

## 1. Introduction

Developed a Twitter application where user can tweet and see the recent tweet from all the users. Added the news API where the user can scroll through the recent trending news around the world. The tweet created by the user includes the avatar of a person, username, date and the content of the tweet—adding the ability for the other users to like and comment on particular tweets. User can also scroll through the list of users along with the comments on the particular tweet. If any of the tweets contain any of the abusive words, the tweet will be revoked and not visible to any of the users. All the tweets are store to the backend database, which will help the user find their past tweets.

## 2. DevOps in Engineering

In recent years, the service cost to deploy software is been decreased. DevOps is a bridge with set of principle and practices to improve collaboration between software development and IT operation. DevOps in the software used to represent a professional collaboration between software development and Software operations. DevOps methodology is the combination of two-word Dev stands for Development and Ops stand for Operation. DevOps is not really a tool for development or deployment. DevOps is just a methodology to promote development and operation process. As a result of using DevOps methodology, the speed to deliver applications and services increases. DevOps methodology affects the company deployment process, software development and operation processes.

### 8 Principles to achieve DevOps at Scale –

The ultimate goal of any software is to be reliable and faster. To achieve scale in DevOps, we need to manage strategy, we can't just focus on a technical implementation like CICD, we need to follow below 8 principles

**1. You don't need to create a team** – DevOps is th best way to break the silos between your development and operation teams. The technical debt can reduce the velocity in the following ways: - Not adopting the latest performance updates can reduce the customer base. - Not including or providing the latest security updates, this can cause a lot of damage to the company and customer data.

**2. Create a self-organized system** – To achieve goal, give the full insight on team's availability. Give the full access to your ongoing work and work on one task at a time to prevent context switching.

**3. Work closely with agile coaches** – As DevOps methodology is the extension of the agile, we need to work with an agile coach. Make sure the whole team works closely with the DevOps coach, so that the speed for the production is maintain.

**4. Train teams on soft skills** – Focus also on the soft skills of team. Some of the skills need to be developed in the team is the knowledge of GitHub, GitHub commands and OpenSSL.

**5. Empower Teams** – Allow team to learn new things and know what they can do apart from the regular work.

**6. Uncertainty is your friend** – Allow a team to engage in group to work on real, end-user performance indicators. Let the team decide how to achieve goals and manage how to handle any failure.

**7. Technical implementation** – Align your IT goals with Business goals. The need for implementation of DevOps should be business-driven. It should not be implemented just because it is the latest trend, but your development process for the business goals should demand this change.

**8. Get up to date** – Update the stack technology.

### 3. Installation Links

#### Frontend Technologies

- React: <https://react-cn.github.io/react/downloads.html>
- React Native: <https://reactnative.dev/>
- Command to create react native project- npx react-native init {Application Name}

#### Backend Technologies

- Node Js: <https://nodejs.org/en/download/>
- Mongo DB: [MongoDB Download Link](#)
- Redis Server: <https://redis.io/download>

