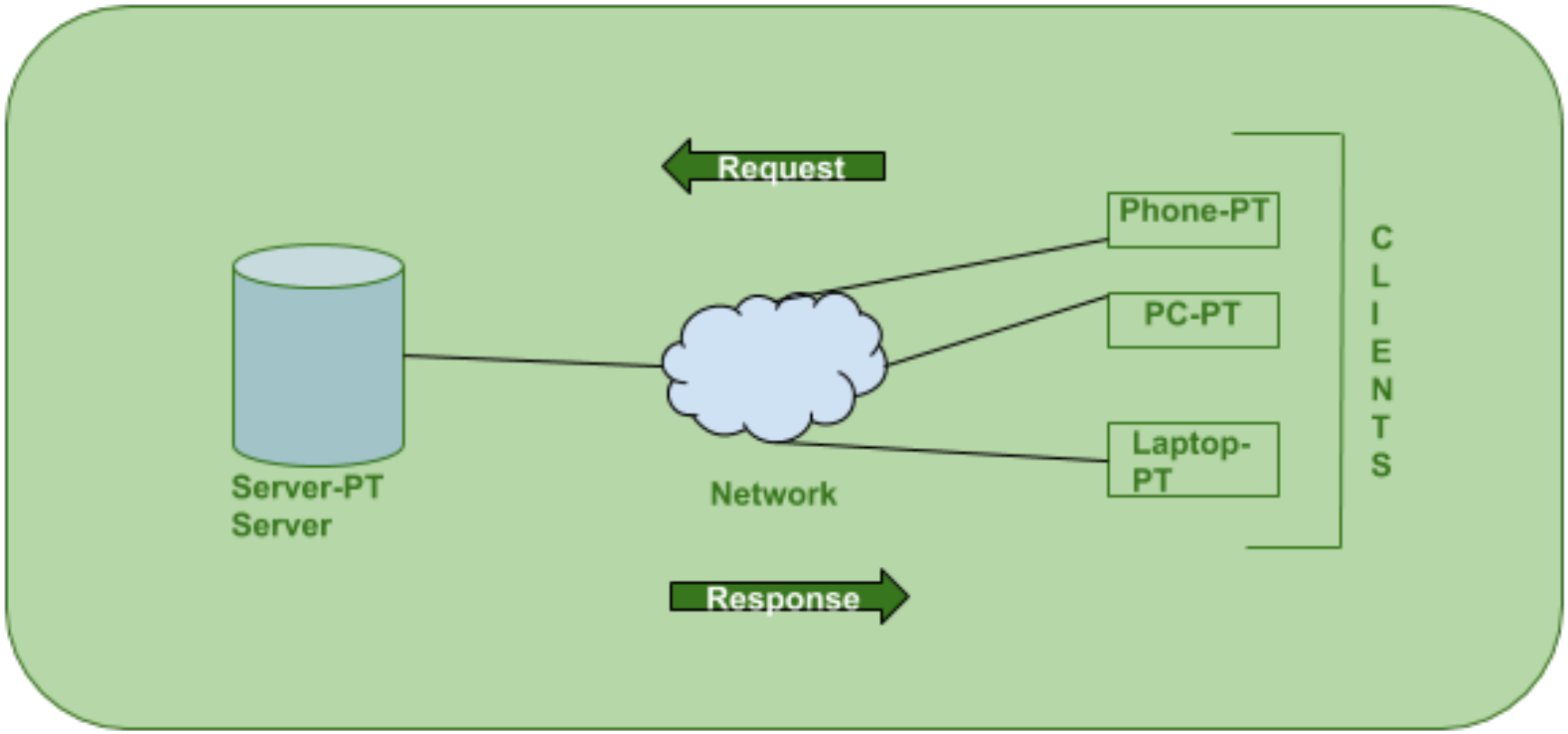
Client-Server Model

- Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients.
- In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.
- **Client**: When we talk the word Client, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a Client is a computer (Host) i.e. capable of receiving information or using a particular service from the service providers (Servers).
- **Servers**: Similarly, when we talk the word Servers, It mean a person or medium that serves something. Similarly in this digital world a Server is a remote computer which provides information (data) or access to particular services.
- So, its basically the Client requesting something and the Server serving it as long as its present in the database.

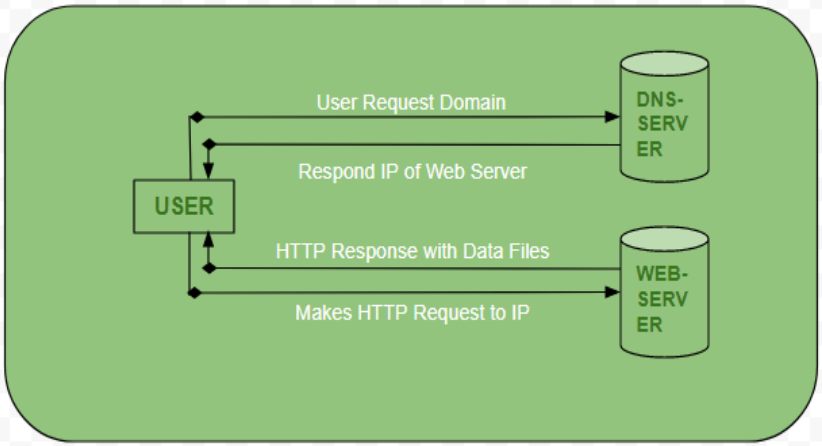
Fenil Shah


Request
Phone-PT
PC-PT
Laptop-PT
CLIENTS
Server-PT
Server
Network
Response

Fenil Shah

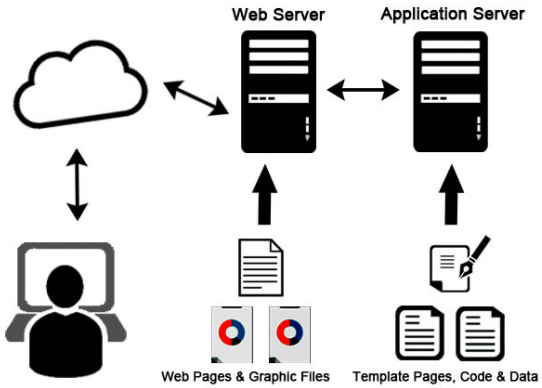
Browser interacts with the servers?

