Project Report on

Survey on Authentication and Authorization on a Login System and Kubernetes

A project report submitted in partial fulfilment of the requirements for the award of the degree

of

Bachelor of Technology

in

Computer Science and Engineering

by

MOHIT PRASAD (17101104025)

SYAMANTAK SARKAR (17101104039)

ROUNAK DE (17101104017)

MD. KARIM SK (17101104028)

OMAR FARUK GAZI (17101104033)

Under the supervision

of

Prof. Dipak Kr. Kole

Professor

Department of Computer science and Engineering

JALPAIGURI GOVERNMENT ENGINEERING COLLEGE

May 2021

JALPAIGURI GOVERNMENT ENGINEERING COLLEGE

(A Government Autonomous College)

Jalpaiguri, West Bengal

Certification

This is to certify that the work in preparing the project entitled "*Survey on Authentication and Authorization using Kubernetes* ", has been carried out by **Mohit Prasad, Syamantak Sarkar, Rounak De, Md. Karim Sk, Omar Faruk Gazi** under my guidance during the session 2020-2021 and accepted in partial fulfilment of requirement for the degree of Bachelor of Technology in Computer Science & Engineering.

Signature of the HoD with seal

Dr. Subhas Barman

Head of Department

Computer Science and Engineering

Jalpaiguri Govt. Engineering College

Signature of the Supervisor with seal

Prof. Dipak Kr. Kole

Professor

Computer Science and Engineering

Jalpaiguri Govt. Engineering College

Acknowledgement

The project work summarized in this report, "**Survey on Authentication and Authorization using Kubernetes".**

In this project, we explore the scopes of Kubernetes. The Login System authenticates a user while logging in and then checks for the authority of the user in the section he/she is trying to fetch the data from.
We hereby thank our project supervisor Dr. Dipak Kr. Kole for guiding us in all the ways possible. This is a combined endeavor of a number of people who directly or indirectly helped us in completing our project survey. A word of thanks also goes to all our friends for being our best critics. And finally, this documentation would never have been more educative and efficient without the constant help and guidance of Mr. Baibhav Ojha. We would like to thank him for giving us the right guidance and encouraging us to complete the project efficiently. We would also like to express our heartiest gratitude to HOD, Dr. Subhas Barman and other faculties for their encouragement and kind suggestion.

Mohit Prasad

Mohit Prasad

Md. Karim Sk

Md Karim SK

Syamantak Sarkar

Syamantak Sarkar

Omar Faruk Gazi

Omar Faruk Gazi.

Rounak De

Rounak De

Contents:

Abstract

Kubernetes (also known as k8s) is a container orchestration tool widely used to deploy and update the application without any downtime. It enables auto-scaling of resources in case of an increase/decrease in requests to the server. The project deals with authentication and authorization of a user when the user tries to login. The user is first authenticated to make sure the user is a part of the system, then it is checked whether the user is authorized to access the section of the website that he/she is trying to access. In Kubernetes RBAC (Role Based Access Control) is used for authorization.

The Login system checks for the user in database (mongo DB) and node JS handles the Server Side. The User Interface is designed using HTML, CSS and JS.

The login system detects malicious login and sends the user a notification in case of any expected privacy breach after the user types the password incorrectly for 3 consecutive times. The system also notifies the admin about the IP through with the malicious login was attempted so that incase of any future malicious attempt the admin can take necessary actions.

Chapter 1:

Introduction:

Our Motive is to survey on how a login system works. We have studied on the working of Authentication and Authorization and how it can be useful in a web application.

We studied about using OAuth for authentication.

We have surveyed on building up of a web application that contains multiple microservices and different tools needed to User Interface. We have also surveyed how the microservices are controlled on the server side, including fetching of responses and API calls.

We have surveyed on the functioning of Kubernetes, and its working with Docker.

We surveyed on the advantages of Docker over Virtual Machines.

Chapter 2: Authentication and Authorization

We the users need to authenticate ourself every time we initiate interaction with a computer system and are then given permissions dependent on our authorization data, either automatically or by providing some extra authorization credentials depending on the operational security policies governing the protected re-sources of the organizations and enterprises we belong to.

Basically, an authentication process is a way by which you provide proofs to a system, that you are in fact, who you claimed to be, by presenting a valid credential. At its simplest form this credential is a pair of user identification and password. A user identification ($\mathcal{U}$) is a unique string of characters or digits directly related to a user entity whether it is a human, a thing or a program. A password ($\mathcal{P}$) is a secret phrase known only to the user and not directly known to the system, but directly related to the user identification. Thus, to be authenticated ($\mathcal{I}$), a user needs to provide a valid pair of [$\mathcal{U}, \mathcal{P}$].

Authentication is not authorization, but to be authorized you need to be authenticated first, since per-missions mandate three attributes' conditions of 1) who, 2) what, and 3) which, within a user's work session in relation to ($\mathcal{I}$) with a system. A permission (Q) can be described by a triplet [$\mathcal{I}$, Φ, $\mathcal{R}$] where $\mathcal{I}$ is the validated identity, Φ is a non-empty set of allowable operations, and $\mathcal{R}$ is a non-empty set of resources.

Permissions are real time attributes for control-ling users' access and operations to protected re-sources. Authorization data is much more involved than the authentication data. To provide real time user's permissions, authorization processes need to be assisted by groups and groups memberships coupled with resource owners and resource groups. Management of authorization data is necessarily delegated to different managers closely related to organizational structure and resource ownership.

The Basic difference between Authentication and Authorization is shown in Fig 2.i.

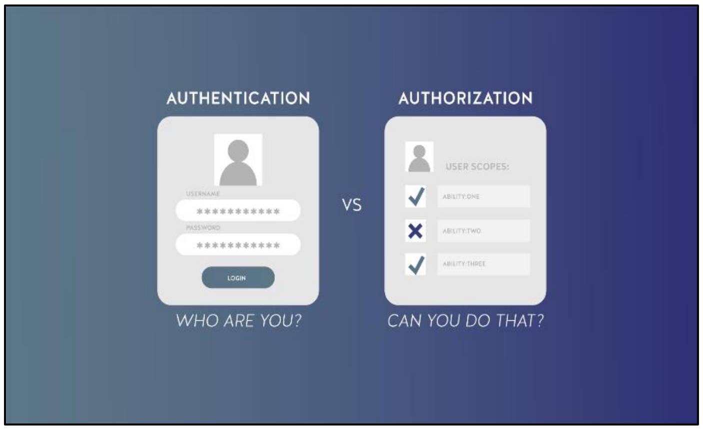

Fig 2.i

2.1 Authentication:

- Authentication is used by a server when the server needs to know exactly who is accessing their information or site.
- Authentication is used by a client when the client needs to know that the server is system it claims to be.
- In authentication, the user or computer has to prove its identity to the server or client.
- Usually, authentication by a server entails the use of a user name and password. Other ways to authenticate can be through cards, retina scans, voice recognition, and fingerprints.
- Authentication by a client usually involves the server giving a certificate to the client in which a trusted third party such as Verisign or Thawte states that the server belongs to the entity (such as a bank) that the client expects it to.
- Authentication does not determine what tasks the individual can do or what files the individual can see. Authentication merely identifies and verifies who the person or system is.