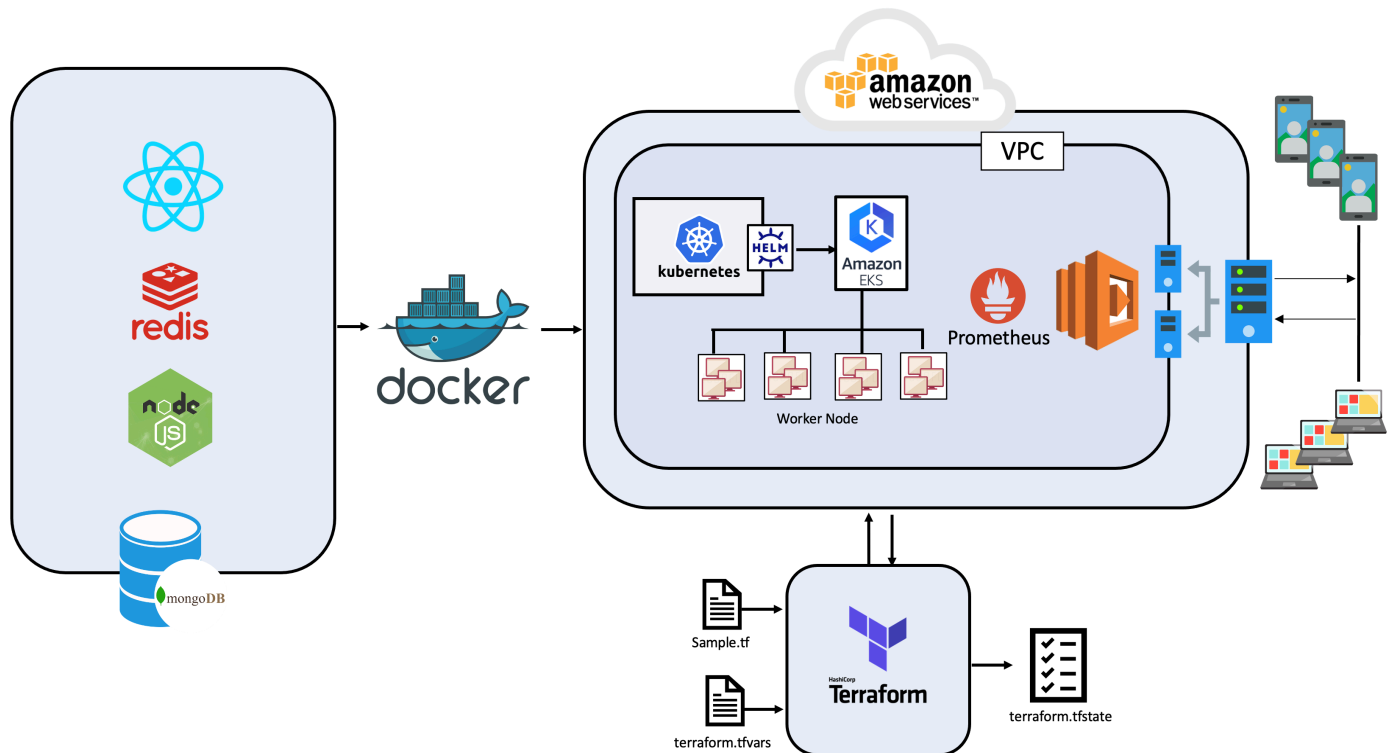
#### Server Technologies

- Docker CLI: <https://docs.docker.com/docker-for-mac/install/>
- Kubernetes: <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- EKS: <https://minikube.sigs.k8s.io/docs/start/>
- Helm Chart: [https://helm.sh/docs/helm/helm\\_pull/](https://helm.sh/docs/helm/helm_pull/)
- AWS CLI: <https://awscli.amazonaws.com/AWSCLIV2.msi>
- Prometheus: <https://prometheus.io/download/>
- Terraform: <https://www.terraform.io/downloads.html>

#### Logging Technologies

-

#### 4. Project Architecture



The Front end is created in the React Native, Redis JSON is working as the middleware for the system. All the requests are clustered in the middleware and after every interval it is transferred to the backend i.e., Node server. The Node server is connected with the MongoDB for saving and fetching user data.

All of the layers are separately containerized in the Docker Container. The Docker containers are further deployed as the Kubernetes pods.

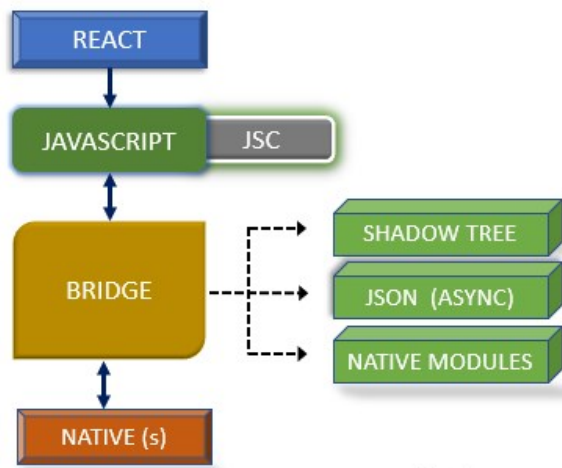
The Kubernetes pods are deployed on the AWS EKS using the Terraform. The Prometheus server is connected to the AWS to track down the request load on the server and show the details for the usage on the Dashboard.

All the incoming requests from the client are load balanced on the AWS using the ELB service of the AWS.

## 5. Frontend

The frontend application is created in React-Native, which enables the application to run on both Mobile and webapp.

React Native is a JavaScript framework used for developing a real, native mobile application for iOS and Android. It uses only JavaScript to build a mobile application. It is like React, which uses native component rather than using web components as building blocks.



### Login Page

User can login to the existing account.

```
// SignIn function definition
export default function SignIn({ changeActiveRoute, history }) {
  const classes = useStyles();

  // State for username and password
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  // State to show auth failed/succeeded
  const [isUserEntryValid, setIsUserEntryValid] = useState(true);

  // Authenticate a user using API endpoint
  const authenticateUser = async (e) => {
    try {
      setIsUserEntryValid(true);
      e.preventDefault();

      // post api call trigger to verify user identity
      const userRequest = {
        username: username,
        password: password,
      };

      const userInfoReceived = await authenticateUserInfo(userRequest);
      userInfoReceived.displayName = userInfoReceived.firstName
        .concat(" ")
        .concat(userInfoReceived.lastName);

      // Set session for user
      setSessionCookie(userInfoReceived);
      history.push("/home");
    } catch (error) {
      console.log(error);
      setIsUserEntryValid(false);
    }
  };
};
```

//Screenshot of Login Page

**Registration Page**

Create a new user just by entering Name and Email ID.

Password field on the Login and Registration page is Token based authentication.

```
// Function to register a user using Register API
const registerUser = async (e) => {
  try {
    setIsUserEntryValid(true);
    e.preventDefault();

    // post api call trigger to register user
    const userRequest = {
      firstName: firstName,
      lastName: lastName,
      username: username,
      password: password,
    };

    // register API call
    await registerUserInfo(userRequest);
    // move to login screen
    changeActiveRoute();
  } catch (error) {
    console.log(error);
    setIsUserEntryValid(false);
  }
};
```

//Screenshot of Registration Page

**Home Page**

```
21 lines (20 sloc) 488 Bytes
1  /**
2   * Component Responsible to hold all the 3 major
3   * components Sidebar, Feed, Widgets
4   */
5   import React from "react";
6   import Feed from "../Feed/Feed";
7   import Sidebar from "../Sidebar/Sidebar";
8   import Widgets from "../Widgets/Widgets";
9   import "../App.css";
10  // Home function definition
11  function Home({ userInfo }) {
12    return (
13      <div className="app">
14        <Sidebar userInfo={userInfo} />
15        <Feed userInfo={userInfo} />
16        <Widgets />
17      </div>
18    );
19  }
20
21  export default Home;
```

User can Tweet what's happening around.  
User will not be able to post abuse words.  
User can like any of the tweets and can also comment on tweets.  
Included BBC News API to show the recent news.  
Logout from the account.

