

LiveLink

INDICE

Introduzione	4
1 Interfaccia grafica	5
1.1 StopVideo Button	6
1.2 Mute Button	7
1.3 Chat Button	8
1.4 EndCall Button	9
1.5 Invite Button	9
1.6 Message Section	11
1.7 Send Button	11
1.8 Input Field.....	12
1.9 Message Container	12
1.10 Welcome Modal	13
1.11 Invite People Button	14
1.12 Submit Button	14
1.13 Ok Button	15
1.14 Connection Error	15
2 Sviluppo	16
2.1 Server.js	16
2.2 Script.js	20
3 Docker	30

3.1 Creazione Immagine	31
3.2 Creazione Repository (Docker Hub)	34
3.3 Pubblicazione Immagine Sul Repo	35
3.4 Collegare Heroku A Docker	35
4 Kubernetes	38
4.1 Configurazione di Kubernetes	39
4.2 Kubernetes E Docker Desktop	40
5 Log	46
5.1 Configurazione Di Elasticsearch & Kibana	48
5.2 Configurazione Del Td-Agent	52
Link utili	55

INTRODUZIONE

La comunicazione in tempo reale è diventata un elemento essenziale nella nostra società e le videochiamate hanno rivoluzionato il modo in cui ci connettiamo con persone in tutto il mondo soprattutto dopo il periodo che abbiamo passato relativo alla pandemia. Proprio per questo nasce l'idea di LiveLink, un'app di videochiamate e messaggistica che offre un'esperienza di comunicazione ricca, coinvolgente ed efficiente.

Livelink sfrutta il protocollo WebRTC (Web Real-Time Communication) per fornire un sistema di videochiamate di alta qualità attraverso una pagina web appositamente creata sfruttando il fatto che questo protocollo è supportato dalla maggior parte dei moderni browser e consente agli utenti di stabilire connessioni dirette e in tempo reale, superando le tradizionali limitazioni della comunicazione a distanza.

Con LiveLink, non solo è possibile effettuare videochiamate ma offre anche la possibilità di sfruttare una chat istantanea presente all'interno della pagina web.

Una caratteristica distintiva di Livelink è la sua facilità d'uso, infatti l'app è progettata per essere intuitiva e accessibile a tutti garantendo che chiunque possa usufruire delle sue funzionalità senza difficoltà.

Per utilizzare l'app sarà sufficiente collegarsi ad un URL specifico per avviare una videochiamata e connettersi con persone in tutto il mondo, rompendo le barriere geografiche e creando esperienze di comunicazione personalizzate e autentiche.