2.2 Authorization:

- Authorization is a process by which a server determines if the client has permission to use a resource or access a file.
- Authorization is usually coupled with authentication so that the server has some concept of who the client is that is requesting access.
- The type of authentication required for authorization may vary; passwords may be required in some cases but not in others.
- In some cases, there is no authorization, any user may be using a resource or access a file simply by asking for it. Most of the web pages on the Internet require no authentication or authorization.

One of the first things to give thought to when creating an auth strategy is what type of token you will use. There are a variety of methods, but two of the most common are:

1. JWT Tokens (JSON Web Tokens

JWT Tokens are actually a full JSON Object that has been base64 encoded and then signed with either a symmetric shared key or using a public/private key pair. The difference is if you have a consumer that needs to verify the token is signed, but that consumer shouldn't be allowed to create tokens, you can give the consumer the public key which can't create tokens but still verify them. I'll save the details for a separate post, but if you remember from your cryptography courses, asymmetric encryption and signature algorithms create a pair of keys that are mathematically related.

2. Opaque Token

Now that we discussed the benefits and drawbacks of JWTs, let's explore a secondary option, Opaque Tokens. Opaque tokens are literally what they sound like. Instead of storing user identity and claims in the token, the opaque token is simply a primary key that references a database entry which has the data. Fast key value stores like Redis are perfect for leveraging in memory hash tables for O(1) lookup of the payload. Since the roles are read from a database directly, roles can be changed and the user will see the new roles as soon as the changes propagate through your backend.

2.3 Cookies vs. Headers vs. URL Parameter

URL Parameters

First of all, we never recommend placing a token in the URL. URLs are bookmarked, shared with friends, posted online, etc. It is too easy for a user to copy a URL of a page they like and forward to a friend. Now, the friend has all the authentication requirements needed to sign in on behalf of that user. In addition, there are a variety of other concerns such as most loggers will log the URL in plain text at a minimum. It's much less likely for someone to open developer tools in Chrome and copy their cookie or HTTP headers.

So, this narrows us down to only Cookies vs HTTP Headers. If you are focused on creating RESTful APIs, really the Cookie is just another header just like the Authorization Header. You could in fact take the same JWT that would be sent via an Authorization Header and wrap it in a cookie. In reality, many pure RESTful APIs designed for consumption by others just use a standard or custom authorization header as it is more explicit. There can also be a blend, for example a web app may talk to a RESTful API behind a proxy using Cookies. The proxy will extract the Cookie and add the appropriate headers when relaying the request. The real reasons come in the next section:

The real debate: Cookie vs. Local Storage:

While the cookie may in fact just wrap the JWT or opaque token, a client web app still has to store the token somewhere. The two choices that most people think of is Cookies and HTML5 Local Storage. So sometimes when people refer to Cookie vs HTTP Header, they are actually asking "Cookie vs. Local Storage?" There are benefits and security risks for each.

Cookies

Cookies can be nice as they have certain flags that can be set to enforce security checks such as HTTP Only and Secure. By setting HTTP Only and Secure flags, the cookie cannot be read by any JavaScript code nor be sent in plain text over HTTP. Thus, the Cookie can be immune to XSS attacks as described in the local storage Section. Cookies can be vulnerable to a different type of attack called cross site request forgery (XSRF or CSRF). XSRF means a hacker on a different site can replicate some input form on your own site and POST form data to our own site. While the hacker doesn't have access to the cookie itself, cookies are transferred with every HTTP request to your real domain that the cookie is valid for. Thus, the hacker doesn't need to read the cookie, it just needs to successfully POST form data to your real site. This is one of the dangers with cookies. They are sent for every request, static, AJAX, etc. There are ways around this, but the fundamental principal is that your web server needs to recognize whether the request came from your real website running in a browser or someone else. One way to do this is with a hidden anti-forgery token. One way is to generate and store a special random key in the cookie that also needs to be sent with the POSTed form data. Remember, only your real site can *access* the cookie but the hacker site cannot due to same origin policy. Your server can then verify that the cookie's token matches the token in form data. There are other options for protection on XSRF.

Local Storage

A security risk for local storage is JavaScript can be subject to Cross-Scripting attacks (XSS). In the early days, XSS was a result of not escaping user input, but now your modern web app probably imports numerous JS libs from analytics and attribution tracking to ads and small UI elements. Local storage is global to your website domain. Thus, any JavaScript on your website, 3rd party lib or not, can access the same local storage. There is no sandboxing within your app. For example, your analytics lib reads and writes from the same local storage as your own application code. While GA is probably fine, did you audit that quick UI element you added? In the past, there was also concern if JavaScript made AJAX calls in plain text even though the website itself was secured via HTTPS. This concern is less than it used to know that browsers are starting to enforce checks for mixed content. Something to still be aware of incase a browser is older or launched without enforcement.

A second downside for local storage is you can't access it across multiple subdomains. If you have a separate blog subdomain or email subdomain, these sites cannot read the local storage. This might be fine if you don't plan on logging in across multiple domains (Think mail.google.com and google.com).

2.4 OAuth

OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords. This mechanism is used by companies such as Amazon, Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites.

Generally, OAuth provides clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without providing credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The third party then uses the access token to access the protected resources hosted by the resource server.

Using OAuth for authentication

Using external OAuth for Authorizing a user login is helpful where the owner doesn’t need to take the headache of authorizing the user by the organization itself, but there might be a case, though highly unlikely in this era, where the user doesn’t have an account in the OAuth service providers

Some popular OAuth service providers are

Google

Facebook

LinkedIn

GitHub

Microsoft

Twitter

Chapter 3: User Interface

3.1 HTML:

The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets and scripting languages such as JavaScript. It defines the basic structure or layout of a webpage.

A Webpage with only HTML (Fig 3.i)

