POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERI INFORMATICA
CORSO CYBERSECURITY AND CLOUD

Business Report

Progetto: avNET

Control of Network Systems

Business Report avNET

Docente:
Prof. Saverio Mascolo
Ing. Carlo Croce

Componenti gruppo:
Detomaso Riccardo,
Iannone Alessandro,
Silvestri Giovanni,
Tangari Mauro

Anno Accademico 2022/2023

Business Report avNET

EXECUTIVE SUMMARY

Scopo & Obiettivi:

Lo scopo del progetto “avNET” è quello di **realizzare** e **gestire** dei **percorsi** per velivoli. Infatti a seguito dell’inserimento di una serie di coordinate, è possibile generare un percorso e caricarlo in un database MongoDB. L’**obiettivo** finale è quello di effettuare una simulazione che testi l’integrazione di un **veicolo** con i dati salvati sul file JSON presente nel nostro database.

Partendo da un’idea concordata con il Prof. Mascolo e guidati dall’Ing Croce, “avNET” è stato ideato e realizzato da un gruppo di 4 studenti dell’università “Politecnico di Bari”, costituito in particolare da:

1) Iannone Alessandro: Full Stack Developer
2) Silvestri Giovanni: Frontend Developer
3) Detomaso Riccardo: Frontend Developer
4) Tangari Mauro: Frontend Developer.

La nostra Web App si divide in 4 obiettivi principali:

1) **Frontend:** permettere di inserire le coordinate delle Control Units e di genereare un percorso, attraverso una mappa o semplicemente scrivendole in una casella di testo. Nello specifico le Control Units sono delle centraline create in base alla loro posizione (latitudine e longitudine), mentre il percorso contiene le coordinate di una serie di centraline precedentemente memorizzate. Inoltre è anche possibile tenere traccia di tutti i percorsi creati tramite una History così che, in caso si voglia rieseguire un percorso già salvato, basti risalire al nome con il quale è stato memorizzato. Grazie all’ausilio dell’opzione **Backup** è possibile gestire il salvataggio

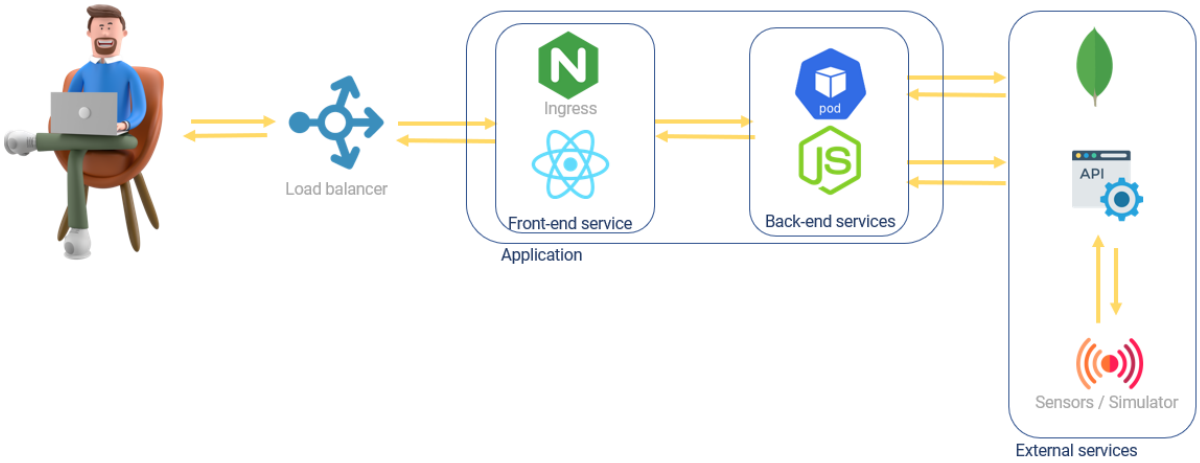
di tutti i file JSON contenenti i percorsi o le coordinate delle singole centraline. Infine tramite il **Simulator** si può seguire un path andando a valutare lo stato di avanzamento del velivolo in tempo reale.

2) **Back-end**: tramite NodeJS si avvia la comunicazione con i servizi esterni, ovvero la connessione con il database MongoDB**,** in cui verranno salvati tutti i dati come file JSON e lo scambio di informazioni tramite API con il veicolo.

   È stato usato **Javascript** per implementare tutta la logica back-end dell'applicativo.