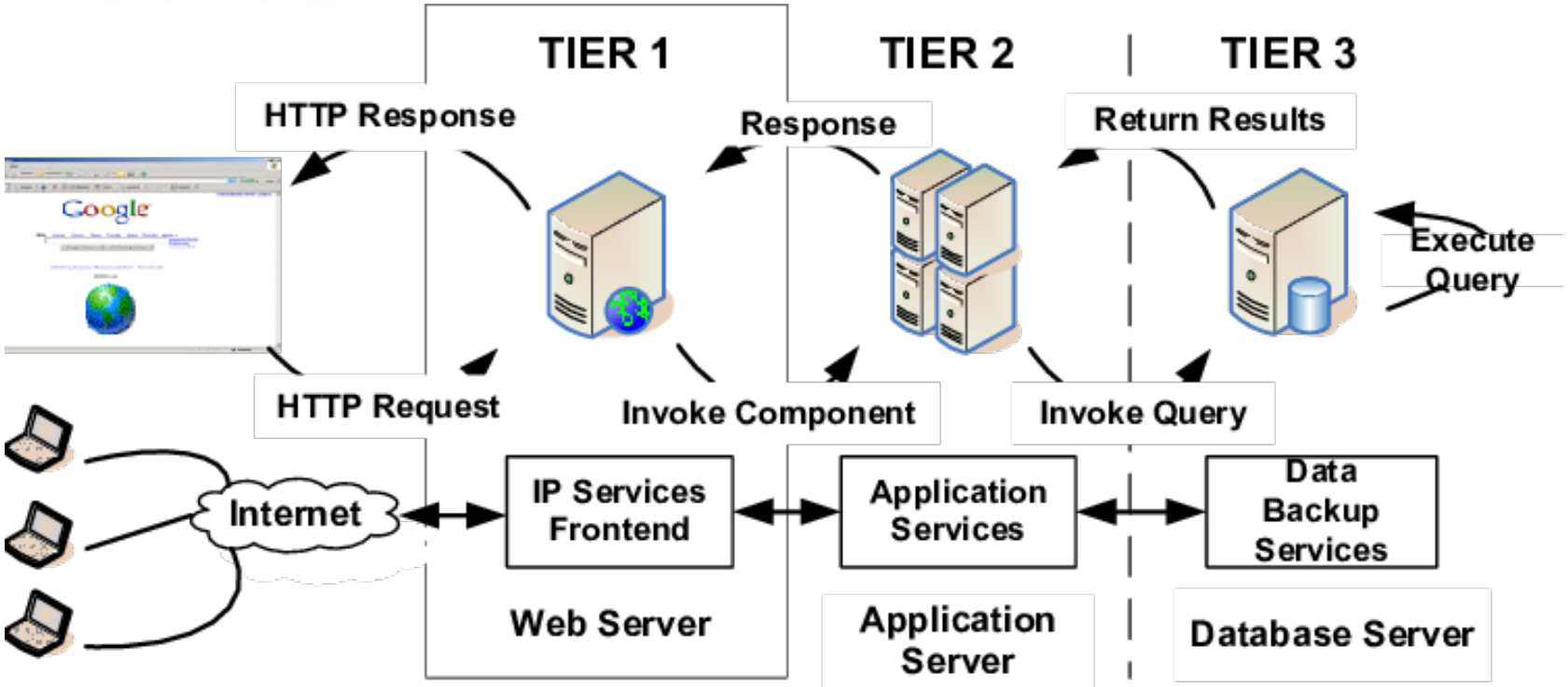
- User enters the URL(Uniform Resource Locator) of the website or file. The Browser then requests the DNS(DOMAIN NAME SYSTEM) Server.
- DNS Server lookup for the address of the WEB Server.
- DNS Server responds with the IP address of the WEB Server.
- Browser sends over an HTTP/HTTPS request to WEB Server's IP (provided by DNS server).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of DOM (Document Object Model) interpreter, CSS interpreter and JS Engine collectively known as the JIT or (Just in Time) Compilers.
- Eg – Search amazon.in
- E.g. – Search browser in google – GetRequest
- Tracert google.com



Application Server

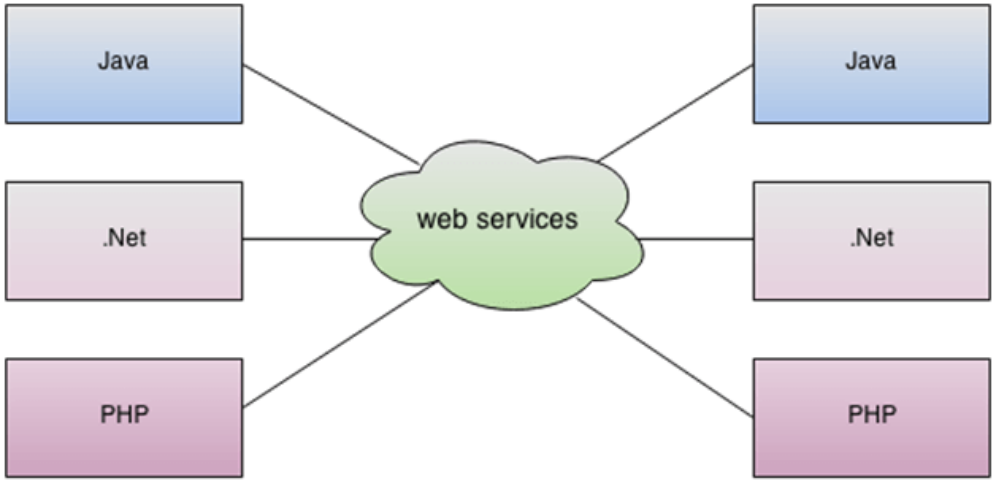
- Application Server is a type of server designed to install, operate, and host applications. An application server is a program that resides on the server-side, and it's a server programmer providing business logic behind any application. This server can be a part of the network or the distributed network. Used in dynamic websites.
- Tier 1 – This is a GUI interface that resides at the client end and is usually a thin client (e.g. browser)
- Tier 2 – This is called the middle tier, which consists of the Application Server.
- Tier 3 – This is the 3rd tier which is backend servers. E.g., a Database Server.
- Provides a mechanism for dealing with all the components and running services like session management, synchronous and asynchronous client notifications.
- Becomes very easy to install applications in one place, enables the ability to distribute requests to different servers based on their availability via Loadbalancing, provides security to applications, enables fault tolerance with the ability to recover/failover recovery, etc E.g.- Jboss, Weblogic, Websphere, Glassfish, Tomcat Server,etc