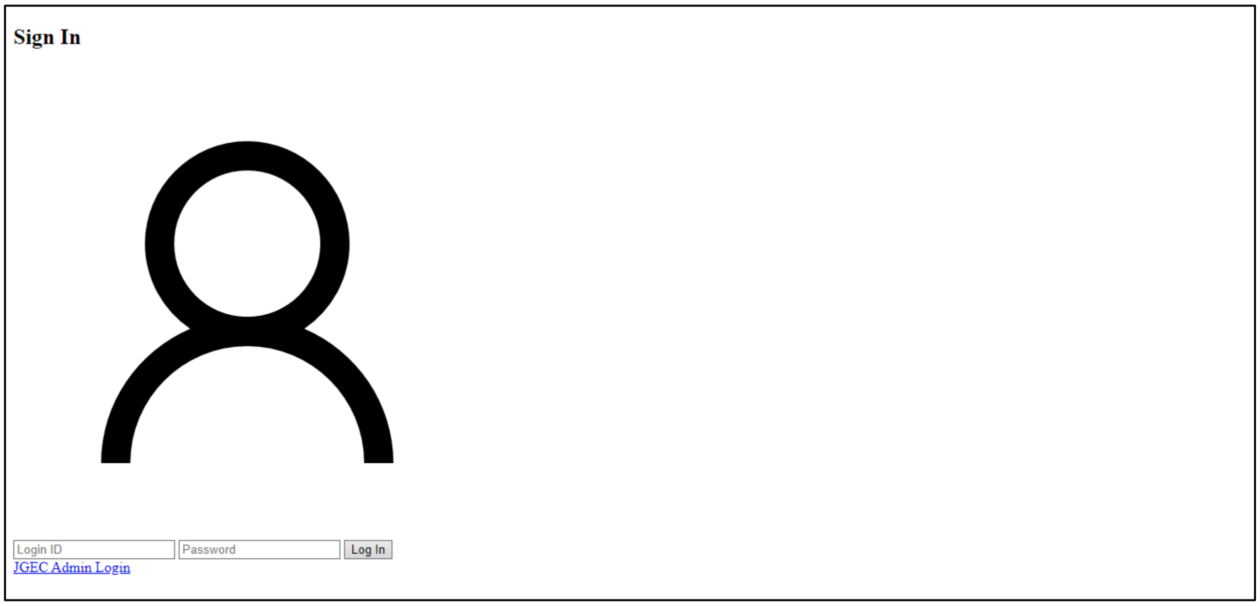

Fig 3.i

3.2 CSS:

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

Webpage with CSS (Fig 3.ii)

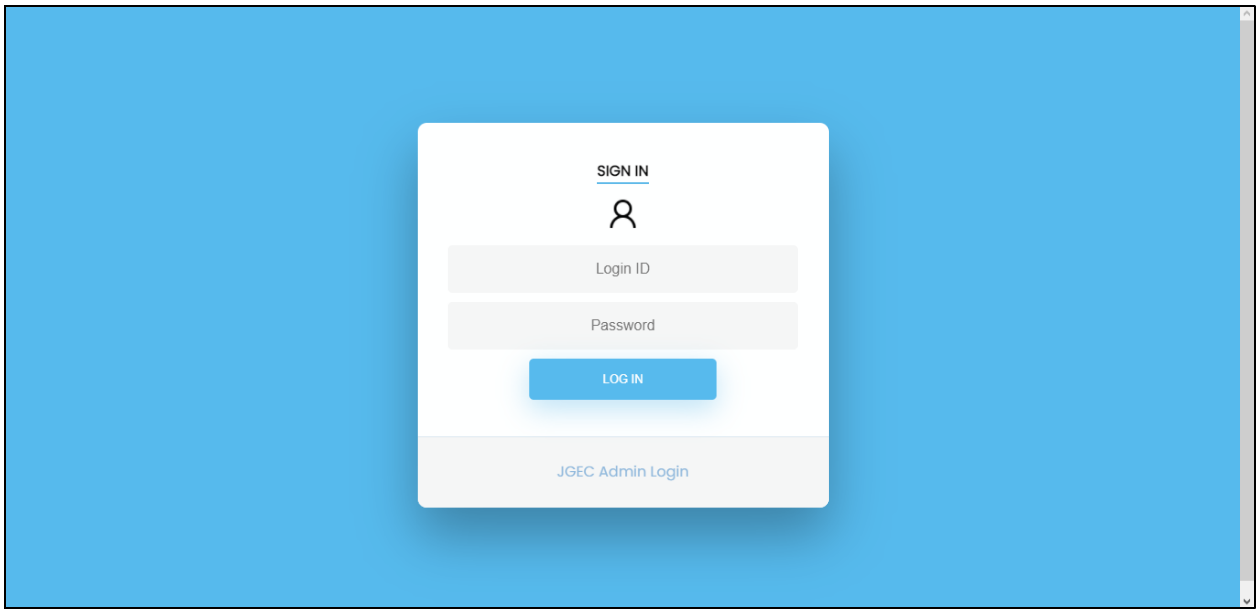

Fig 3.ii

3.3 JavaScript:

JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Where HTML and CSS are languages that give structure and style to web pages, JavaScript gives web pages interactive elements that engage a user. Incorporating JavaScript improves the user experience of the web page by converting it from a static page into an interactive one. Developers can use various JavaScript frameworks for developing and building web and mobile apps. JavaScript frameworks are collections of JavaScript code libraries that provide developers with pre-written code to use for routine programming features and tasks—literally a framework to build websites or web applications around.

Popular JavaScript front-end frameworks include React, React Native, Angular, and Vue. Many companies use Node.js, a JavaScript runtime environment built on Google Chrome’s JavaScript V8 engine.

Webpage with JavaScript (Fig 3.iii)

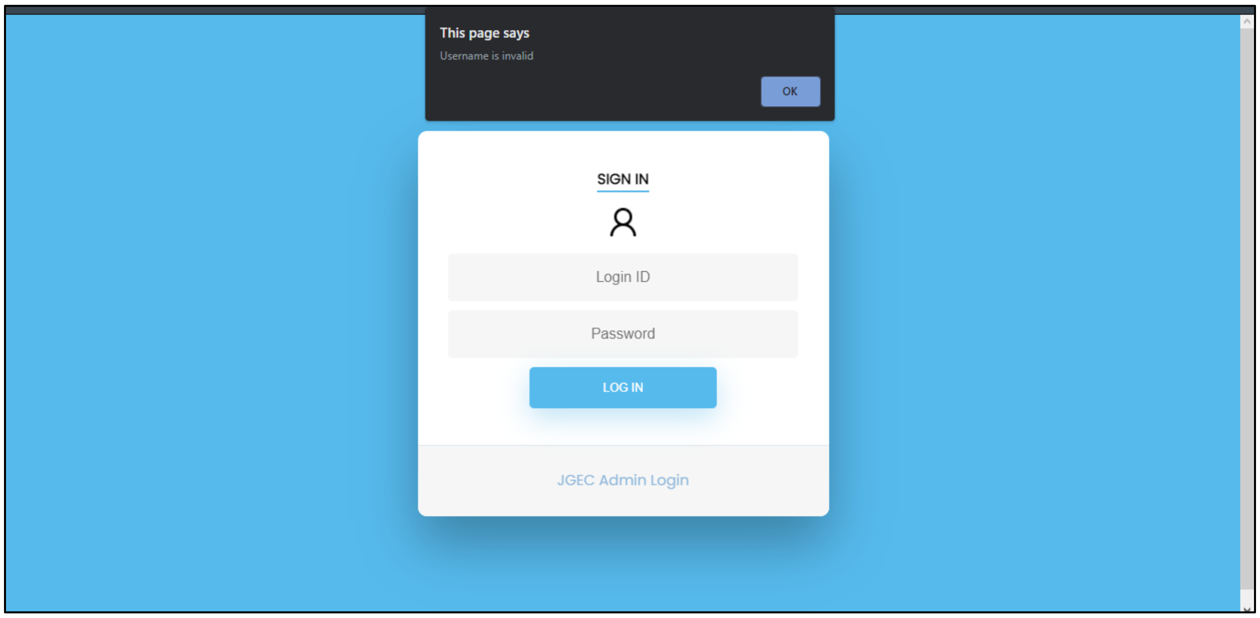
This page says
Username is invalid

Fig 3.iii

Chapter 4: Server Side

4.1 Node.js:

Introduction

● Node.js is a very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine.Node.js uses asynchronous programming. It is used to develop web and mobile applications.

