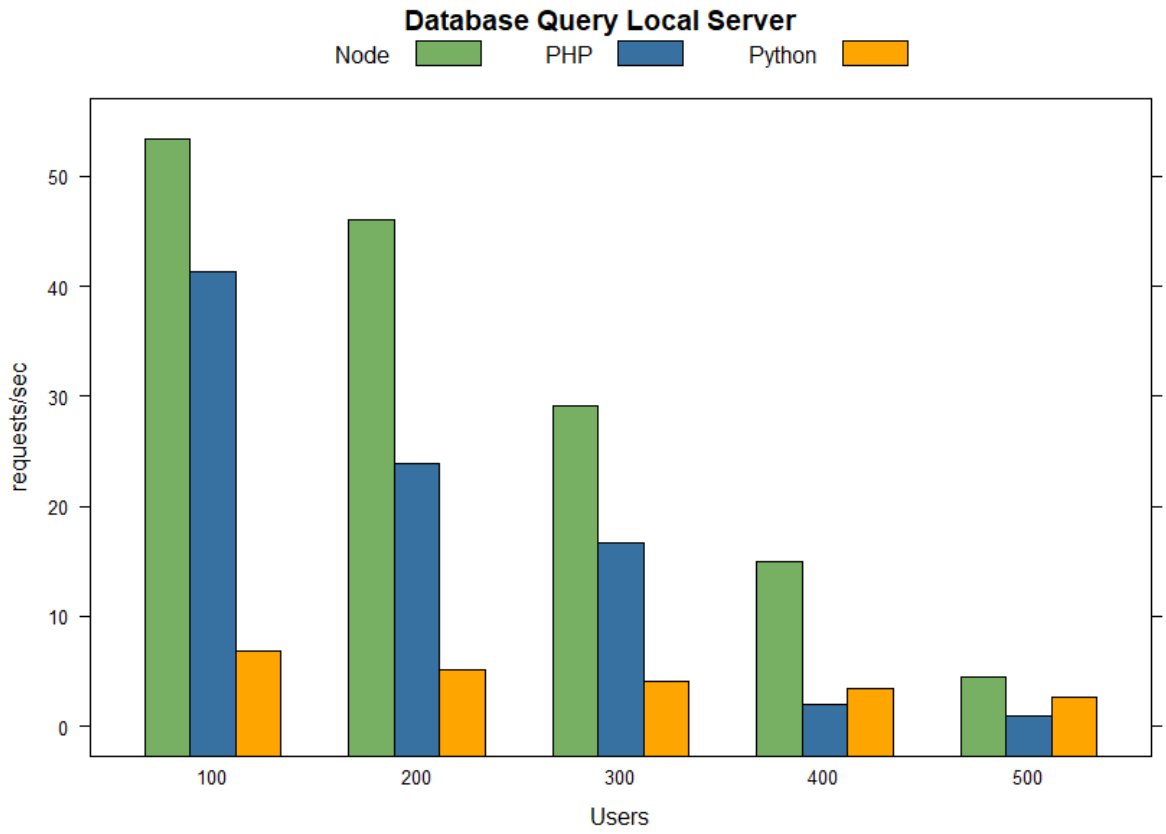



**Rysunek 3:** Requests per second, Fibonacci module on remote server

# 4. TEST METHODOLOGY

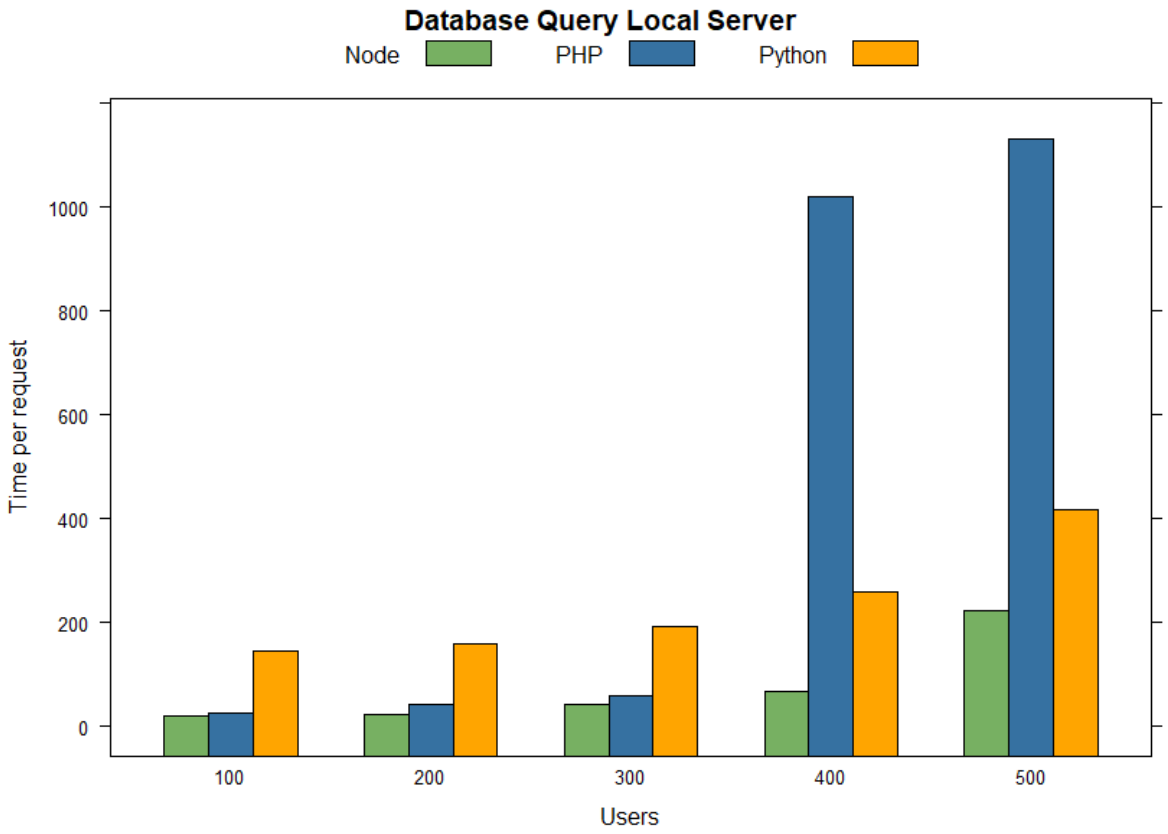



**Rysunek 4:** Time per request, Fibonacci module on remote server




**Rysunek 5:** Requests per second, Database query on local server

# 4. TEST METHODOLOGY




**Rysunek 6:** Time per request, Database query on local server

# 5 Conclusion

On comparing the two performance testing tools, namely JMeter and Apachebench, we can observe that they yield the similar results for the respective web technologies.

Since Node is event-driven and single threaded, it performs better in situations that involve a lot of I/O operations, however it's performance suffers when we compare it to more cpu-intensive such as database calls tasks.

PHP performs well when the number of requests is relatively low, but as the number of requests begins to increase, it's performance drops exponentially.

## Literatura

[1] Apachebench. 16

[2] Jmeter. 17

[3] S.tilkov, s.vinoski, “node.js: Using javascript to build high- performance network programs”, ieee internet computing, 2010. 32

[4] Nodejs documentation, 2017. 3

[5] Number of javascript repositories on github, 2017. 3

[6] Joseph Ottinger. What is an app server?, 2008. 3

[7] Maira Wenzel. .net core, 2017. 14