Web Server
Application Server
Web Pages & Graphic Files
Template Pages, Code & Data


TIER 1
TIER 2
TIER 3
HTTP Response
Response
Return Results
Execute Query
HTTP Request
Invoke Component
Invoke Query
Internet
IP Services Frontend
Application Services
Data Backup Services
Web Server
Application Server
Database Server

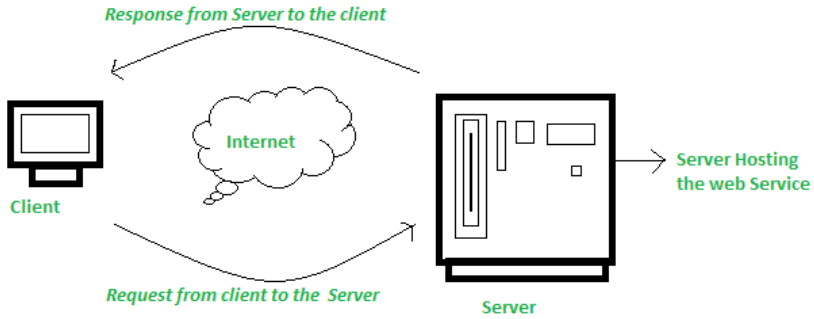
Webservice

- It is a client-server application or application component for communication. The method of communication between two devices over the network. It is a software system for the interoperable machine to machine communication. It is a collection of standards or protocols for exchanging information between two devices or application.
- E.g.- Java, .net, and PHP applications can communicate with other applications through web service over the network. For example, the Java application can interact with Java, .Net, and PHP applications. So web service is a language independent way of communication.



Webservice

- Web service is a software module that is intended to carry out a specific set of functions. web service would be able to deliver functionality to the client that invoked the web service.
- Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) – across the internet is considered a web service.
- Web services have the advantage of allowing programs developed in different languages to connect with one another by exchanging data over a web service between clients and servers. A client invokes a web service by submitting an XML request, which the service responds with an XML response.



Fenil Shah

Advantage of Webservices

- (a) **Business Functions can be exposed over the Internet**: A web service is a controlled code component that delivers functionality to client applications or end-users. This capability can be accessed over the HTTP protocol, which means it can be accessed from anywhere on the internet. web service can be located anywhere on the internet and provide the required functionality.
- (b) **Interoperability**: Web administrations allow diverse apps to communicate with one another and exchange information and services. A .NET application, for example, can communicate with Java web administrations and vice versa. To make the application stage and innovation self-contained, web administrations are used.
- (c) **Communication with Low Cost**: Because web services employ the SOAP over HTTP protocol, you can use your existing low-cost internet connection to implement them.
- (d) **A Standard Protocol that Everyone Understands**: Web services communicate via a defined industry protocol. In the web services protocol stack, all four layers (Service Transport, XML Messaging, Service Description, and Service Discovery) use well-defined protocols.
- (e) **Reusability**: A single web service can be used simultaneously by several client applications.

Application Programming Interfaces (API)

- Software interface that helps in connecting between the computer or between computer programs. It is an interface that provides the accessibility of information such that weather forecasting. In simple words, you can say it is a software interface that offers service to the other pieces of software. Different(Android app, ios app, website, etc) devices call Api to get data.
- An application programming interface is a software that allows two applications to talk to each other, helps in enabling applications to exchange data and functionality easily, also called a middle man between two systems.
- E.g.- Login – In this functionality, APIs are widely used to log in via Google, Linked In, Git Hub, Twitter and allow users to access the log-in portal by using the API interface.
- Entertainment – In this field, APIs are used to access and provide a huge set of databases to access movies, web series, comedy, etc.

Fenil Shah

Types Of API

- **Open APIs** are publically available for anyone to use. BigCommerce, for example, uses roughly 25 different APIs, which is available for the public to use., weather apis, etc
- **Partner APIs** are designed by companies to offer API access to strategic business partners as an extra revenue channel for both parties. For example, Ticketmaster offers a Partner API to allow it's clients the ability to reserve, buy, and retrieve ticket/event information., flipkart shopsy, etc
- **Private APIs** are not designed for public use and are designed for internal use. Let's say you are traveling to a different city for a business meeting. You need to make a quick trip to the bank. You walk into "ABC Bank" and give the teller your account number. She quickly pulls up your account and you make a withdrawal. The teller was able to pull up your information by using ABC's internal system, which uses an API to pull your account information and to update your new account balance.

Fenil Shah

Benefits Of API

- **Security:**
- Security is enhanced when sites use APIs. Whenever you send a request, you aren't directly linked to a server. You send small amounts of information, the API delivers it, and the server sends it back. This minimizes the risk of a breach or someone accessing the backend of a server.
- **Speed:**
- Without APIs, you would have to call a store and ask them to look at their inventory from all their suppliers, which they would eventually get back to you. This, instead of having an API where you could easily see what a product was, the price, or it's stock level.
- **Scalability:**
- APIs allow scalability and flexibility when expanding your store's catalog, security, or data needs. Your store can grow at a faster rate when you don't have to factor in new code for every single product or user.

Fenil Shah

Webservice Vs Api

- An API (Application Programming Interface) is the means by which third parties can write code that interfaces with other code. A Web Service is a type of API, one that almost always operates over HTTP (though some, like SOAP, can use alternate transports, like SMTP). The official W3C definition mentions that Web Services don't necessarily use HTTP, but this is almost always the case and is usually assumed unless mentioned otherwise.
- For examples of web services specifically, see SOAP, REST, and XML-RPC. For an example of another type of API, one written in C for use on a local machine, see the Linux Kernel API.
- As far as the protocol goes, a Web service API almost always uses HTTP (hence the Web part), and definitely involves communication over a network. APIs in general can use any means of communication they wish. The Linux kernel API, for example, uses Interrupts to invoke the system calls that comprise its API for calls from user space.