//Screenshot of the Home Page

**In the project directory, you can run:**

**npm start**

Runs the app in the development mode.  
Open <http://localhost:3000> to view it in the browser.

The page will reload if you make edits.  
You will also see any lint errors in the console.

**npm test**

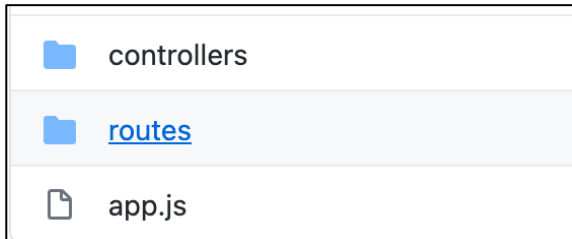
Launches the test runner in the interactive watch mode.  
See the section about running tests for more information.

**npm run build**

Builds the app for production to the build folder.  
It correctly bundles React in production mode and optimizes the build for the best performance.  
The build is minified, and the filenames include the hashes.  
Your app is ready to be deployed!

## 6. Redis Server

The Redis server will act as the middleware for the application. All the request from the frontend will be forwarded to the Redis server and it will as the queuing part.



The Set interval method is used to queue the api calls and after every 6000ms it will be pushed to the backend server.

```
setInterval(clearCache, 1000 * 60 * cacheTimeOut);

// Function used to clear cache after every cacheTimeOut minutes and calls the main backend to
// update the records
function clearCache() {
  // get the values from cache
  client.lrange(myKey, 0, -1, function (err, reply) {
    if (!err) {
      const tweets = reply.map(JSON.parse);
      if (tweets && tweets.length > 0) {
        // call the node server to dump the data
        fetch(`${baseUrl}/tweets`, {
          method: "POST",
          body: JSON.stringify(tasks),
          headers: { "Content-Type": "application/json" },
        })
          .then((res) => res.json())
          .then((json) => console.log(json));
      }
    }
  });
}
```

### Redis Server Code

```
const express = require("express"),
  app = express(),
  bodyParser = require("body-parser"),
  port = process.env.PORT || 3000,
  cors = require("cors");

// enable cors
app.use(cors());

//Adding body parser for handling request and response objects.
app.use(
  bodyParser.urlencoded({
    extended: true,
  })
);
app.use(bodyParser.json({ limit: "50mb" }));
app.use(
  bodyParser.urlencoded({
    limit: "50mb",
    extended: true,
    parameterLimit: 50000,
  })
);

//Initialize app
let initApp = require("./api/app");
initApp(app);

app.listen(port);
console.log(`ChatterApp Redis server started on: ${port}`);
```

## 7. Backend

The backend of the application is connected with the Redis Server, where all the request is gathered and further it is transported to the backend Node JS server.

Server Code:

```
const utilConstants = require("../api/utils/Constants");
const express = require("express"),
    app = express(),
    port = process.env.PORT || utilConstants.PORT,
    mongoose = require("mongoose"), //created model loading here
    bodyParser = require("body-parser");
const cors = require("cors");

// Mongo Atlas
const uri = utilConstants.MONGODB_URL;
mongoose.connect(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
  promiseLibrary: global.Promise,
})
.then(() => {
  console.log("MongoDB Connected...")
})
.catch(err => console.log(err))

// enable cors
app.use(cors());







//Adding body parser for handling request and response objects.
app.use(
  bodyParser.urlencoded({
    extended: true,
  })
);
app.use(bodyParser.json({ limit: "50mb" }));
app.use(
  bodyParser.urlencoded({
    limit: "50mb",
    extended: true,
    parameterLimit: 50000,
  })
);

//Initialize app
let initApp = require("../api/app");
initApp(app);

app.listen(port);
console.log("ChatterApp RESTful API server started on: " + port);
```



**Server API folder Structure**

	controllers
	models
	routes
	services
	utils
	app.js

The Server Controllers API contains the Tweet methods which are responsible for creating and displaying tweets, adding comments to the tweet, adding like to the tweet and User controller methods which are responsible for creating a new user or fetching the existing user details.

The Server Model API contains the Schema of the tweet and user data.

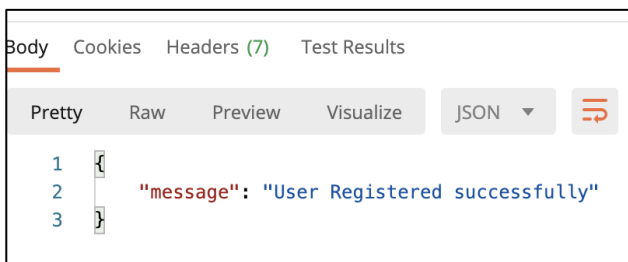
The Route services API contains the routes throw which the data will be communicated between frontend and backend.

We have created the following endpoint for the backend communication –

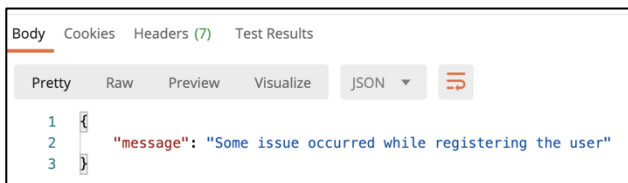
Create User: Post method - <http://localhost:3000/v1/register>

```
{
  "username": "ravi123",
  "password": "Admin@123",
  "firstName": "ravi",
  "lastName": "test1"
}
```

If the user is successfully created, the server will respond with the success message.