● Nodes 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.

● Node is quite popular and used by some big companies like eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, and Yahoo! just to name a few.

How It Works

Compared to traditional web-serving techniques where each connection (request) spawns a new thread, taking up system RAM and eventually maxing-out at the amount of RAM available, Node.js operates on a single-thread, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections held in the event loop.

A quick calculation: assuming that each thread potentially has an accompanying 2 MB of memory with it, running on a system with 8 GB of RAM puts us at a theoretical maximum of 4,000 concurrent connections (calculations taken from Michael Abernethy's article "Just what is Node.js?", published on IBM developer Works in 2011), plus the cost of context-switching between threads. That's the scenario you typically deal with in traditional web-serving techniques. By avoiding all that, Node.js achieves scalability levels of over 1M concurrent connections, and over 600k concurrent web sockets connections.

Advantages of Node.js over Traditional

- Node.js is relatively new. So, it provides a wide range of new plugins that are designed according to the modern architectural approach.
- Node.js provides structure in programming which makes it easy to understand and maintain. This facilitates programmers to structure and make their task more organized as shown in Fig 4.i.
- Node.js is event driven and non-blocking so, speed (performance) is very high and this is the main advantage of using it.
- Node.js can handle concurrent requests more than other web technologies due to its event-driven nature.
- If you do something where you have many connections open at the same time then you must go with Node.js because it does not require a lot of memory and enhances speed.
- Node.js is more secure than PHP. Actually, the main issue with PHP is security threats and a lot of alternatives have been launched in past few years to overcome with this.

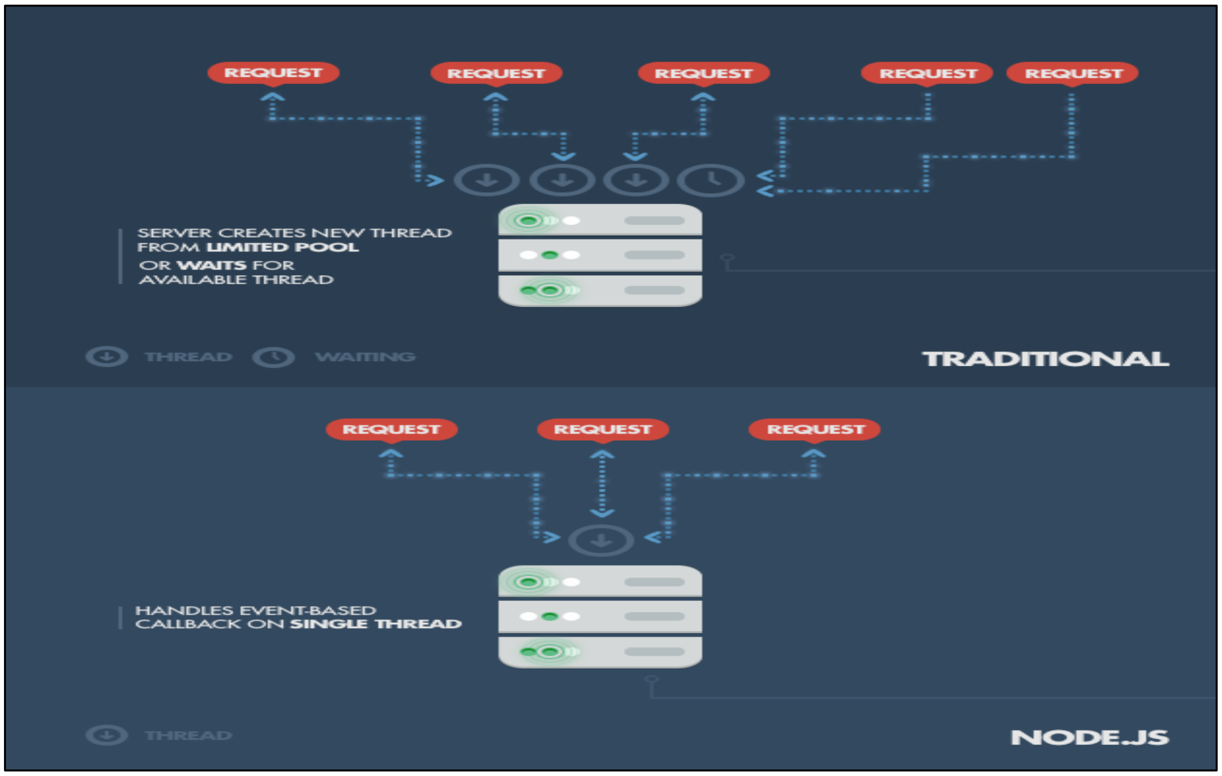
REQUEST
REQUEST
REQUEST
REQUEST
REQUEST
SERVER CREATES NEW THREAD FROM LIMITED POOL OR WAITS FOR AVAILABLE THREAD
THREAD
WAITING
TRADITIONAL
REQUEST
REQUEST
REQUEST
HANDLES EVENT-BASED CALLBACK ON SINGLE THREAD
THREAD
NODE.JS

Fig 4.i

4.2 NPM: Node Package Manager

When discussing Node.js, one thing that definitely should not be omitted is built-in support for package management using NPM, a tool that comes by default with every Node.js installation.

Some of the most useful npm modules today are:

- express - Express.js—or simply Express—a Sinatra-inspired web development framework for Node.js, and the de-facto standard for the majority of Node.js applications out there today.
- **hapi** - a very modular and simple to use configuration-centric framework for building web and services applications
- connect - Connect is an extensible HTTP server framework for Node.js, providing a collection of high performance "plugins" known as middleware; serves as a base foundation for Express.
- socket.io and sockjs - Server-side component of the two most common web sockets components out there today.
- **pug** (formerly **Jade**) - One of the popular templating engines, inspired by HAML, a default in Express.js.
- mongoDB and mongojs - MongoDB wrappers to provide the API for MongoDB object databases in Node.js.

Development Focus

- Three in four Node.js users are focused primarily on back-end or full stack development.
- There has been a slight drop in this wave in those who have any focus on back-end development.