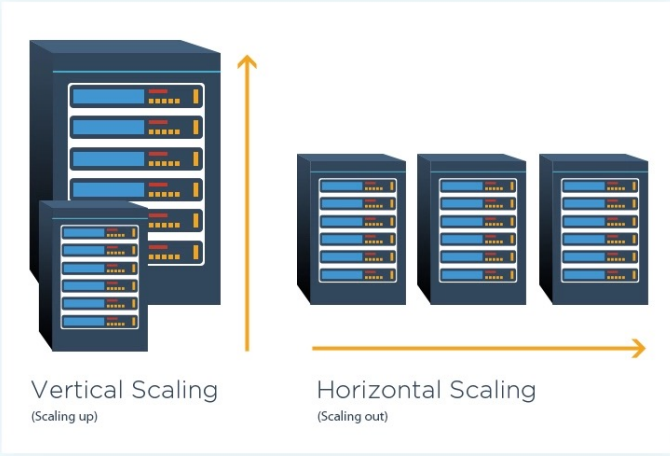
Fenil Shah

Scalability

- Scalability describes a system's elasticity. We can scale down, scale up, and scale out accordingly.
- If you are running a website, web service, or application, its success hinges on the amount of network traffic it receives. It is common to underestimate just how much traffic your system will incur, especially in the early stages. This could result in a crashed server and/or a decline in your service quality. Thus, scalability describes your system's ability to adapt to change and demand. Good scalability protects you from future downtime and ensures the quality of your service.
- **Horizontal Scaling**: Horizontal scaling (aka scaling out) refers to adding additional nodes or machines to your infrastructure to cope with new demands. If you are hosting an application on a server and find that it no longer has the capacity or capabilities to handle traffic, adding a server may be your solution.
- It is quite similar to delegating workload among several employees instead of one. However, the downside of this may be the added complexity of your operation. You must decide which machine does what and how your new machines work with your old machines.

Scalability

- **Vertical Scaling**: Vertical scaling (aka scaling up) describes adding additional resources to a system so that it meets demand. How is this different from horizontal scaling?
- While horizontal scaling refers to adding additional nodes, vertical scaling describes adding more power to your current machines. For instance, if your server requires more processing power, vertical scaling would mean upgrading the CPUs. You can also vertically scale the memory, storage, or network speed. Additionally, vertical scaling may also describe replacing a server entirely or moving a server's workload to an upgraded one.



Advantages of Horizontal scaling

- Scaling is easier from a hardware perspective - All horizontal scaling requires you to do is add additional machines to your current pool. It eliminates the need to analyze which system specifications you need to upgrade.
- Fewer periods of downtime - Because you’re adding a machine, you don’t have to switch the old machine off while scaling. If done effectively, there may never be a need for downtime and clients are less likely to be impacted.
- Increased resilience and fault tolerance - Relying on a single node for all your data and operations puts you at a high risk of losing it all when it fails. Distributing it among several nodes saves you from losing it all.
- Increased performance - If you are using horizontal scaling to manage your network traffic, it allows for more endpoints for connections, considering that the load will be delegated among multiple machines.
- Disadvantages : Increased complexity of maintenance and operation, Increased Initial costs

Fenil Shah

Advantages of Vertical scaling

- Cost-effective - Upgrading a pre-existing server costs less than purchasing a new one. Additionally, you are less likely to add new backup and virtualization software when scaling vertically. Maintenance costs may potentially remain the same too.
- Less complex process communication - When a single node handles all the layers of your services, it will not have to synchronize and communicate with other machines to work. This may result in faster responses.
- Less complicated maintenance - Not only is maintenance cheaper but it is less complex because of the number of nodes you will need to manage.
- Less need for software changes - You are less likely to change how the software on a server works or how it is implemented.
- Disadvantages : Higher possibility for downtime, Single point of failure, Upgrade limitations

When to Choose What?

- **Cost** - Initial hardware costs for horizontal upgrades are higher. If you are working on a tight budget and need to add more resources to your infrastructure quickly and cheaply, then vertical scaling may be the best option for you.
- **Future-proofing** - Adding additional updated machines through horizontal scaling will increase the overall performance threshold of your organization. There is a limit to how much you can vertically scale a single node and it may not be able to handle the demands of the future.
- **Topographic distribution** - If you plan to have nationwide or global clients, it is unreasonable to expect them all to access your services from a single machine in a single location. In a situation like this, you'll need to horizontally scale your resources to maintain your service level agreement (SLA).

When to Choose What?

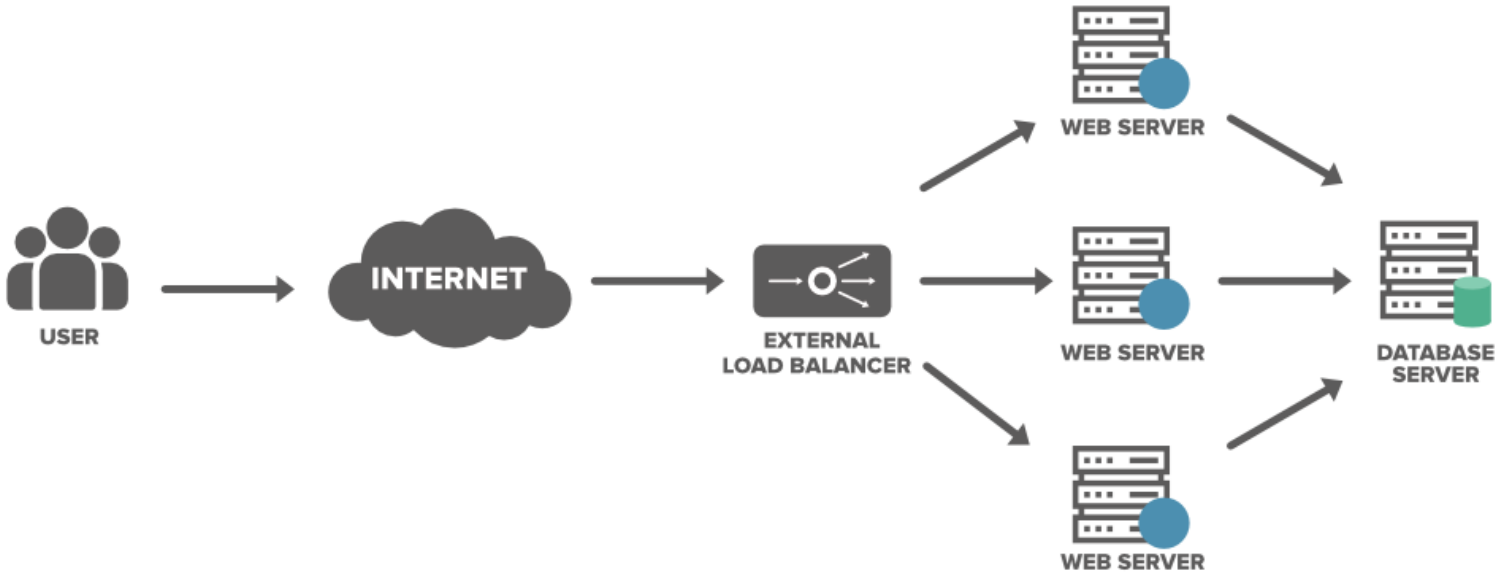
- **Reliability** - Horizontal scaling may offer you a more reliable system. It increases redundancy and ensures that you are not relying on a single machine. If one machine fails, another may be able to pick up the slack temporarily.
- **Upgradability and flexibility** - If you are running your application's tiers on individual machines, they are easier to decouple and upgrade without any downtime.
- **Performance and complexity** - Performance will depend on how your services work and how they are interconnected. Simple straightforward applications won't benefit much from being run on multiple machines. In fact, it may degrade its quality. Sometimes it's better to leave the application as is and upgrade the hardware to meet demand. Horizontal scaling may require you to rewrite the code or add a virtual machine that unifies all the servers.