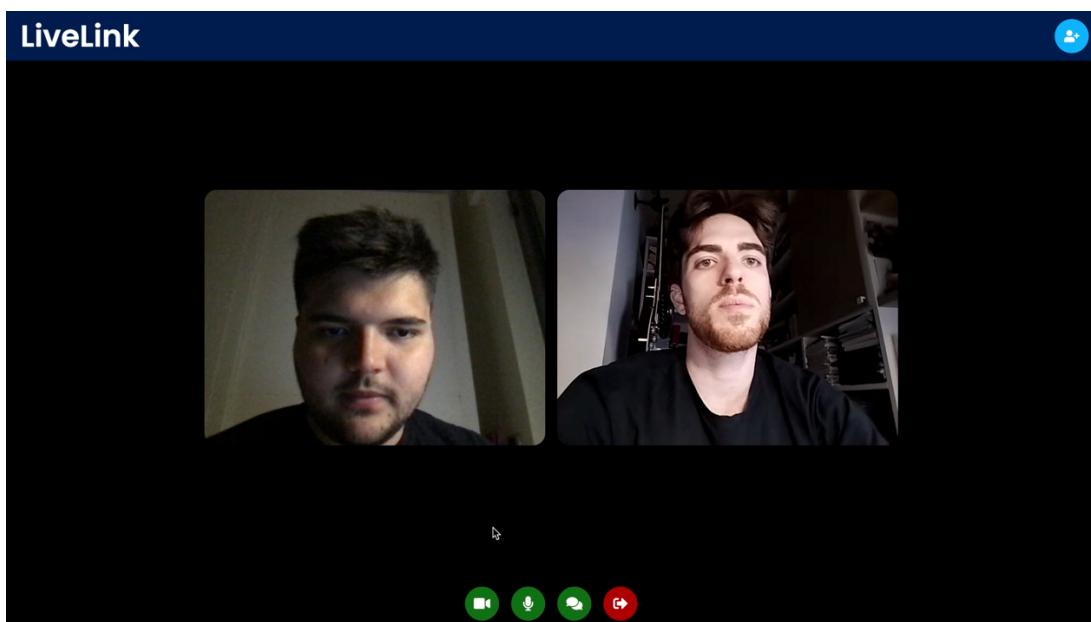
1.0 INTERFACCIA GRAFICA

Il primo approccio è stato quello di rendere il più semplice e intuitivo possibile l'interfaccia utente, inserendo quattro semplici pulsanti ciascuno con un suo compito specifico.

Dal punto di vista grafico ogni pulsante è dotato di uno stato hover e focus ai quali corrisponde un cambiamento di colore, infatti, tutta la paletta dei colori è stata studiata in modo accurato, inoltre, le variabili che contengono i colori si trovano nelle prime righe del file **style.css** così da permettere al gestore della piattaforma di cambiare un determinato colore in modo semplice e veloce.

```
3   :root {  
4     --color-primary-900: #001d4d;  
5     --color-primary-800: #01317e;  
6     --color-red-button: #ab0000;  
7     --color-red-button-hover: #rgb(211 3 3);  
8     --color-error: red;  
9     --color-button: #117012;  
10    --color-button-hover: #rgb(36 129 28);  
11    --primary-color: #2f80ec;  
12    --color-basic: #ffffff;  
13    --color-gray-600: #595959;  
14    --color-gray-200: #eeeeee;  
15    --color-gray-400: #9f9f9f;  
16    --color-primary-200: #00b4ff;  
17    --color-primary-200-hover: #4bc7fc;  
18    --color-dark: black;  
19    font-family: "Poppins", sans-serif;  
20 }
```

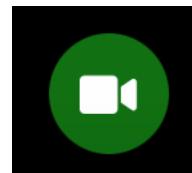
Main Section



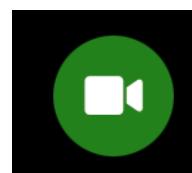
1.1 STOPVIDEO BUTTON

Lo stopVideo button che permette all'utente di attivare/disattivare la fotocamera in base alle sue esigenze, vediamo di seguito i possibili stati abilitati su di esso.

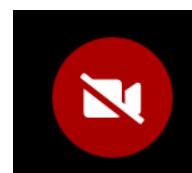
- Attivo



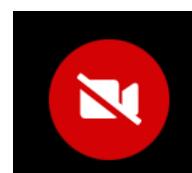
- Attivo: Hover



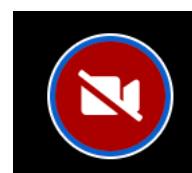
- Disattivo



- Disattivo:hover



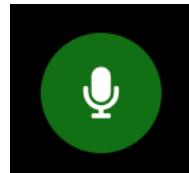
- Default/Disattivo: focus



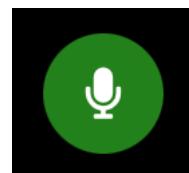
1.2 MUTE BUTTON

Il muteButton permette all'utente di attivare/disattivare la fotocamera in base alle sue esigenze, vediamo di seguito i possibili stati abilitati su di esso.

- Default



- Default: hover



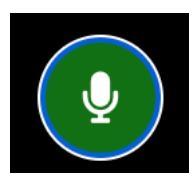
- Disattivo



- Disattivo: hover



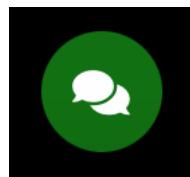
- Default /Disattivo: focus



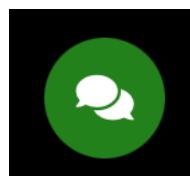
1.3 CHAT BUTTON

Questo button viene utilizzato per mostrare/nascondere la sezione relativa alla chat, vediamo di seguito i possibili stati abilitati su di esso.