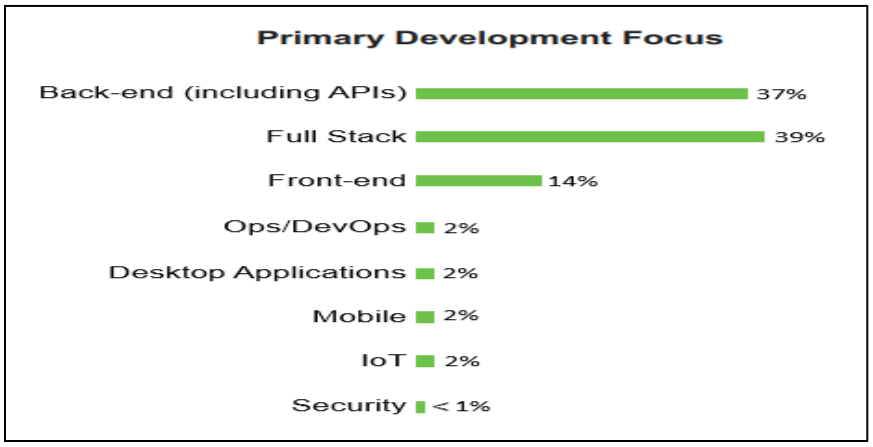

Fig 4.ii

4.3 Applications of Node.js

▪ APPLICATION MONITORING DASHBOARD

Another common use-case in which Node-with-web-sockets fits perfectly: tracking website visitors and visualizing their interactions in real-time.

we would be gathering real-time stats from our user, or even moving it to the next level by introducing targeted interactions with our visitors by opening a communication channel when they reach a specific point in our funnel.

Imagine how we would improve your business if we knew what our visitors were doing in real-time—if we could visualize their interactions. With the real-time, two-way sockets of Node.js, now you can.

▪ SYSTEM MONITORING DASHBOARD

Now, let's visit the infrastructure side of things. Imagine, for example, an SaaS provider that wants to offer its users a service-monitoring page, like GitHub's status page. With the Node.js event-loop, we can create a powerful web-based dashboard that checks the services' statuses in an asynchronous manner and pushes data to clients using websockets.

Both internal (intra-company) and public services' statuses can be reported live and in real-time using this technology. Push that idea a little further and try to imagine a Network Operations Center (NOC) monitoring applications in a telecommunications operator, cloud/network/hosting provider, or some financial institution, all run on the open web stack backed by Node.js and websockets instead of Java and/or Java Applets.

4.4 MongoDB:

MongoDB is an open-source document-oriented database. It is used to store a larger amount of data and also allows you to work with that data. MongoDB is not based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data, that’s why it is known as NoSQL database. Here, the term ‘NoSQL’ means ‘non-relational’. The format of storage is called BSON (similar to JSON format).

▪ Why do we need NoSQL with Node.js?

Document-based data storage is the main aim of using a non-structured database like NoSQL. MongoDB is a distributed database which allows ad-hoc queries, real-time integration, and indexing efficiently. Moreover, MongoDB is open-source and perfect for frequently changing data. It also offers server-side data validation.

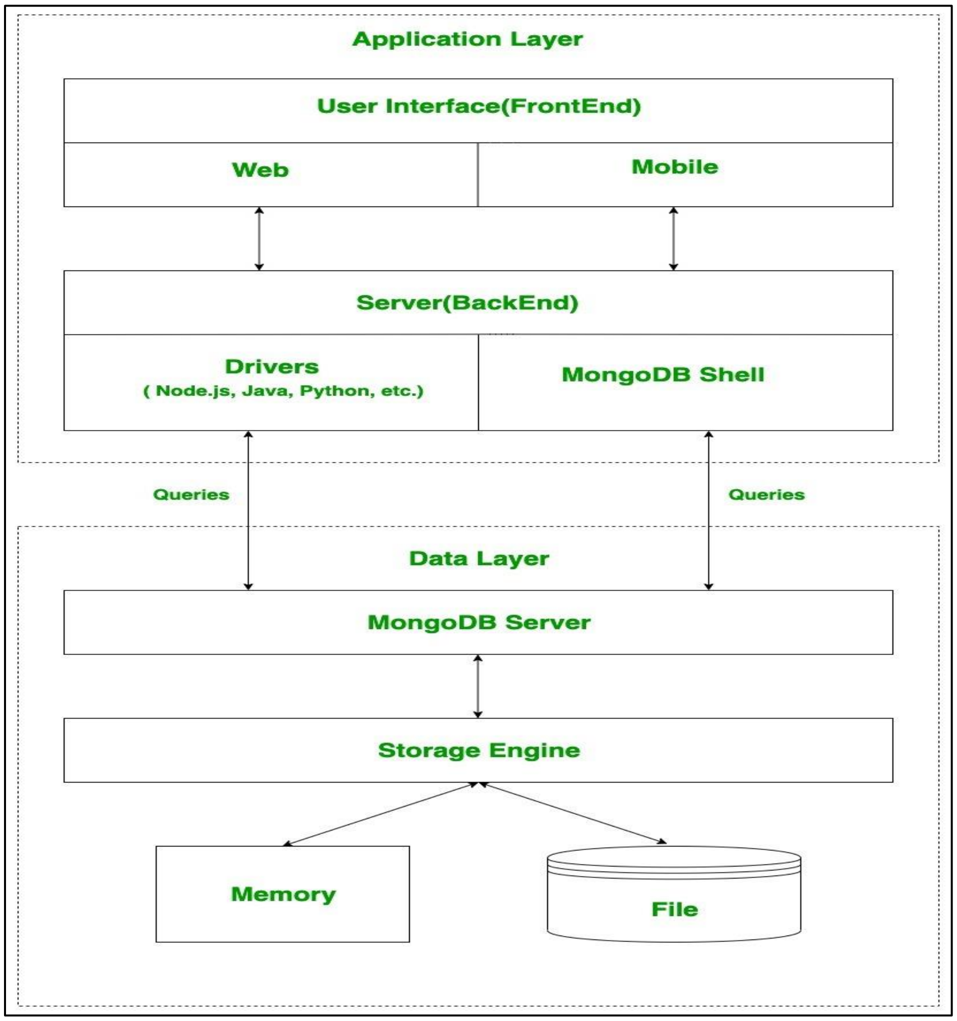

Fig 4.iii. Working of MongoDB

▪ Combining NodeJS and NoSQL- Why MongoDB is the best choice?

Node.js is popularly being used in web applications because it lets the application run while it is fetching data from the backend server. It is asynchronous, event-driven and helps to build scalable web applications. Even though Node.js works well with a MySQL database, the perfect combination is a NoSQL like MongoDB wherein the schema need not be well-structured. MongoDB represents the data as a collection of documents rather than tables related by foreign keys. This makes it possible for the varied types of data dealt over the internet to be stored decently and accessed in the web applications using Node.js. Another option is using CouchDB that also stores the data as a JSON/BSON environment.