Load Balancer

- When a website becomes extremely popular, the traffic on that website increases, and the load on a single server also increases. The concurrent traffic overwhelms the single server and the website becomes slower for the users. In order to meet the request of these high volumes of data and to return the correct response in a fast and reliable manner we need to scale the server. This can be done by adding more servers to the network and distributing all the requests across these servers. But….who is going to decide which request should be routed to which server…???
- A load balancer works as a “traffic cop” sitting in front of your server and routing client requests across all servers. It simply distributes the set of requested operations (database write requests, cache queries) effectively across multiple servers and ensures that no single server bears too many requests that lead to degrading the overall performance of the application. A load balancer can be a physical device or a virtualized instance running on specialized hardware or a software process.
- Problems: Single Point of Failure, Overloaded Servers

Load Balancer

- To solve the above issue and to distribute the number of requests we can add a load balancer in front of the web servers and allow our services to handle any number of requests by adding any number of web servers in the network. We can spread the request across multiple servers. For some reason, if one of the servers goes offline the service will be continued. Also, the latency on each request will go down because each server is not bottle-necked on RAM/Disk/CPU anymore.



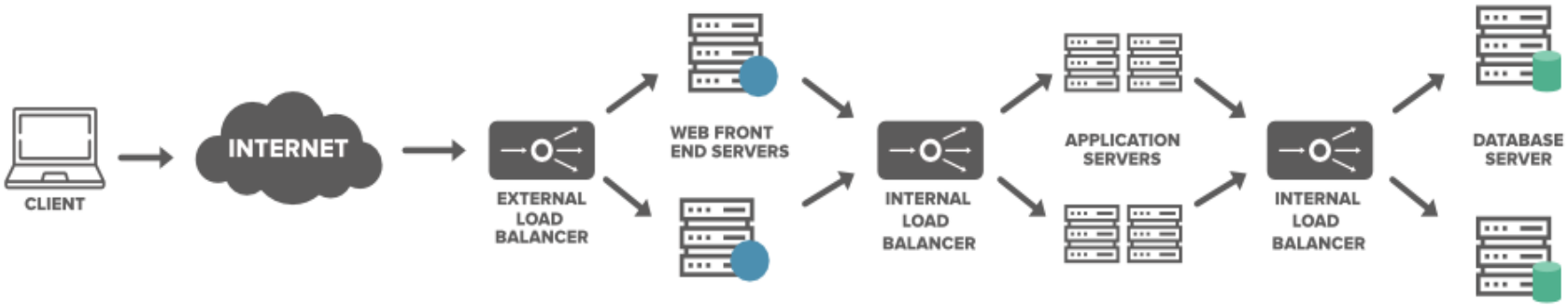
Load Balancer

- Load balancers minimize server response time and maximize throughput.
- Load balancer ensures high availability and reliability by sending requests only to online servers
- Load balancers do continuous health checks to monitor the server’s capability of handling the request.
- Depending on the number of requests or demand load balancer add or remove the number of servers.
- A load balancer enables elastic scalability which improves performance and throughput of data. It allows you to keep many copies of data (redundancy) to ensure the availability of the system. In case if a server goes down or fails you’ll have the backup to restore the services.
- Load balancers can be placed at any software layer.
- Many companies use both hardware and software to implement the load balancers, depending on the different scale points in their system.

Fenil Shah

Load Balancer

- load balancer can be placed:
- In between the client application/user and the server
- In between the server and the application/job servers
- In between the application servers and the cache servers
- In between the cache servers the database servers



Types of Load Balancer

- 1**. Software Load Balancers in Clients**
- As the name suggests all the logic of load balancing resides on the client application (Eg. A mobile phone app). The client application will be provided with the list of web servers/application servers to interact with. The application chooses the first one in the list and requests data from the server. If any failure occurs persistently (after a configurable number of retries) and the server becomes unavailable, it discards that server and chooses the other one from the list to continue the process. This is one of the cheapest ways to implement load balancing.
- 2. **Software Load Balancers in Services**
- These load balancers are the pieces of software that receive a set of requests and redirect these requests according to a set of rules. This load balancer provides much more flexibility because it can be installed on any standard device (Ex: Windows or Linux machine). It is also less expensive because there is no need to purchase or maintain the physical device, unlike hardware load balancers. You can have the option to use the off-the-shelf software load balancer or you can write your custom software (Ex: load balance Active Directory Queries of Microsoft Office365) for load balancing.