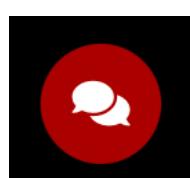
- Default



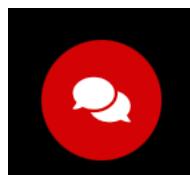
- Default: hover



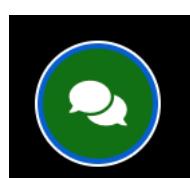
- Disattivo



- Disattivo: hover



- Default /Disattivo: focus



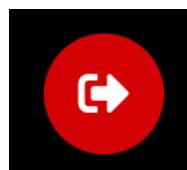
1.4 ENDCALL BUTTON

L'endCall-button quando viene premuto la connessione video e audio termina automaticamente e la griglia che contiene i video scompare, così la connessione tra i due utenti termina e si può chiudere la pagina web. Vediamo di seguito i possibili stati abilitati su di esso.

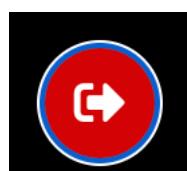
- Default



- Default: hover



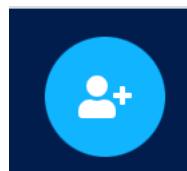
- Default: focus



1.5 INVITE BUTTON

L'invite-button viene utilizzato per invitare un altro utente all'interno della piattaforma così da poter iniziare la comunicazione, vediamo di seguito i possibili stati abilitati su di esso.

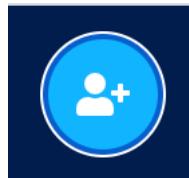
- Default



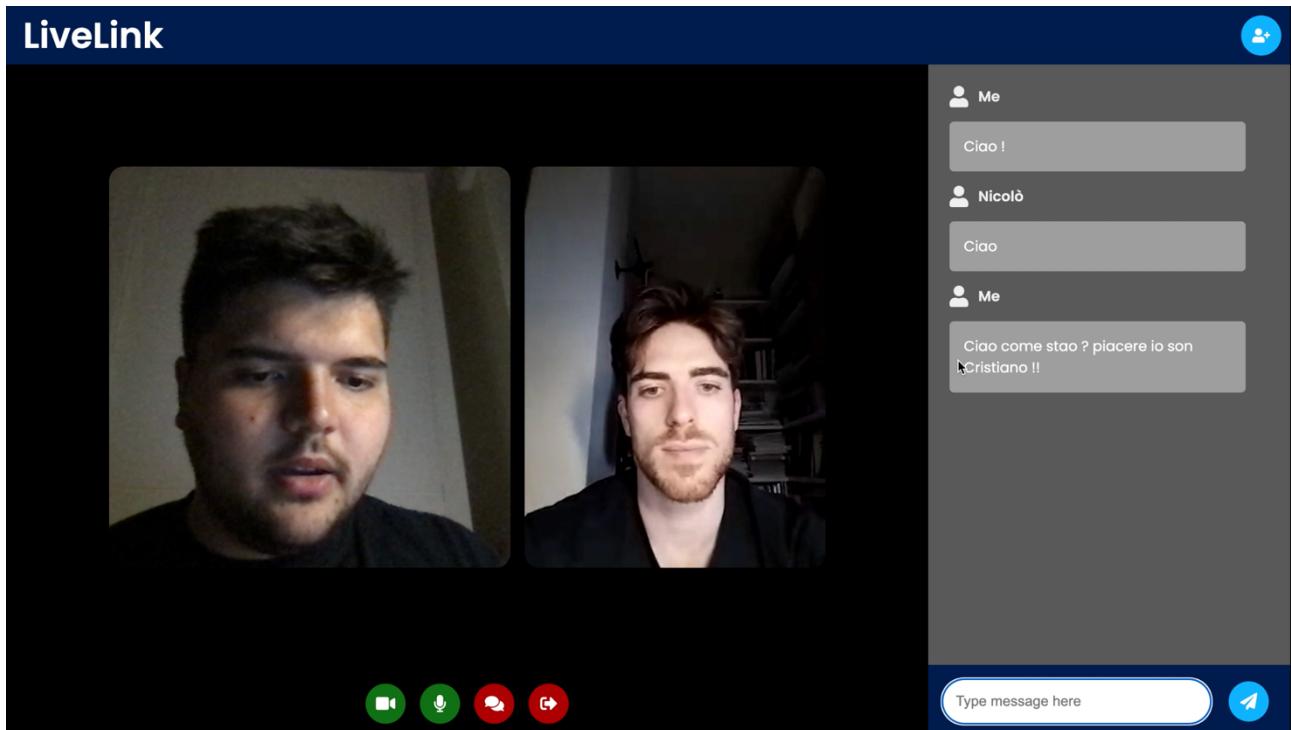
- Default: hover



- Default: focus



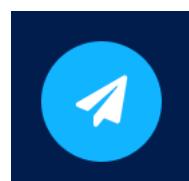
1.6 MESSAGE SECTION



1.7 SEND-BUTTON

Il send-button viene utilizzato per inviare un messaggio agli utenti presenti nella piattaforma, vediamo di seguito i possibili stati abilitati su di esso.