3) **Docker** : permette di isolare la nostra applicazione e le sue dipendenze all'interno di un "container", ovvero un ambiente eseguibile leggero e portatile che contiene tutto ciò di cui l'applicazione ha bisogno per funzionare correttamente. Questo rende la distribuzione dell'applicazione molto più facile e prevedibile, in quanto il container può essere eseguito su qualsiasi macchina che abbia installato Docker senza dover preoccuparsi delle differenze di configurazione del sistema operativo o delle librerie.
4) **Kubernates**: consente di automatizzare la distribuzione, la scalabilità e la gestione dei container da noi realizzati su un'infrastruttura distribuita e scalabile. Il sistema si occupa di gestire la distribuzione dei nostri container su nodi di calcolo, la gestione del networking, la scalabilità e il monitoraggio dell'applicazione.

Architettura del Progetto



Il LoadBalancer gestisce il carico di lavoro per distribuirlo correttamente nei pods.

Esso quindi riceve la richiesta da parte del client e tramite l'Ingress Controller, che mapperà all'Ingress, riesce a restituire il Front-end Services collegato al pod Front-end precedentemente deployato.

Il Front-end services comunica con il Back-end services, che a sua volta è collegato al pod del back-end e inoltre sarà il back-end a comunicare con il nostro database MongoDB.

Infine nel database MongoDB verranno salvati i dati, sottoforma di file JSON, contenenti le info su centraline e percorsi.

Per un maggior approfondimento sulle tecnologie e sulle piattaforme usate, è consigliabile leggere la sezione 1) PIATTAFORME E TECNOLOGIE di cui si parla in seguito.

Come capitale di partenza, nessun membro del team ha dovuto contribuire ad una spesa iniziale per l'avvio del progetto.

CONTENUTI

1) PIATTAFORME E TECNOLOGIE

Frontend

Per il Frontend della Web App abbiamo utilizzato React, una libreria JavaScript open source gestita da Meta per la creazione di interfacce utente basate su componenti.

È stato ideato per consentire, in maniera semplice, lo sviluppo di applicazioni con metodologia «cross-platform», quindi in grado di girare su PC, smartphone e tablet.

Nello specifico, per lo sviluppo della nostra web-app, abbiamo sfruttato la libreria «Material UI», che fornisce una serie di componenti in Material Design.

Perché React?

-è un linguaggio di programmazione JavaScript, semplice, intuitivo e di rapido apprendimento;

-la rappresentazione del componente si traduce in chiamate API a React che intervengono nel modo più rapido ed efficiente possibile sul DOM per creare gli elementi necessari;

-creazione di web app «responsive», ossia il cui design può adattarsi dinamicamente alle dimensioni del display su cui l'applicazione è visualizzata.

Per il frontend sono stati realizzati:

- Un deployment, per generare il pod kubernetes relativo, con il seguente yaml file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: avnet-frontend
spec:
  selector:
    matchLabels:
      app: avnet-frontend
  template:
    metadata:
      labels:
        app: avnet-frontend
    spec:
      containers:
        - name: avnet-frontend
          image: vannisil/avnet-frontend
          ports:
            - containerPort: 3000
```

;

- Un service, ovvero un'astrazione logica per esportare l'accesso alla nostra applicazione:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: NodePort
  selector:
    app: avnet-frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
      name: http
```

Backend

Grazie a NodeJS si avvia la comunicazione con i servizi esterni, ovvero la connessione con MongoDB, in cui verranno salvati tutti i dati come JSON, e lo scambio di informazioni tramite API con il veicolo.

È stato usato Javascript per implementare tutta la logica back-end dell'applicativo.

Anche per il backend sono stati realizzati:

- Un deployment, per generare il pod kubernetes relativo, con il seguente yaml file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: avnet-backend
spec:
  selector:
    matchLabels:
      app: avnet-backend
  template:
    metadata:
      labels:
        app: avnet-backend
    spec:
      containers:
        - name: avnet-backend
          image: vannisil/avnet-backend
          ports:
            - containerPort: 5000
```

- Un service, ovvero un'astrazione logica per esportare l'accesso alla nostra applicazione:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: NodePort
  selector:
    app: avnet-backend
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
```

Docker

È una piattaforma software che permette di creare, testare e distribuire applicazioni in modo rapido ed efficiente. Incorpa il software, realizzato mediante immagini, in container che offrono tutto il necessario per la corretta esecuzione (librerie, strumenti di sistema, codice e runtime).