Fenil Shah

Types of Load Balancer

- 3. **Hardware Load Balancers**
- As the name suggests we use a physical appliance to distribute the traffic across the cluster of network servers. These load balancers are also known as Layer 4-7 Routers and these are capable of handling all kinds of HTTP, HTTPS, TCP, and UDP traffic. HLDs provides a virtual server address to the outside world. When a request comes from a client application, it forwards the connection to the most appropriate real server doing bi-directional network address translation (NAT). HLDs can handle a large volume of traffic but it comes with a hefty price tag and it also has limited flexibility.
- HLDs keep doing the health checks on each server and ensure that each server is responding properly. If any of the servers don’t produce the desired response, it immediately stops sending the traffic to the servers. These load balancers are expensive to acquire and configure, that is the reason a lot of service providers uses it only as the first entry point of user requests. Later the internal software load balancers are used to redirect the data behind the infrastructure wall.

Load Balancing Algorithms

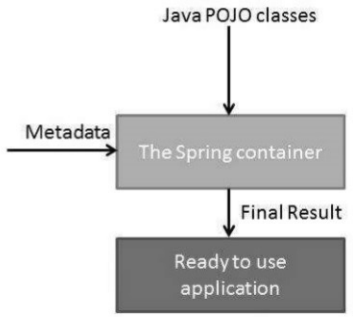
- We need a load balancing algorithm to decide which request should be redirected to which backend server.
- **Round Robin**: Requests are distributed across the servers in a sequential or rotational manner. For example, the first request goes to the first server, the second one goes to the second server, the third request goes to the third server and it continues further for all the requests. It is easy to implement but it doesn't consider the load already on a server so there is a risk that one of the servers receives a lot of requests and becomes overloaded.
- **Least Connection Method**: In this method, the request will be directed to the server with the fewest number of requests or active connections. To do this load balancer needs to do some additional computing to identify the server with the least number of connections. This may be a little bit costlier compared to the round-robin method but the evaluation is based on the current load on the server. This algorithm is most useful when there is a huge number of persistent connections in the traffic unevenly distributed between the servers.
- **Least Response Time Method**: This technique is more sophisticated than the Least connection method. In this method, the request is forwarded to the server with the fewest active connections and the least average response time. The response time taken by the server represents the load on the server and the overall expected user experience.

Spring Framework

- EJB extends the Java components, such as Web and enterprise components, and provides services that help in enterprise application development. However, developing an enterprise application with EJB was not easy, as the developer needed to perform various tasks, such as creating Home and Remote interfaces and implementing lifecycle callback methods which lead to the complexity of providing code for EJBs.
- Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code. It is an open source Java platform. Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around 2MB.
- Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
- Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
- Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
- Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

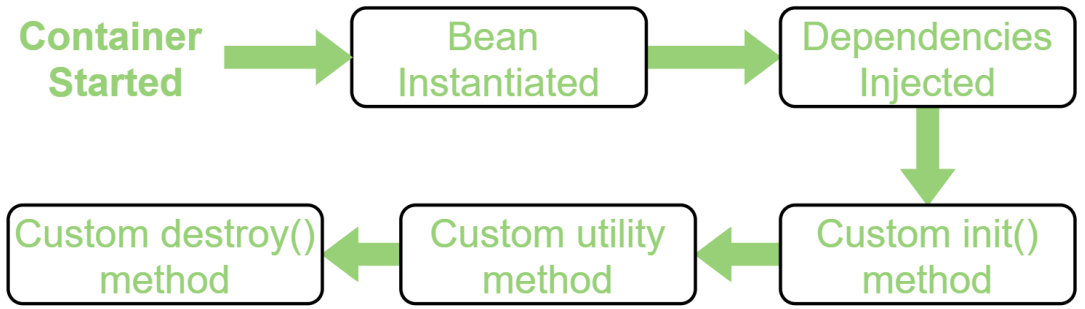
Spring Framework

- Spring framework can be considered as a collection of sub-frameworks, also called layers, such as Spring AOP. Spring Object-Relational Mapping (Spring ORM). Spring Web Flow, and Spring Web MVC. It is a lightweight application framework used for developing enterprise applications. You can use any of these modules separately while constructing a Web application. The modules may also be grouped together to provide better functionalities in a Web application. Spring framework is loosely coupled because of dependency Injection.
- **Spring container** is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans
- The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



Bean

- **Singleton**: This scopes the bean definition to a single instance per Spring IoC container (default).
- **Prototype**: Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made. prototype scope for all state-full beans and the singleton scope for stateless beans.
- **Request**: scopes a bean definition to an HTTP request.
- **Session**: scopes a bean definition to an HTTP session.
- if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.



Spring Framework

- Spring ApplicationContext is where Spring holds instances of objects that it has identified to be managed and distributed automatically.
- **Inversion of Control**: Instead of we managing the lifecycle of objects, Spring does on behalf of us. Eg- drive car or free cab.
- **IoC container**: Refers to the core container that uses the DI or IoC pattern to implicitly provide an object reference in a class during runtime. This pattern acts as an alternative to the service locator pattern. The IoC container contains assembler code that handles the configuration management of application objects. The Spring framework provides two packages, namely org.springframework.beans and org.springframework.context which helps in providing the functionality of the IoC container. Spring will create object for us and we will inject it wherever required. Eg- Injecting service class in controller.
- **Dependency Injection** is merely one concrete example of Inversion of Control. When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection helps in gluing these classes together and at the same time keeping them independent
- **Disadvantage** of spring projects is that configuration is really time-consuming and can be a bit overwhelming for the new developers. Making the application production-ready takes some time if you are new to the spring.

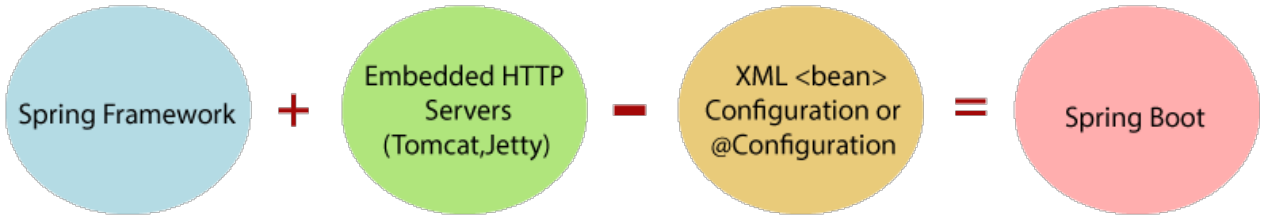
Spring vs Springboot

Spring

- used Java EE framework for building applications.
- **dependency injection**.
- develop **loosely coupled** applications.
- developer writes a lot of code (**boilerplate code**) to do the minimal task.
- To test the Spring project, we need to set up the sever explicitly.
- It does not provide support for an in-memory database.
- Developers manually define dependencies for the Spring project in **pom.xml**.