If there is some issue while creating the user, the server will respond with the failure message.



User Login: Post Method - <http://localhost:3000/v1/authenticate>

```
{
  "username": "ravi123",
  "password": "Admin@123"
}
```

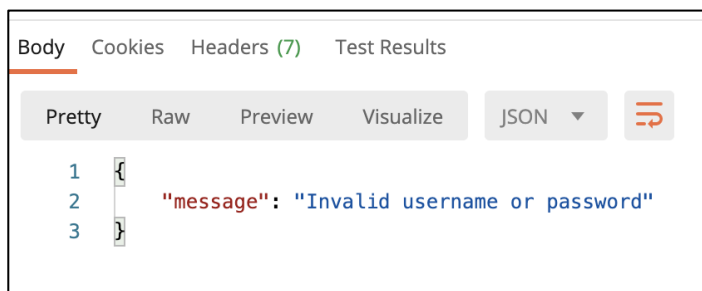
If the credentials provided are correct, the server will respond with the success message as follow



A screenshot of a REST client interface showing a successful login response. The 'Body' tab is selected, and the response is displayed in JSON format. The status is 200 OK, with a response time of 114 ms and a size of 562 B. The JSON object contains user details: username, firstName, lastName, createdAt, id, and token.

```
1 {
2   "username": "ravi123",
3   "firstName": "ravi",
4   "lastName": "test1",
5   "createdAt": "2020-12-06T03:34:54.186Z",
6   "id": "5fcc515e6b3aed35842e3d59",
7   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1ZmNjNTE1ZTZiM2FLZDM1ODQyZTNkNTkiLCJpYXQiOiJlE2MDcyMjc5NDQsImV4cCI6MTYwNzgzMjc0NH0GEqHgUcHNK8YjZjsI-qadSWYVnn4Z2nxEi50o2MapVA"
8 }
```

If the credentials provided are not correct, the server will respond with the failure message



A screenshot of a REST client interface showing a failure response. The 'Body' tab is selected, and the response is displayed in JSON format. The status is 200 OK, with a response time of 114 ms and a size of 562 B. The JSON object contains a message: "Invalid username or password".

```
1 {
2   "message": "Invalid username or password"
3 }
```

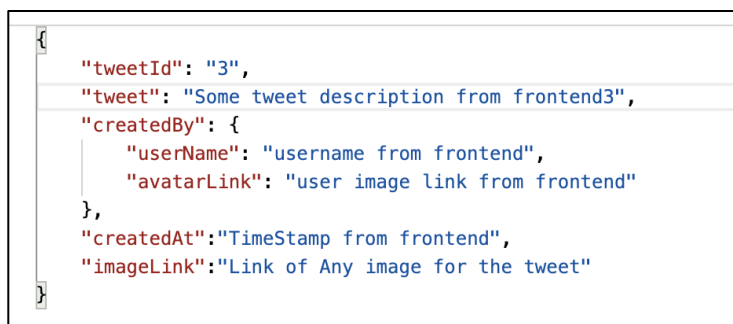
Creating Tweet: Post method - <http://localhost:3000/v1/tweets>

The parameters used for the creating tweets are as below –

TweetID – uuid

CreatedBy – Name of the user who created the tweet

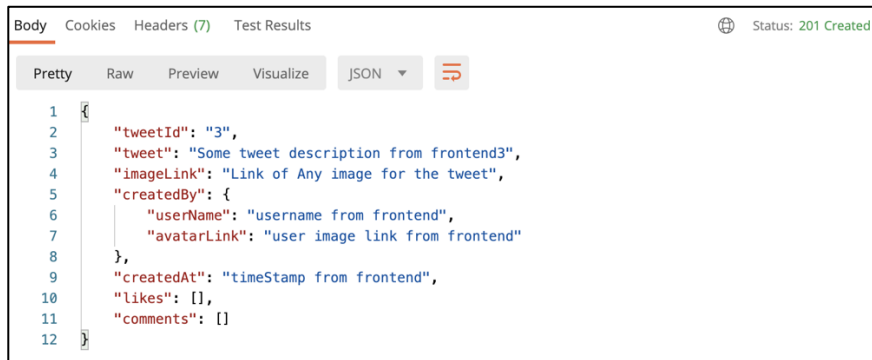
CreatedAt – TimeStamp



A screenshot of a REST client interface showing a successful tweet creation response. The 'Body' tab is selected, and the response is displayed in JSON format. The status is 200 OK, with a response time of 114 ms and a size of 562 B. The JSON object contains tweet details: tweetId, tweet, createdBy (with userName and avatarLink), createdAt, and imageLink.

```
1 {
2   "tweetId": "3",
3   "tweet": "Some tweet description from frontend3",
4   "createdBy": {
5     "userName": "username from frontend",
6     "avatarLink": "user image link from frontend"
7   },
8   "createdAt": "TimeStamp from frontend",
9   "imageLink": "Link of Any image for the tweet"
10 }
```

If the entered data validated and the tweet is stored in the database, the server will acknowledge using the below success message.