Offre un modo altamente affidabile e poco costoso per creare ed eseguire applicazioni distribuite su qualsiasi scala. Sono state create due immagini, una per il frontend e una per il backend.

Kubernetes

Può esporre un container docker usando un nome DNS o l'indirizzo IP e permette di montare automaticamente un sistema di archiviazione.

È in grado di distribuire il traffico su più container in modo che il servizio rimanga stabile ed è anche in grado di riavviare i container che si bloccano, terminare quelli che non rispondono

agli health checks, e evitare di far arrivare traffico a quelli non pronti per rispondere correttamente.

Le tecnologie relative a Kubernetes che sono state utilizzate sono:

-i **pods**, le più piccole unità di distribuzione in Kubernetes, che rappresentano un singolo processo in esecuzione all'interno del cluster, generati a partire dai file yaml precedentemente visti

```
ubuntu@ubuntu-server:~$ kubectl get pods
NAME                              READY   STATUS    RESTARTS        AGE
avnet-backend-55b8448cb9-qt2pv    1/1     Running   2 (5m12s ago)   33m
avnet-frontend-6d8c7cdcf9-jnxk4   1/1     Running   2 (8m34s ago)   32m
```

;

-i **services**, astrazioni logiche che esportano l'accesso a un'applicazione o un insieme di pod, attraverso una rete stabile e prevedibile

```
ubuntu@ubuntu-server:~$ kubectl get svc
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
  AGE
backend-service    NodePort       10.111.143.160   <none>        5000:30724/TCP
  6h57m
frontend-service   NodePort       10.101.94.82     <none>        80:30984/TCP
  6h59m
kubernetes         ClusterIP      10.96.0.1        <none>        443/TCP
  7h6m
lb-avnet           LoadBalancer   10.102.216.41    <pending>     3000:30402/TCP
  18s
```

-il **load balancer,** un componente che si occupa di distribuire il traffico di rete in modo equo tra i diversi nodi del cluster di Kubernetes, che è stato generato a partire dal seguente file yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: lb-avnet
spec:
  selector:
    app: avnet-frontend
  ports:
    - port: 3000
      targetPort: 3000
  type: LoadBalancer
```

-l'**ingress**, un'astrazione che esporta le connessioni HTTP e HTTPS dall'esterno del cluster verso i servizi all'interno del cluster, che è stato generato a partire dal seguente file yaml:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: avnet-ingress
spec:
  rules:
  - host: avnet.evizy.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

e visualizzabile mediante riga di comando come segue:

```
ubuntu@ubuntu-server:~$ kubectl get ing
NAME            CLASS    HOSTS              ADDRESS        PORTS   AGE
avnet-ingress   <none>   avnet.epizy.com    192.168.58.2   80      6h42m
```

Visual Studio Code

Visual Studio Code, è un editor di codice sorgente sviluppato da Microsoft che include il supporto per debugging, un controllo per Git integrato e può essere usato con vari linguaggi di programmazione tra cui Javascript e framework come React.

Git Hub

GitHub è un servizio di hosting per progetti software, è una implementazione dello strumento di controllo versione distribuito Git, che permette di tenere traccia delle modifiche e delle versioni apportate al codice sorgente del software. L'abbiamo usato come sistema per la gestione dello sviluppo di software di tipo collaborativo.

Docher Hub

È un registro di container che consente agli utenti di caricare, scaricare e condividere immagini Docker. Gli utenti possono cercare e scaricare immagini Docker di base, come sistemi operativi, software applicativo e altri strumenti, direttamente da Docker Hub. Inoltre,

gli utenti possono caricare le proprie immagini di container su Docker Hub per condividerle con la comunità o per utilizzarle in altri progetti.

Source Tree

SourceTree è un client desktop gratuito per Git. SourceTree offre una varietà di funzionalità, tra cui l'accesso a repository locali e remoti, la visualizzazione di commit, branch e tag, la gestione delle richieste di pull e dei conflitti e molto altro ancora.

2) SVILUPPI FUTURI

FUTURI SVILUPPI

- Integrazione di una **mappa** grazie alla quale l'utente può selezionare una centralina tra quelle disponibili, visualizzandone la posizione geografica;
- Integrazione di **API** grazie alle quali la Web App è in grado di interagire con un **drone reale** per lo scambio di informazioni.