- Default



- Default: hover



1.8 INPUT FIELD

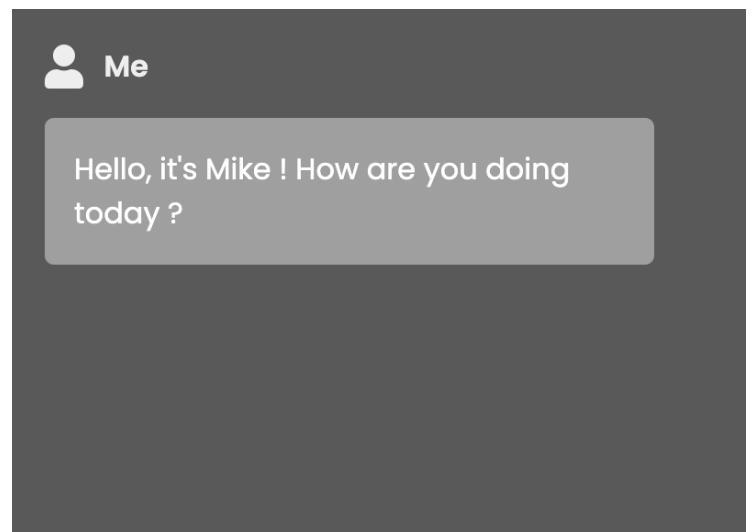
- Default



- Default: focus



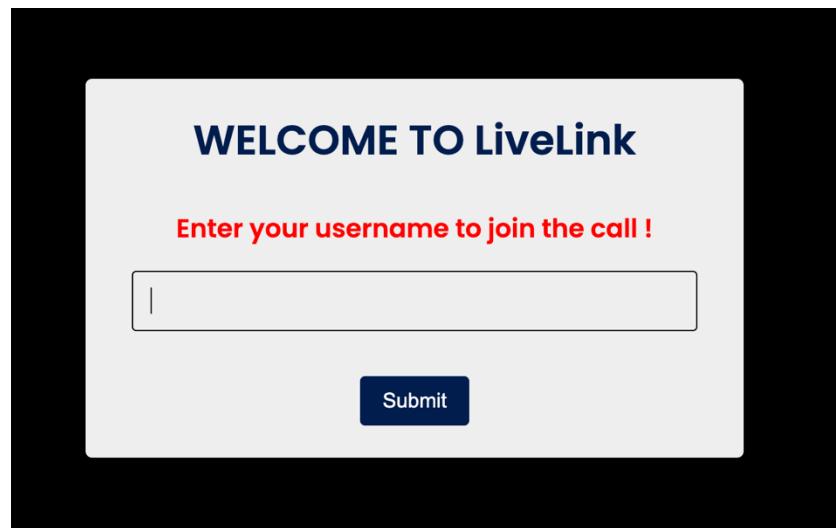
1.9 MESSAGE CONTAINER



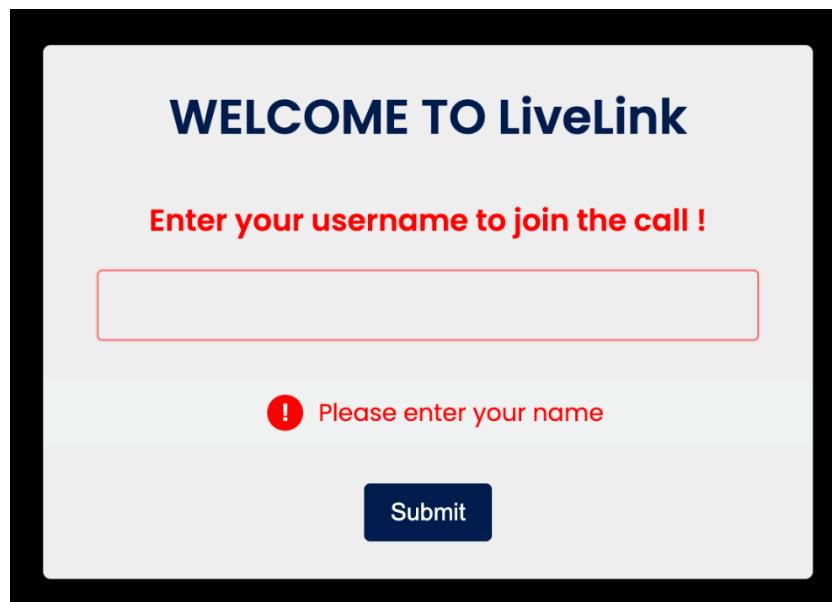
MODALI

1.10 WELCOME MODAL

La welcome-modal è il primo componente che viene visualizzato quando viene caricata la pagina e serve per inserire il proprio username così da essere certi che un utente non lasci questo campo vuoto evitando di essere identificato.