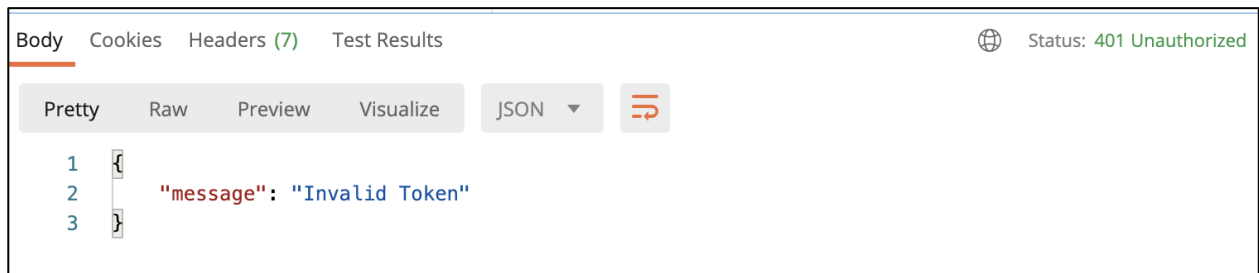


Body Cookies Headers (7) Test Results Status: 201 Created

Pretty Raw Preview Visualize JSON

```
1 {
2   "tweetId": "3",
3   "tweet": "Some tweet description from frontend3",
4   "imageLink": "Link of Any image for the tweet",
5   "createdBy": {
6     "userName": "username from frontend",
7     "avatarLink": "user image link from frontend"
8   },
9   "createdAt": "timeStamp from frontend",
10  "likes": [],
11  "comments": []
12 }
```

If it fails, the server will acknowledge with the below error message



Body Cookies Headers (7) Test Results Status: 401 Unauthorized

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Invalid Token"
3 }
```

Get List of Tweets: Get Method - <http://localhost:3000/v1/tweets>

The server will provide the list of all tweet in the system



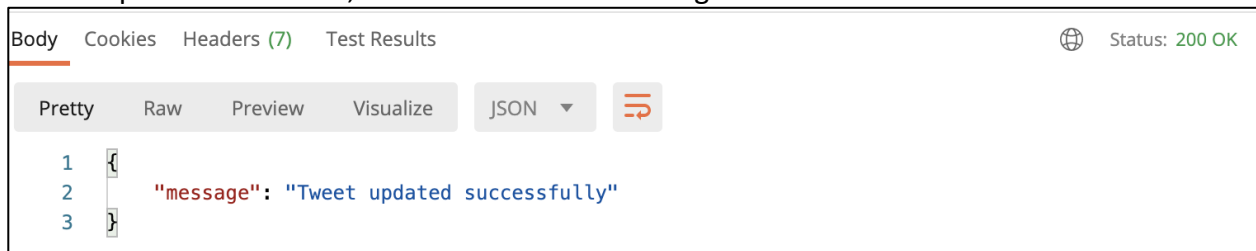
```
[
  {
    "likes": [],
    "tweetId": "1",
    "tweet": "Some tweet description from frontend1",
    "imageLink": "",
    "createdBy": {
      "userName": "username from frontend",
      "avatarLink": "user image link from frontend"
    },
    "createdAt": "timeStamp from frontend",
    "comments": [
      {
        "commentedBy": "name of the other user1",
        "avatarLink": "imageLinkFor the other user",
        "comment": "Testing comments on tweet1",
        "commentId": "id from fe"
      }
    ]
  },
  {
    "likes": [
      "id of the user"
    ],
    "tweetId": "2",
    "tweet": "Some tweet description from frontend2",
    "imageLink": "image link for tweet 2",
    "createdBy": {
      "userName": "username from frontend",
      "avatarLink": "user image link from frontend"
    },
    "createdAt": "timeStamp from frontend",
    "comments": []
  }
]
```

```
{
  "tweetId": "2",
  "tweet": "Some tweet description from frontend2",
  "imageLink": "",
  "createdBy": {
    "userName": "username from frontend",
    "avatarLink": "user image link from frontend"
  },
  "createdAt": "timeStamp from frontend",
  "comments": []
},
{
  "likes": [],
  "tweetId": "3",
  "tweet": "Some tweet description from frontend3",
  "imageLink": "Link of Any image for the tweet",
  "createdBy": {
    "userName": "username from frontend",
    "avatarLink": "user image link from frontend"
  },
  "createdAt": "timeStamp from frontend",
  "comments": []
}
```

Comment on Tweet: PUT method - <http://localhost:3000/v1/tweetId/comments>

```
{
  "commentedBy": "name of the user",
  "avatarLink": "imageLinkFor the other user",
  "comment": "Testing comments on tweet1",
  "commentId": "id from fe"
}
```

If any of the user comments on the tweet, the frontend will send PUT request to the backend  
If the request is successful, the server will acknowledge with the below code



Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

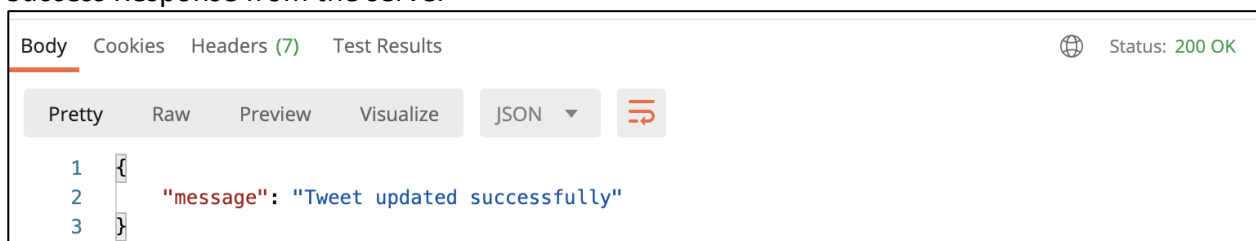
```
1 {
2   "message": "Tweet updated successfully"
3 }
```

Likes on Tweet: PUT method - <http://localhost:3000/v1/tweetId/likes>

If the user likes any of the tweets, the frontend will send request to the backend