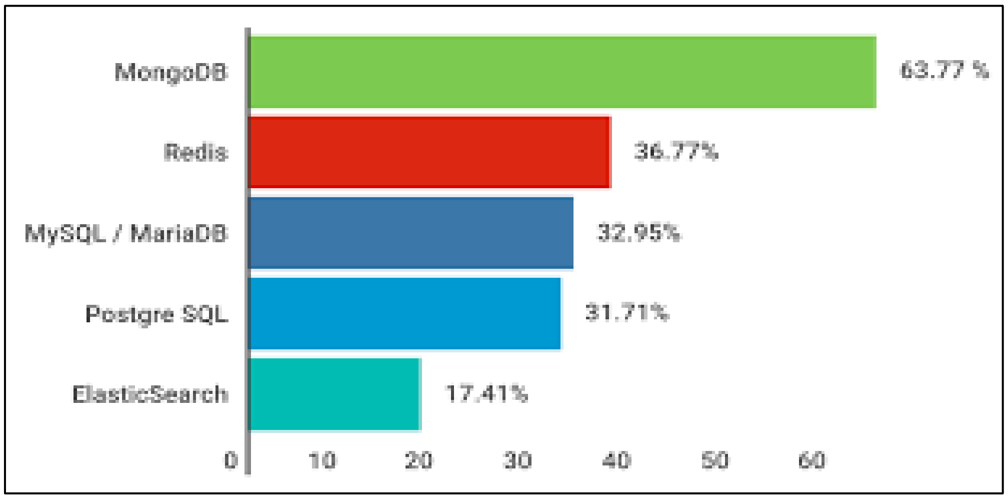

Fig 4.iv Survey Report of using MongoDB with Node.js

4.5 Pros and Cons of using mongoDB and node.js as a backend

Pros:

- Node uses V8 JavaScript Runtime engine, which makes it really fast. It is much faster than PHP, Ruby, Python, or Perl.
- It uses a single programming language (JavaScript) on both the client-side and server-side, resulting in increased code reusability.
- Open source NPM repository of over 50,000 packages helps developers to create full featured Node.js applications.
- It is open source and completely free to use.
- Cross platforms (Windows, Linux, Unix, Mac OS X, etc.) support.
- It Supports Caching for individual modules.
- It is easy to build RESTful APIs for database support JSON such as MongoDB or CouchDB.

Cons:

- Node.js is not suited for CPU-intensive operations.
- It is difficult to deal with relational databases in Node.js.
- It doesn’t support multi-threaded programming.
- Not Suitable for Large and Complex Web Applications.

Chapter 5: DOCKER

5.1 Introduction:

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

5.2 The Docker platform:

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

5.3 What can I use Docker for?

Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

- Your developers write code locally and share their work with their colleagues using Docker containers.
- They use Docker to push their applications into a test environment and execute automated and manual tests.
- When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
- When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Responsive deployment and scaling

Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

5.4 Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers. Architecture is shown in Fig 5.i.

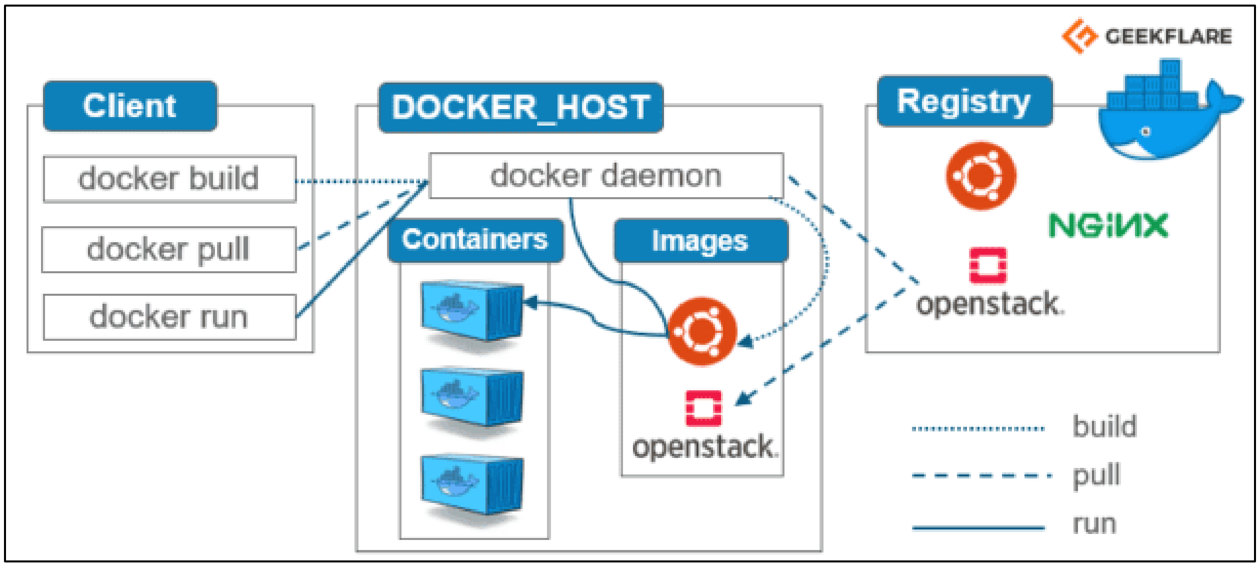

Fig 5.i

The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

- Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

- Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

5.5 Real-World Use Case for VMs

Starling Bank is a digital-only bank that was built in just one year on VMs provided by AWS. This is possible because of the efficiency virtual machines deliver over traditional hardware servers. Importantly, it cost Starling Bank just a tenth of traditional servers.

5.6 Real-World Use Case for Docker

Paypal uses Docker to drive "cost efficiency and enterprise-grade security" for its infrastructure. Paypal runs VMs and containers side-by-side and says that containers reduce the number of VMs it needs to run.

- **Application development**: Docker is primarily used to package an application's code and its dependencies. The same container can be shared from Dev to QA and later to IT, thus bringing portability to the development pipeline.
- **Running microservices applications**: Docker lets you run each microservice that makes up an application in its own container. In this way, it enables a distributed architecture.

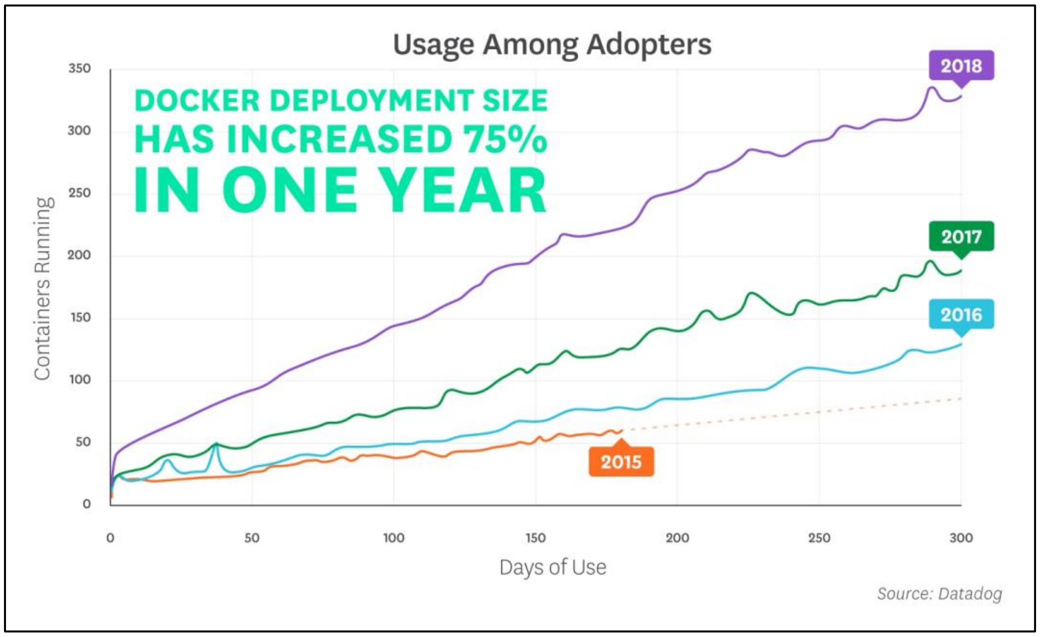

Fig 5.ii

Docker Containers vs. VMs

At last, we arrive at the big question: how are the two different? It all comes down to what you want to do with them. Below, we'll mention a few advantages of Docker as opposed to a virtual machine (specifically Docker vs. VMware), and vice versa.