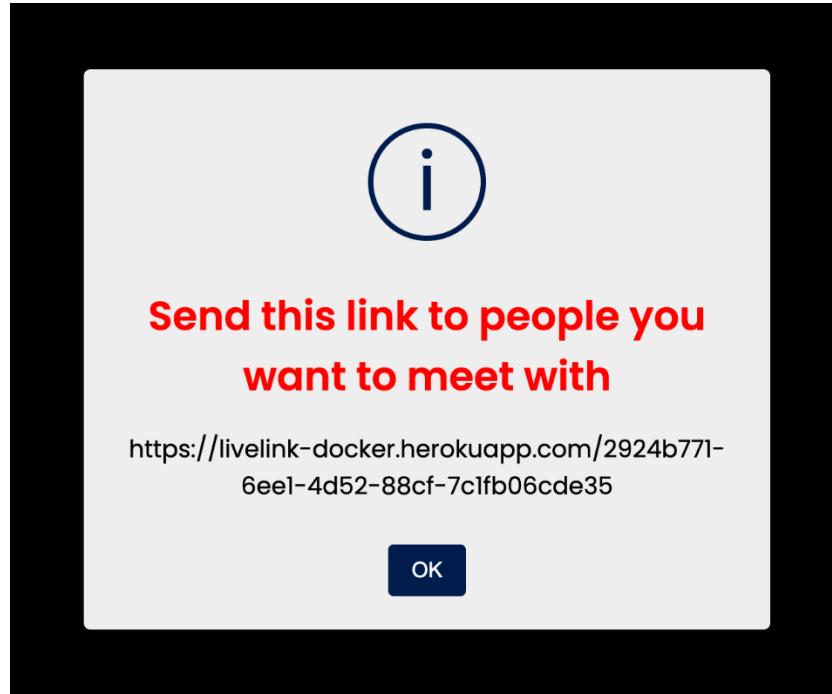


Nel caso in cui l'utente provasse a lasciare questo campo vuoto comparirebbe un errore come nella seguente immagine:



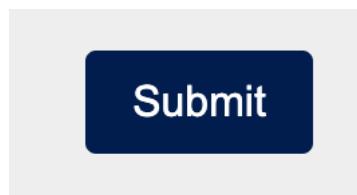
1.11 INVITE PEOPLE

L'invite-people modal viene visualizzata quando viene premuto l'invite-button e contiene il link da inviare all'utente con il quale si intende interagire.