```
{
  "userId": "id of the user1",
  "liked": 1
}
```

Success Response from the server –



Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

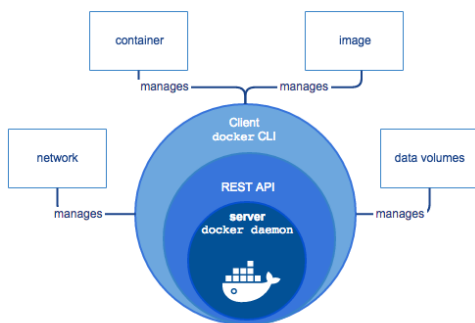
```
1 {
2   "message": "Tweet updated successfully"
3 }
```

## 8. Infrastructure

### Docker

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.



All of the microservices will be containerized as a separate container. Create separate Docker files for frontend, middleware, and Backend.

#### Docker file for React Application

2 lines (2 sloc) | 43 Bytes

```
1 FROM nginx
2 COPY build /usr/share/nginx/html
```

#### Docker file for Redis Server

7 lines (7 sloc) | 118 Bytes

```
1 FROM node:14-slim
2 WORKDIR /usr/src/app
3 COPY ./package.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD [ "server.js" ]
```

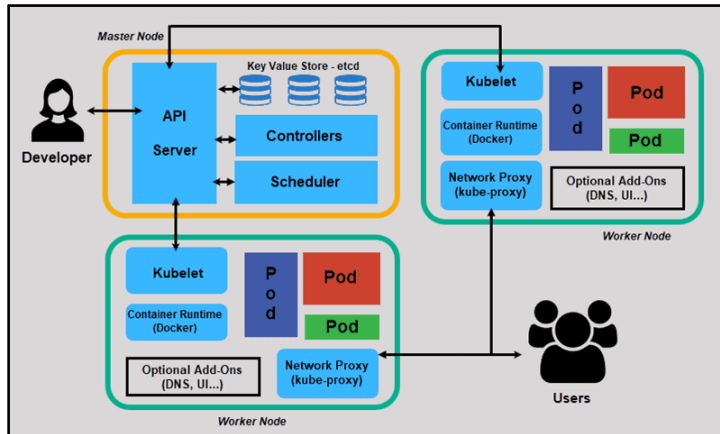
#### Docker file for the Node Server

7 lines (7 sloc) | 118 Bytes

```
1 FROM node:14-slim
2 WORKDIR /usr/src/app
3 COPY ./package.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD [ "server.js" ]
```

## Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.



In Kubernetes, we can expose the services publicly by choosing the type Load Balancer. That will create a public IP address for each service. But we want to reduce the number of IP addresses to make some saving. And we want to map a URL like `mycompany.com/login` and `mycompany.com/products` to the right.

Well, this could be done through Kubernetes Ingress resources.

Kubernetes API doesn't provide an implementation for an ingress controller. So, we need to install it ourselves.

Many ingress controllers are supported for Kubernetes:

1. Nginx Controller
2. HAProxy Ingress, Contour, Citrix Ingress Controller
3. API Gateways like Traffic, Kong and Ambassador
4. Service mesh like Istio
5. Cloud managed ingress controllers like Application Gateway Ingress Controller (AGIC), AWS ALB Ingress Controller, Ingress GCE

The first part will start by configuring Ingress:

1. Installing an ingress controller (NGINX) into Kubernetes.
2. Deploying 2 different applications/services.
3. Configuring ingress to route traffic depending on the URL.

The second part will deal with configuring SSL/TLS using Cert Manager.

In this second part of the lab, we will enable HTTPS in Kubernetes using Cert Manager and Let's Encrypt. The Cert Manager is used to automatically generate and configure Let's Encrypt certificates.

**# Create a namespace for Cert Manager**

```
kubectl create namespace cert-manager
```

**# Get the Helm Chart for Cert Manager**

```
helm repo add jetstack https://charts.jetstack.io  
helm repo update
```

**# Install Cert Manager using Helm charts**

```
helm install cert-manager jetstack/cert-manager `--namespace cert-manager`  
--version v0.14.0 `  
--set installCRDs=true
```

**# Check the created Pods**

```
kubectl get pods --namespace cert-manager
```

**# Install the Cluster Issuer**

```
kubectl apply --namespace app -f ssl-tls-cluster-issuer.yaml
```

**# Install the Ingress resource configured with TLS/SSL**

```
kubectl apply --namespace app -f ssl-tls-ingress.yaml
```

**# Verify that the certificate was issued**

```
kubectl describe cert app-web-cert --namespace app
```

**# Check the services**

```
kubectl get services -n app
```

# Now test the app with HTTPS: <https://frontend.<ip-address>.nip.io>

**# Cleanup resources**