Springboot

- used to develop **REST APIs**.
- **Autoconfiguration**
- create a **stand-alone** application with less configuration.
- **reduces** boilerplate code.
- Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc.
- It offers several plugins for working with an embedded and **in-memory** database such as **H2**
- Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement.



Spring Boot

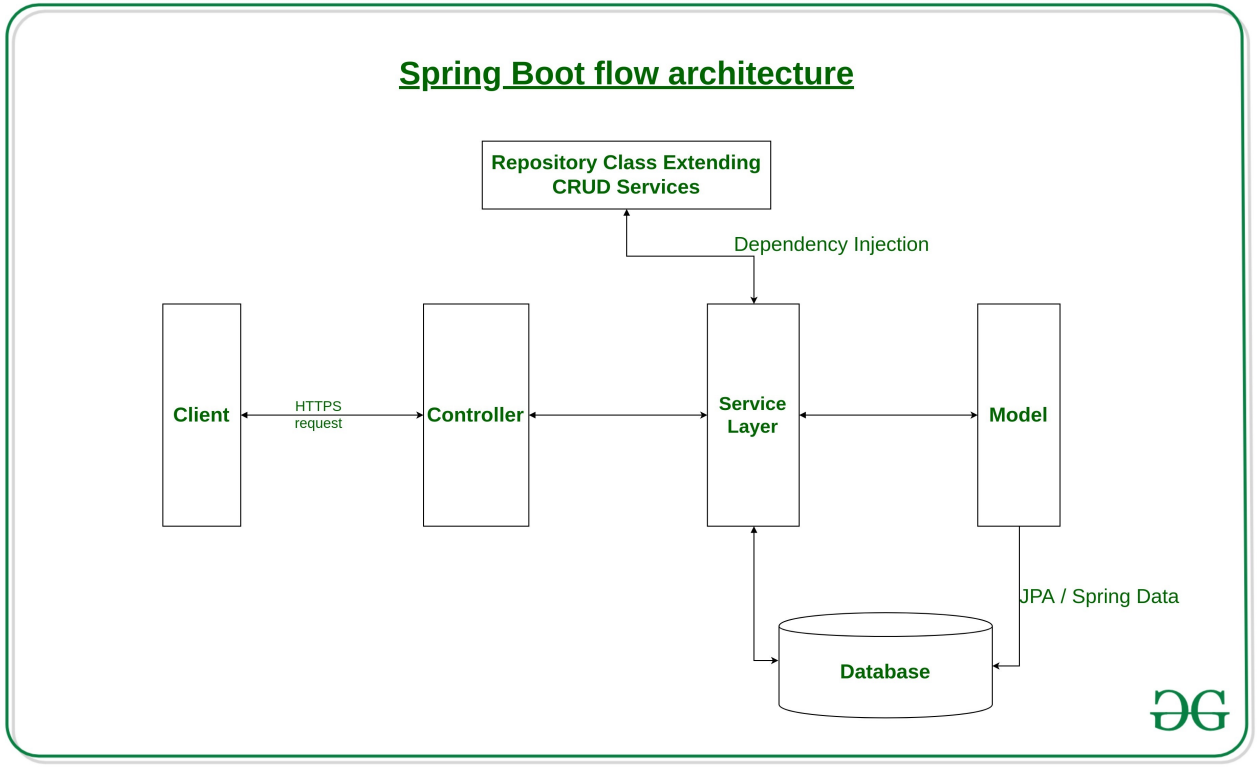
- Spring Boot is built on the top of the spring and contains all the features of spring. And is becoming favourite of developer's these days because of it's a rapid production-ready environment which enables the developers to directly focus on the logic instead of struggling with the configuration and set up.
- Spring Boot is an opinionated, easy to get-started addition to the Spring platform – highly useful for creating stand-alone, production-grade applications with minimum effort.
- **It allows to avoid heavy configuration of XML which is present in spring**: Unlike the Spring MVC Project, in spring boot everything is auto-configured. We just need to use proper configuration for utilizing a particular functionality. For example: If we want to use hibernate(ORM) then we can just add @Table annotation above model/entity class(discussed later) and add @Column annotation to map it to table and columns in the database
- **It provides easy maintenance and creation of REST end points**: Creating a REST API is very easy in Spring Boot. Just the annotation @RestController and @RequestMapping(/endPoint) over the controller class does the work.
- **It includes embedded Tomcat-server**: Unlike Spring MVC project where we have to manually add and install the tomcat server, Spring Boot comes with an embedded Tomcat server, so that the applications can be hosted on it.

Spring Boot

- **Deployment is very easy, war and jar file can be easily deployed in the tomcat server**: war or jar files can be directly deployed on the Tomcat Server and Spring Boot provides the facility to convert our project into war or jar files. Also, the instance of Tomcat can be run on the cloud as well.
- creates stand-alone Spring applications that can be started using Java –jar, minimizes writing multiple boilerplate codes, increases productivity and reduces development time. better to use if we want to develop a simple Spring-based application or RESTful services.
- **Layers in Spring Boot**: There are four main layers in Spring Boot:
- **Presentation Layer**: As the name suggests, it consists of views(i.e. frontend part)
- **Data Access Layer**: CRUD (create, retrieve, update, delete) operations on the database comes under this category.
- **Service Layer**: This consist of service classes and uses services provided by data access layers.
- **Integration Layer**: It consists of web different web services(any service available over the internet and uses XML messaging system).
- Then we have utility classes, validator classes and view classes. All the services provided by the classes are implemented in their corresponding classes and are retrieved by implementing the dependency on those interfaces.

Spring Boot Flow

- The client makes the HTTP requests (PUT or GET). The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
- In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes. A JSP page is returned to the user if no error occurred.