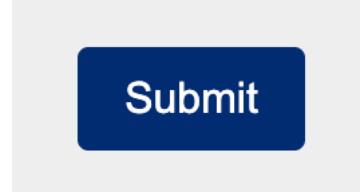


1.12 SUBMIT BUTTON

- Default



- Default: hover



1.13 OK BUTTON

- Default

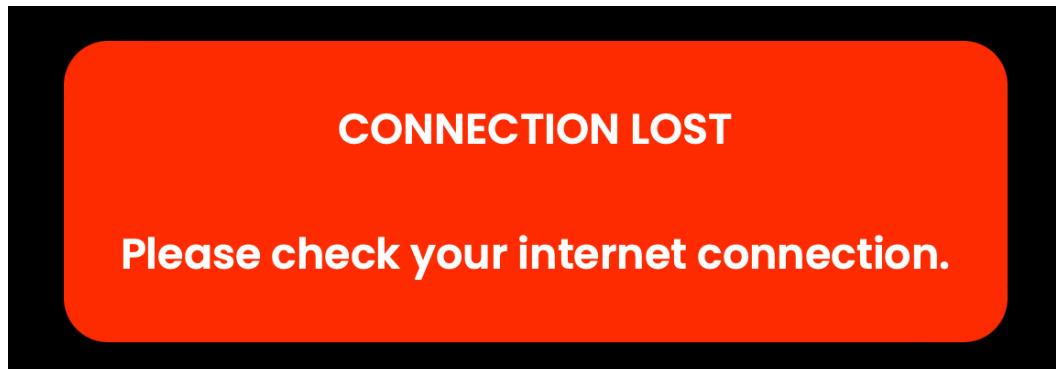


- Default: hover



1.14 CONNECTION ERROR

Un errore di connessione si ottiene quando l'utente che lo visualizza non è più connesso ad internet, quando appare l'errore il video automaticamente viene disattivato, per poter ripristinare la connessione sarà necessario riaggiornare la pagina web.



2 SVILUPPO

2.1 Server.js

```
cristiano, 3 weeks ago | 1 author (cristiano)
1 const express = require("express");
2 const app = express();
3 const server = require("http").Server(app);
4 const { v4: uuidv4 } = require("uuid");
5 app.set("view engine", "ejs");
6 const io = require("socket.io")(server, {
7   cors: {
8     origin: "*",
9   },
10 });
11 const { ExpressPeerServer } = require("peer");
12 const opinions = {
13   debug: true,
14};
```

Il codice contenuto all'interno di questo file permette di creare un'applicazione web utilizzando le funzionalità di Express.js il quale è un framework per applicazioni web che utilizzano Node.js, inoltre rappresenta un componente back-end fondamentale per le applicazioni stack MEAN, MERN e MEVN i quali permettono di creare un'applicazione completa (front end, back end, database) utilizzando interamente Javascript e JSON.

Il primo passo è stato quello di inizializzare un server HTTP utilizzando il modulo *http* di Node.js.

Il modulo *uuid* è stato utilizzato per generare un identificatore univoco (UUID) da associare ad ogni utente e ad ogni stanza che contiene gli utenti.

Il motore di visualizzazione usato dall'app è quello di *ejs* grazie al quale è possibile creare template HTML con codice JavaScript incorporato.

Infatti, si possono definire variabili, condizioni e funzioni JavaScript all'interno del template, che vengono elaborati dal motore EJS per generare il risultato finale, ovvero la pagina HTML completa con i dati dinamici inseriti.

Per garantire la comunicazione in tempo reale abbiamo utilizzato Socket.IO, cioè, una libreria JavaScript per applicazioni web in tempo reale che permette una comunicazione bidirezionale real-time tra i web client e i server.

È formata da due parti: una libreria lato client che gira sul browser e una libreria lato server per Node.js.

Dopo aver creato un'istanza di socket.io e averla assegnata ad una variabile chiamata **io** abbiamo utilizzato la funzione **require("socket.io")** che ha due parametri: server e un oggetto di opzioni.

- Server, che è stato creato in precedenza con il modulo **http**, rappresenta l'istanza del server Node.js con cui si desidera associare Socket.IO.
- L'oggetto di opzioni contiene una proprietà **cors** che consente di configurare le opzioni del Cross-Origin Resource Sharing (CORS) per Socket.IO.
In questo caso viene specificato che qualsiasi origine è consentita attraverso **origin: "*"** per consentire richieste provenienti da qualsiasi dominio.

Per la comunicazione peer to peer abbiamo utilizzato la libreria javascript peer.js che semplifica l'implementazione della comunicazione in un'applicazioni web, consentendo ai client di comunicare direttamente tra loro senza la necessità di un server centrale.

La libreria si basa sul protocollo WebRTC (Web Real-Time Communication), che utilizza una tecnologia web standard per la comunicazione in tempo reale tra browser.
Utilizzando WebRTC semplifica la gestione delle connessioni P2P, la negoziazione degli indirizzi IP e la trasmissione dei dati tra i client.