```
helm delete cert-manager --namespace cert-manager  
kubectl delete namespace cert-manager  
kubectl delete --namespace app -f ssl-tls-cluster-issuer.yaml  
kubectl delete --namespace app -f ssl-tls-ingress.yaml
```



## Helm Chart

Helm is a package Manager for Kubernetes. Package managers (like yum and APT) are a big reason many of the popular Linux distributions are as successful as they are. They provide a standard, opinionated way to install, configure, upgrade, and run an application in a matter of minutes. And the packages themselves are open source, and anyone can contribute to them.

Helm is made of two components: A server called Tiller, which runs inside your Kubernetes cluster and a client called helm that runs on your local machine. A package is called a chart to keep with the maritime theme.

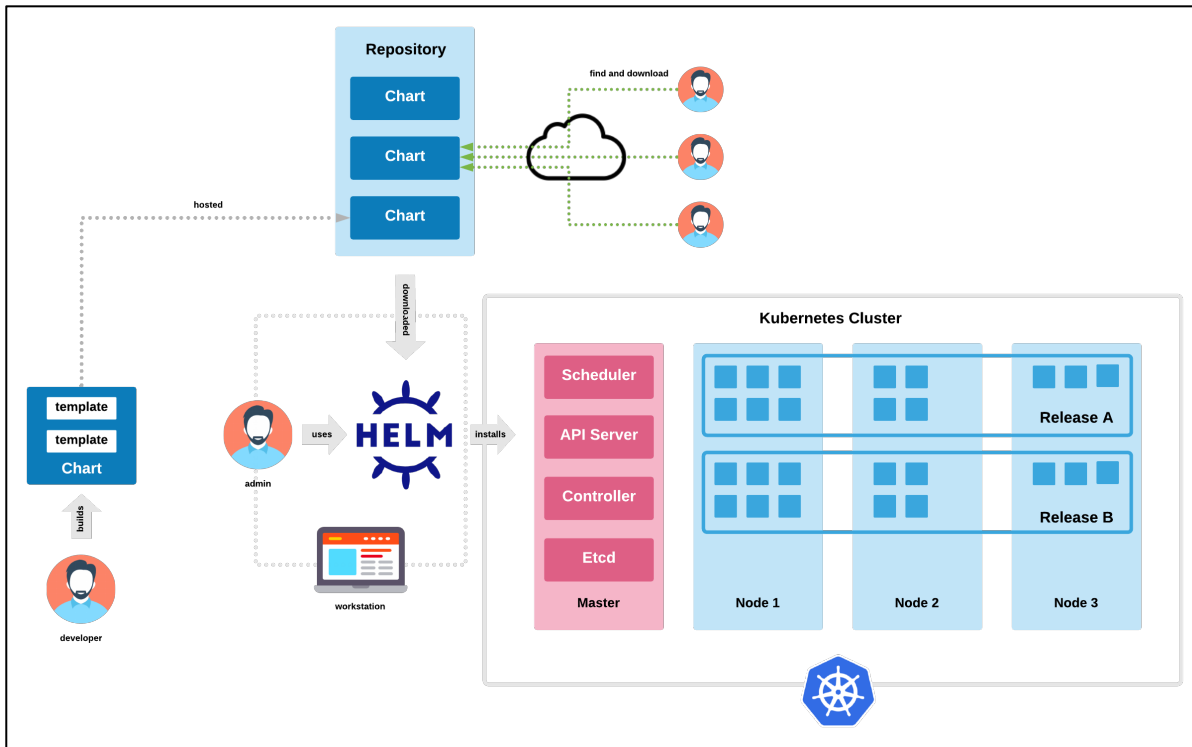
With the Helm client, you can browse package repositories (containing published Charts) and deploy those Charts on your Kubernetes cluster. Helm will pull the Chart and talking to Tiller will create a *release* (an instance of a Chart).

## Structure of a Chart

A Chart is easy to demystify; it is an archive of a set of Kubernetes resource manifests that make up a distributed application.

```
...  
  