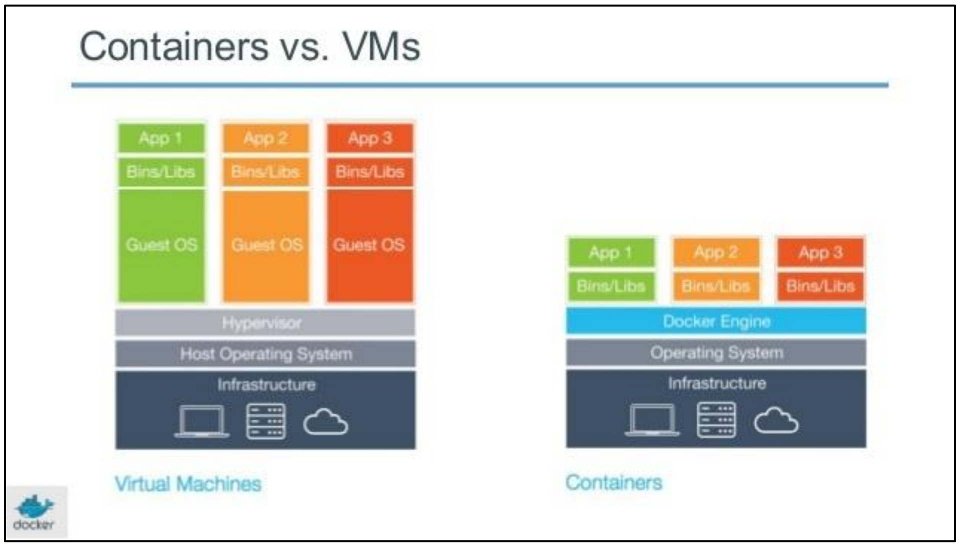

Fig 5.iii

Advantages of Virtual Machines

- The tools associated with a virtual machine are easier to access and simpler to work with. Docker has a more complicated tooling ecosystem, that consists of both Docker-managed and third-party tools.
- As mentioned earlier, once you have a virtual machine up and running, you can start a Docker instance within that VM, and run docker container within the VM (which is the predominant method of running containers at present). This way, containers and virtual machines are not mutually exclusive and can co-exist alongside each other.

Advantages of Docker Containers

- Docker containers are process-isolated and don’t require a hardware hypervisor. This means Docker containers are much smaller and require far fewer resources than a VM.
- Docker is fast. Very fast. While a VM can take an at least a few minutes to boot and be dev-ready, it takes anywhere from a few milliseconds to (at most) a few seconds to start a Docker container from a container image.
- Containers can be shared across multiple team members, bringing much-needed portability across the development pipeline. This reduces ‘works on my machine’ errors that plague DevOps teams.

Chapter 6: KUBERNETES

6.1 Introduction:

▪ Where is Kubernetes used?

Used as a container management tool or application.

▪ What is container management?

Container management refers to a set of practices that govern and maintain containerization software. Container management tools automate the creation, deployment, destruction and scaling of application or systems containers.

▪ Language used to write programs on Kubernetes:

Work is done using Go Language. It is a very reliable open-source programming language.

▪ What is containerization?

Containerization is a process of OS virtualization through which applications are run via isolated user spaces called containers.

▪ How to deploy Kubernetes?

i. Create a cluster
ii. Deploy your app in the cluster
iii. Explore your app
iv. Make your app public
v. Make it open to changes/scale it up
vi. Update your app

- Basic uses of Kubernetes:

It is used to coordinate and run any containerized application across a cluster of machines. It is basically a platform that has been designed to complete the life cycle of any containerized application. The part of the life cycle includes making the app public and open to changes like scaling up while maintaining its basic security against threats such as hacking (high reliability).

6.2 Modular Architecture:

- What is modular architecture?

It is basically a software architecture which involves breaking down a complex problem into simple and small manageable modules. When it is broken down into small modules, it becomes easier to individually look into and solve the problems.

Modular Architecture, as a style, helps us view the system, not just in layers or services, but goes one level below as composition of smaller, physical modules. The keyword here is module which a deployable, reusable, manageable small unit of software.

- What is orchestration?

In system administration, orchestration is the automated configuration, coordination, and management of computer systems and software.

- Why use Kubernetes?

Google built Kubernetes and has been using it for 10 years. That it has been used to run Google's massive systems for that long is one of its key selling points. Two years ago, Google pushed Kubernetes into open source.

Kubernetes is a cluster and container management tool. It lets you deploy containers to clusters, meaning a network of virtual machines. It works with different containers, not just Docker.

The basic idea of Kubernetes is to further abstract machines, storage, and networks away from their physical implementation. So, it is a single interface to deploy containers to all kinds of clouds, virtual machines, and physical machines.

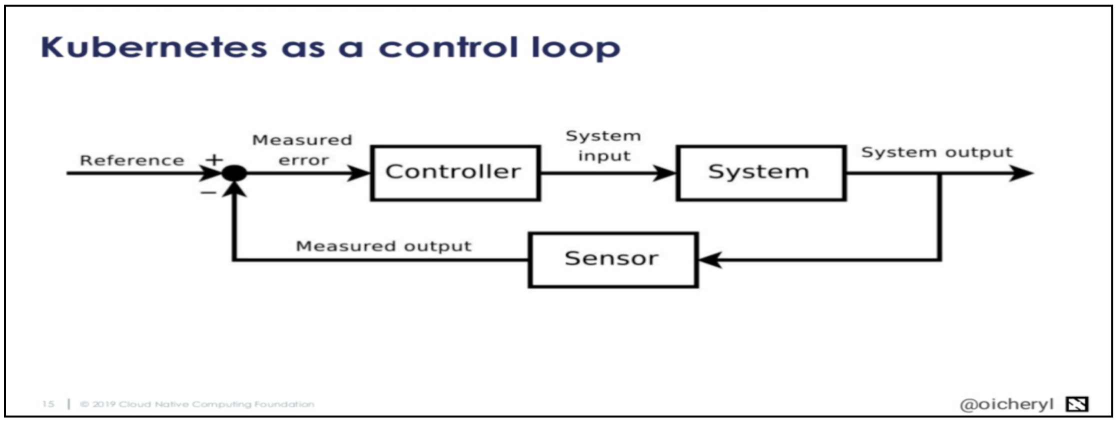

Fig 6.i

- Kubernetes architecture and components:

Kubernetes is made up many components that do not know are care about each other. The components all talk to each other through the API server. Each of these components operates its own function and then exposes metrics that we can collect for monitoring later on. We can break down the components into three main parts.