Fenil Shah

Spring Boot Annotations

- **@SpringBootApplication**: Marks the Main class of Application. encapsulates @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.
- **@EnableAutoConfiguration**: It auto-configures the bean that is present in the classpath and configures it to run the methods. Enables Spring Boot to auto-configure the application context. Therefore, it automatically creates and registers beans based on both the included jar files in the classpath and the beans defined by us. Applied in the root package so that every sub-packages and class can be examined. E.g. - when we define the spring-boot-starter-web dependency in our classpath, Spring boot auto-configures Tomcat and Spring MVC.
- **@ComponentScan**: Used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components. Enables Spring to scan for things like configurations, controllers, services, and other components we define. If no package is specified, then it considers the package of the class declaring the @ComponentScan annotation as the starting package.
- **@Configuration**: indicates that the class has @Bean definition methods. So Spring container can process the class and generate Spring Beans to be used in the application. It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions. @Configuration annotation with configuration classes to make sure our spring container is behaving like the way we want it

Fenil Shah

Spring Boot Annotations

- **@component:** Allows Spring to automatically detect our custom beans. Scan our application for classes annotated with @Component. Instantiate them and inject any specified dependencies into them. Inject them wherever needed.
- **@RestController**: a convenience annotation for creating Restful controllers. Used at the class level and allows the class to handle the requests made by the client. specialization of @Component. It adds the @Controller and @ResponseBody annotations. It converts the response to JSON or XML.
- **@Service**: In an application, the business logic resides within the service layer so we use this Annotation to indicate that a class belongs to that layer. It is also a specialization of @Component Annotation. Used to mark the class as a service provider. Used with classes that provide some business functionalities. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used. Improves User readability.
- **@Repository**: a specialization of @Component annotation which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects. This annotation is a general-purpose stereotype annotation which very close to the DAO pattern where DAO classes are responsible for providing CRUD operations on database tables. helps us to convert the non-spring-based expectation to a spring unchecked exception. This helps us to identify where does the problem occur. we implement this repository as the interface in Java and extend different repositories available most commonly we used JPARepository to perform the crud operations on the object.

Fenil Shah

Embedded Server

- In a typical microservice architecture, there could be hundreds of microservice instances deployed at a given point in time. We would like to automate development and deployment of microservices to the maximum extent possible. An interesting approach would be to make the server a part of your application.
- It gives you low level code that helps to start the server . An embedded server is embedded as part of the deployable application. If we talk about Java applications, that would be a JAR. The advantage with this is you don’t need the server pre-installed in the deployment environment. With SpringBoot, the default embedded server is Tomcat (built by Apache). Other options available are Jetty(built by Eclipse) and UnderTow(built by Redhat) .
- These can be used with applications for deployment in high-workload environments, without sacrificing any reliability or stability.
- Embedded servers are also quite lightweight. If you look at a conventional WebSphere or Weblogic installation, or even a default Tomcat setup, their install sizes are huge!
- Embedded server images don’t generally result in huge archive sizes and helps in building smaller containers.
- An embedded Tomcat server consists of a single Java web application along with a full Tomcat server distribution, packaged together and compressed into a single JAR, WAR or ZIP file.

Jar and War Packaging

- **JAR** – or Java Archive – is a package file format. JAR files have the .jar extension and may contain libraries, resources, and metadata files. it's a zipped file containing the compressed versions of .class files and resources of compiled Java libraries and applications.
- The META-INF/MANIFEST.MF file may contain additional metadata about the files stored in the archive. We can create a JAR file using the jar command or with tools like Maven.
- JAR files allow us to package multiple files in order to use it as a library, plugin, or any kind of application. we can run a JAR from the command line if we build it as an executable JAR without using additional software
- **WAR** stands for Web Application Archive or Web Application Resource. These archive files have the .war extension and are used to package web applications that we can deploy on any Servlet/JSP container.
- The META-INF directory is private and can't be accessed from the outside. it also contains the WEB-INF public directory with all the static web resources, including HTML pages, images, and JS files. Moreover, it contains the web.xml file, servlet classes, and libraries.
- WAR files are used only for web applications. we need a server to execute a WAR.

Fenil Shah

LOGGING LEVELS IN SPRING BOOT

- A good logging infrastructure is necessary for any software project as it not only helps in understanding what's going on with the application but also to trace any unusual incident or error present in the project.
- Severity: Error > Warning > Info > Debug > Trace. Default logging level is INFO . We can change it by defining property in the application.properties file : logging.level.<package_name> = debug Example : logging.level.root = debug where root stands for applying the logging level to all packages in the application
- Spring boot allows us to see the logs in the console even if we do not provide any specific configuration for it. This is because spring boot uses Logback for its default logging. Spring boot's internal logging provider is Apache Commons which provides support for Java Util Logging ,Log4j2, and Logback.
- It is possible to activate the debug and trace level by starting the application with –debug flag or –trace flag. E.g. - java -jar target/log-0.0.1-SNAPSHOT.jar –debug
- Logging to file: logging.file.path=logs/ & logging.file.name=logs/application.log
- logback-spring.xml file: we can specify logging pattern, color, different properties for file logging & console logging, and an efficient rolling policy to prevent the creation of huge log files. Whenever Spring boot finds a file with any of the following names, It automatically overrides the default configuration.
- E.g. - 2022-07-10 10:49:59.242 INFO 27560 --- [restartedMain] com.example.demo.DemoApplication : Started DemoApplication in 0.79 seconds (JVM running for 1.276)

Fenil Shah

Spring Profiles

- Profiles help to have different application configuration for different environments. Eg- db configuration, logging mechanism, server port, etc.
- Profiles are a core feature of the framework — allowing us to map our beans to different profiles. We can then activate different profiles in different environments to bootstrap only the beans we need. E.g. – dev, preprod, prod, qa, etc
- a profile may influence the application properties, and may influence which beans are loaded into the application context. Adding certain beans to the application context for one profile, but not for another, can quickly add complexity to our application! We always have to pause and think if a bean is available in a particular profile or not, otherwise, this may cause NoSuchBeanDefinitionExceptions when other beans depend on it. Most use cases can and should be implemented using profile-specific properties instead of profile-specific beans.
- Activating Profiles: In Environment tab of Run Configurations or java -Dspring.profiles.active=dev -jar profiles-0.0.1-SNAPSHOT.jar