.  
├─ Chart.yaml  
├─ README.md  
├─ templates  
|   └─ NOTES.txt  
|   └─ _helpers.tpl  
|   └─ configmap.yaml  
|   └─ deployment.yaml  
|   └─ pvc.yaml  
|   └─ secrets.yaml  
|   └─ svc.yaml  
└─ values.yaml  
...
```

1. Chart.yaml = contains some metadata about the Chart, such as its name, version, keywords.
2. values.yaml=contains keys and values that are used to generate the release in your Cluster. These values are replaced in the resource manifests.
3. configMap.yaml=contains database configuration.
4. secrets.yaml= database passwords are stored in secret file.



### Commands to Run:

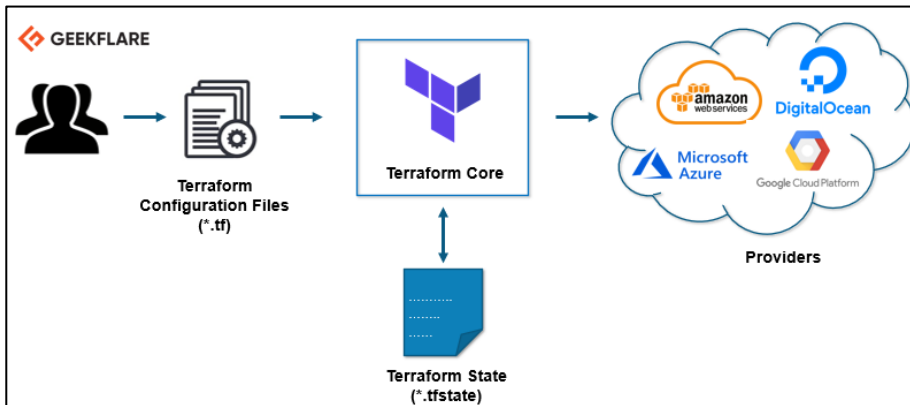
Check all the values in the Helm chart are correct - **helm lint ./twitterhelmchart**

Dry Run (Check everything is running well without deploying actual services) – **helm install --dry-run --debug ./twitterhelmchart --generate-name**

Run the Helm chart - **helm install example ./twitterhelmchart/ --set service.type=NodePort**

## Terraform

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, terraform is able to determine what changed and create incremental execution plans which can be applied.



## Ingress Controller

Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster. However, in real-world Kubernetes deployments, there are frequently additional considerations beyond routing for managing ingress. We'll discuss these requirements in more detail below.

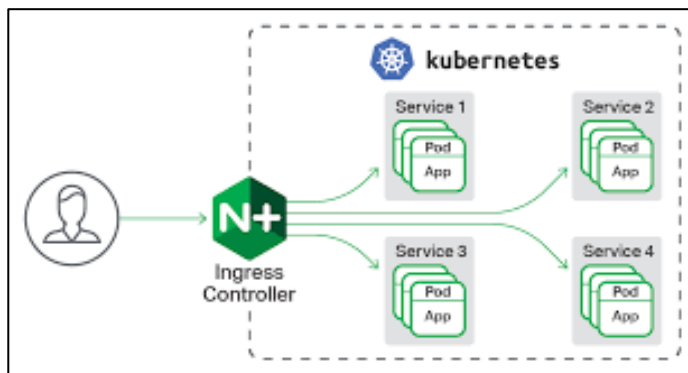
## Ingress in Kubernetes

In Kubernetes, there are three general approaches to exposing your application.

Using a Kubernetes service of type Node Port, which exposes the application on a port across each of your nodes.

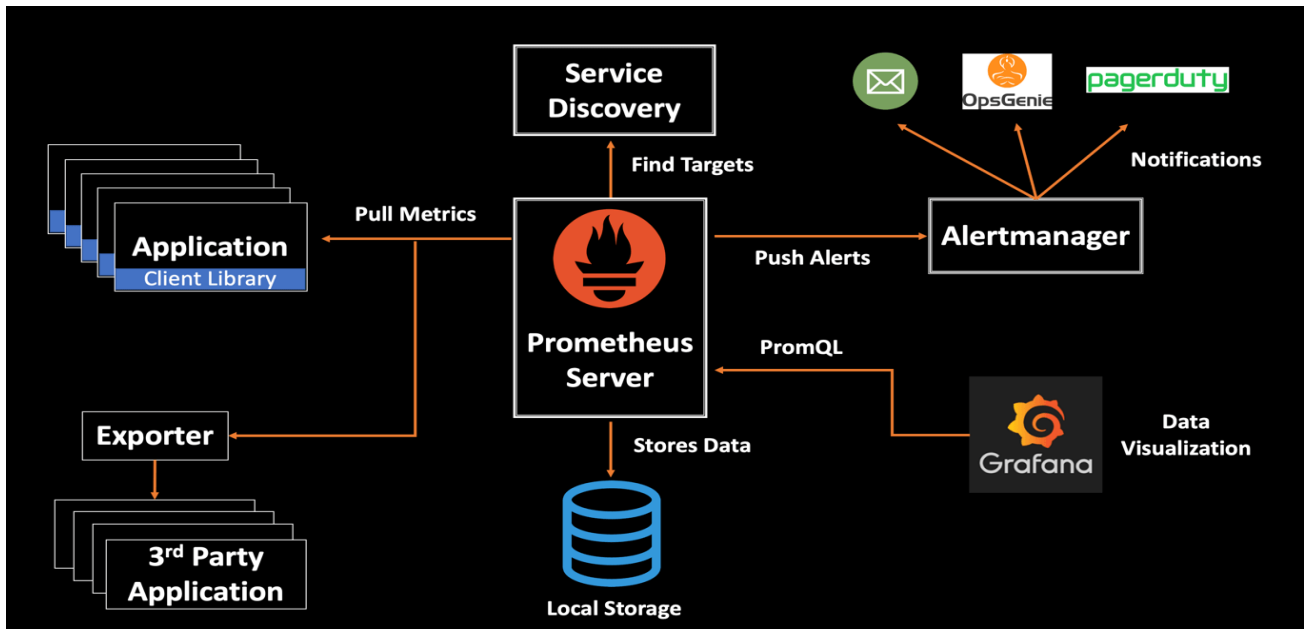
Use a Kubernetes service of type Load Balancer, which creates an external load balancer that points to a Kubernetes service in your cluster

Use a Kubernetes Ingress Resource



## Prometheus Server

Prometheus is an open-source system monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community.



## Prometheus Service yaml file

15 lines (15 sloc) | 290 Bytes

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: prometheus-service
5    namespace: monitoring
6    annotations:
7      prometheus.io/scrape: 'true'
8      prometheus.io/port: '9090'
9  spec:
10   selector:
11     app: prometheus-server
12   type: LoadBalancer
13   ports:
14     - port: 80
15       targetPort: 9090
```

//Screenshot of Our dashboard and description of Dashboard

**Command Guide****Frontend application**

- Special installation needed:
- Commands to Run:
  - Open the Application Folder Chatter and Navigate to the frontend folder
  - `cd frontend`
  - `npm install`
  - `npm start`
  - Navigate to the link:

**Redis Server**

- Special installation needed:
- Commands to Run Redis server:
  - `cd redis`
  -

**Configuring MongoDB**

- Special installation needed:
- Commands to setup DB:
  -

**Backend Server**

- Special installation needed:
- Commands to Run Node Server:
  - `cd server`
  - `node server`
  - Navigate to the link:

**Docker**

- Special installation needed:
- Commands to Run:
  -

**Kubernetes**

- Special installation needed:
- Commands to Run:
  -

**Amazon EKS**

- Special installation needed:
- Commands to Run:
  -

**Prometheus Server**

- Special installation needed:
- Commands to Run:
  -