Con Peer.js abbiamo creato un'istanza di un oggetto **Peer** che rappresenta un client all'interno della comunicazione per stabilire una connessione P2P con altri client utilizzando un identificatore univoco creato dal modulo **uuid** visto in precedenza.

```
16  app.use("/peerjs", ExpressPeerServer(server, opinions));
17  app.use(express.static("public"));
18
19  app.get("/", (req, res) => {
20  |  res.redirect(`/${uuidv4()}`);
21  });
22
23  app.get("/:room", (req, res) => {
24  |  res.render("room", { roomId: req.params.room });
25  });
```

In questo caso ExpressPeerServer viene utilizzato come middleware dell'applicazione grazie all'implementazione di Express, dato che viene inclusa all'interno della libreria peer di Peer.js. È stato grazie a questo modulo che sono state gestite le connessioni peer-to-peer all'interno dell'app.

L'oggetto server viene passato a ExpressPeerServer per associare il server Peer.js al server http creato inizialmente.

L'oggetto opinions rappresenta le opzioni di configurazione per il server Peer.js.

In seguito, abbiamo impostato la cartella **public** come cartella statica, consentendo di servire file statici al suo interno come il foglio di stile **style.css** che contiene tutti gli stili dell'applicazione e il file **script.js** che invece, contiene tutta la logica dell'applicazione sottoforma di codice in formato javascript.

Quando viene effettuata una richiesta GET alla route "/" viene eseguita la funzione **app.get()**.

All'interno della funzione, viene chiamato il metodo **redirect()** sull'oggetto res (response) che rappresenta la risposta HTTP da inviare al client.

Il metodo **redirect()** viene utilizzato per reindirizzare il client a un'altra route specificata come argomento, nel nostro caso viene creato un percorso dinamico utilizzando la funzione **uuidv4()** vista in precedenza.

La funzione **uuidv4()** restituisce una stringa casuale come ad esempio /1194b42f-7f7d-4af4-96ca-00440c6b6688.

Quindi, la chiamata a **res.redirect()** reindirizza il client a /1194b42f-7f7d-4af4-96ca-00440c6b6688, ovvero una nuova route dinamica generata ogni volta che viene effettuata una richiesta a "/".

Il client riceverà quindi una risposta di reindirizzamento con uno stato HTTP 302 e verrà instradato alla nuova route generata.

L'ultima parte dell'immagine si riferisce a quando viene effettuata una richiesta GET a una route che corrisponde al pattern **/:room**, la funzione associata viene eseguita.

La funzione prende due parametri:

- req (request)
- res (response)

che rappresentano l'oggetto di richiesta e l'oggetto di risposta HTTP, rispettivamente.

All'interno della funzione, viene chiamato il metodo **render()** sull'oggetto res per renderizzare un template specifico.

Il primo argomento di **res.render()** è il nome del template da renderizzare (room).

Il secondo argomento di **res.render()** è un oggetto che contiene i dati che verranno passati al template. In questo caso, viene passato un oggetto con una proprietà **roomId** il cui

valore è ***req.params.room*** il quale rappresenta il valore del parametro *:room* estratto dall'URL della richiesta.

```
27  io.on("connection", (socket) => {
28    socket.on("join-room", (roomId, userId, userName) => {
29      socket.join(roomId);
30      setTimeout(() => {
31        socket.to(roomId).broadcast.emit("user-connected", userId);
32      }, 1000);
33      socket.on("message", (message) => {
34        io.to(roomId).emit("createMessage", message, userName);
35      });
36    });
37  });
38
39 server.listen(process.env.PORT || 3030);
40
```

Questa sezione, invece, si focalizza sulla parte di messaggistica in tempo reale dell'applicazione gestendo le connessioni dei socket e l'invio di messaggi.

Nella prima parte del codice l'istruzione ***io.on*** si attiva quando viene stabilita una nuova connessione tra il server e un client tramite un socket e di conseguenza vengono definiti i gestori per gli eventi specifici che si verificano durante la connessione.