1. The Control Plane - The Master.
2. Nodes - Where pods get scheduled.
3. Pods - Holds containers.

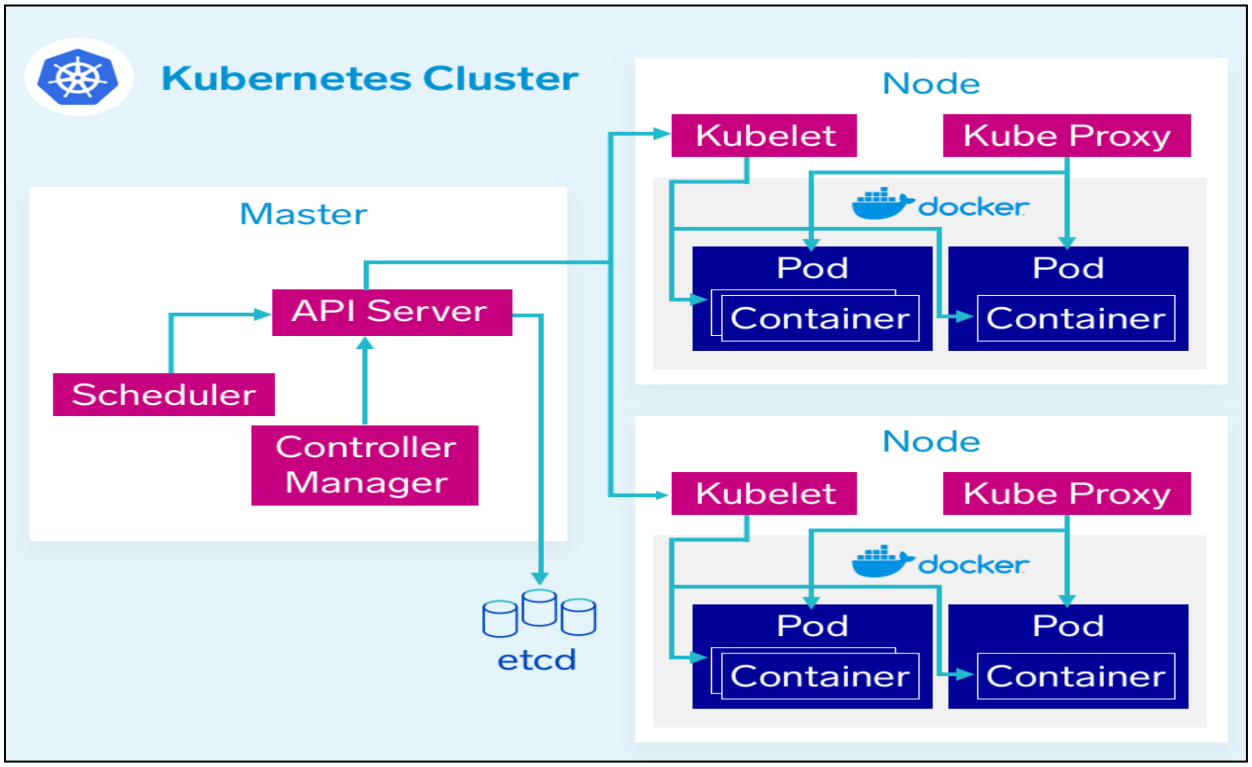

Fig 6.ii

- The Control Plane - The Master Node:

The control plane is the orchestrator. Kubernetes is an orchestration platform, and the control plane facilitates that orchestration. There are multiple components in the control plane that help facilitate that orchestration. ETCD for storage, the API server for communication between components, the scheduler which decides which nodes pods should run on, and the controller manager, responsible for checking the current state against the desired state.

- Nodes:

Nodes make up the collective compute power of the Kubernetes cluster. This is where containers actually get deployed to run. Nodes are the physical infrastructure that your application runs on, the server of VMs in your environment.

- Pods:

Pods are the lowest level resource in the Kubernetes cluster. A pod is made up of one or more containers, but most commonly just a single container. When defining your cluster, limits are set for pods which define what resources, CPU and memory, they need to run. The scheduler uses this definition to decide on which nodes to place the pods. If there is more than one container in a pod, it is difficult to estimate the required resources and the scheduler will not be able to appropriately place pods.

6.3 The relationship between Docker and Kubernetes:

Kubernetes and Docker are both comprehensive de-facto solutions to intelligently manage containerized applications and provide powerful capabilities, and from this some confusion has emerged. "Kubernetes" is now sometimes used as shorthand for an entire container environment based on Kubernetes. In reality, they are not directly comparable, have different roots, and solve for different things.

Docker is a platform and tool for building, distributing, and running Docker containers. It offers its own native clustering tool that can be used to orchestrate and schedule containers on machine clusters. Kubernetes is a container orchestration system for Docker containers that is more extensive than Docker Swarm and is meant to coordinate clusters of nodes at scale in production in an efficient manner. It works around the concept of pods, which are scheduling units (and can contain one or more containers) in the Kubernetes ecosystem and they are distributed among nodes to provide high availability. One can easily run a Docker build on a Kubernetes cluster, but Kubernetes itself is not a complete solution and is meant to include custom plug-in.

Kubernetes and Docker are both fundamentally different technologies but they work very well together, and both facilitate the management and deployment of containers in a distributed architecture.

- Can you use Docker without Kubernetes?

Docker is commonly used without Kubernetes; in fact this is the norm. While Kubernetes offers many benefits, it is notoriously complex and there are many scenarios where the overhead of spinning up Kubernetes is unnecessary or unwanted.

In development environments it is common to use Docker without a container orchestrator like Kubernetes. In production environments often the benefits of using a container orchestrator do not outweigh the cost of added complexity. Additionally, many public cloud services like AWS, GCP, and Azure provide some orchestration capabilities making the tradeoff of the added complexity unnecessary.

▪ Can you use Kubernetes without Docker?

As Kubernetes is a container orchestrator, it needs a container runtime in order to orchestrate. Kubernetes is most commonly used with Docker, but it can also be used with any container runtime. RunC, cri-o, containerd are other container runtimes that you can deploy with Kubernetes. The Cloud Native Computing Foundation (CNCF) maintains a listing of endorsed container runtimes on their ecosystem landscape page and Kubernetes documentation provides specific instructions for getting set up using ContainerD and CRI-O.

References:

- https://docs.docker.com/
- https://kubernetes.io/docs/home/
- https://nodejs.org/en/docs/
- https://docs.mongodb.com/
- https://www.w3schools.com/
- https://stackoverflow.com/
- https://dockerhost.com/
- https://mydatabasefriend.com/docs/
- https://newboston.com/
- https://devops.com/
- https://geeksforgeeks.com/