- ***join-room*** il socket viene aggiunto alla stanza specificata dall'ID della stanza attraverso la variabile *roomId*.
- **setTimeout** Dopo un ritardo di 1 secondo viene inviato un evento "user-connected" agli altri partecipanti della stanza avvisando gli altri utenti che un nuovo utente si è aggiunto alla stanza.
- **socket.on** Il socket ascolta l'evento ***message*** che viene scatenato quando un utente invia un messaggio, quindi il server inoltra il messaggio a tutti gli utenti nella stessa stanza utilizzando la funzione ***io.to(roomId).emit("createMessage", message, userName)***.

Infine, l'istruzione ***server.listen(process.env.PORT || 3030)*** avvia il server e si mette in ascolto su una porta specifica.

2.2 Script.js

```
1 const socket = io("/");
2 const videoGrid = document.getElementById("video-grid");
3 const myVideo = document.createElement("video");
4 const showChat = document.querySelector("#showChat");
5 const backBtn = document.querySelector(".header__back");
6 myVideo.muted = true;
7
8 Swal.fire([
9   title:
10   |   "<div class='title-username-modal'><span>WELCOME TO LiveLink</span></div>",
11   input: "text",
12   inputAttributes: {
13     |   autocapitalize: "off",
14   },
15   showCancelButton: false,
16   confirmButtonText: "Submit",
17   allowOutsideClick: false,
18   preConfirm: (name) => {
19     if (!name) {
20       Swal.showValidationMessage("Please enter your name");
21     }
22     return name;
23   },
24 ]) .then((result) => {
25   if (result.isConfirmed) {
26     const user = result.value;
27
28     var peer = new Peer({
29       host: window.location.hostname,
30       port:
31         |   window.location.port ||
32         |   (window.location.protocol === "https:" ? 443 : 80),
33       path: "/peerjs",
34       config: {
35         iceServers: [
36           { urls: "stun:stun.l.google.com:19302" },
37           { urls: "stun:stun1.l.google.com:19302" },
38         ],
39       },
40       debug: 3,
41     });
42   }
43 });
44
45 
```

Nella prima parte dell'immagine il codice inserito è necessario per mostrare la modale welcome-modal richiamando la funzione ***fire()*** del modulo SweetAlert per creare una finestra di dialogo.

Infatti, grazie a questa funzione l'utente è costretto ad inserire un username senza il quale viene visualizzato un messaggio di errore che gli vieta di proseguire.

Nella seconda parte dell'immagine il codice inserito viene utilizzato per terminare la configurazione del canale di comunicazione tra il due communication peers creando un nuovo oggetto *Peer* utilizzando il costruttore *Peer* per gestire la connessione peer-to-peer tramite la libreria PeerJS.

In seguito viene impostato l'host del server Peer sullo stesso hostname del sito web in cui viene eseguito il codice e la porta del server Peer viene settata sulla stessa porta del sito web in cui viene eseguito il codice in base al tipo di protocollo scelto, se il protocollo è HTTPS sulla porta 443, invece se il protocollo è http sulla porta 80.

Infine, vengono configurati i server ICE (Interactive Connectivity Establishment) per la gestione delle connessioni WebRTC.

Gli oggetti "urls" specificano gli URL dei server STUN (Session Traversal Utilities for NAT) utilizzati per stabilire le connessioni peer-to-peer.

```
43  let myVideoStream;
44  navigator.mediaDevices
45    .getUserMedia({
46      audio: true,
47      video: true,
48    })
49    .then((stream) => {
50      myVideoStream = stream;
51      addVideoStream(myVideo, stream);
52      logVideoStreamInfo(stream);
53
54      peer.on("call", (call) => {
55        console.log("someone call me");
56        call.answer(stream);
57        const video = document.createElement("video");
58        call.on("stream", (userVideoStream) => {
59          addVideoStream(video, userVideoStream);
60          logVideoStreamInfo(userVideoStream);
61        });
62      });
63    });
64
```

Inizialmente viene dichiarata una variabile `myVideoStream` che verrà utilizzata per memorizzare lo stream video del dispositivo utilizzato dall'utente.

Viene utilizzato l'oggetto `navigator.mediaDevices.getUserMedia()` per ottenere l'accesso alla fotocamera e al microfono dell'utente. Questo metodo restituisce una promise che viene risolta con uno stream contenente l'audio e il video del dispositivo.

All'interno del blocco `then()` della promise lo stream appena ottenuto viene assegnato alla variabile `myVideoStream` e successivamente viene chiamata la funzione `addVideoStream()` per aggiungere il proprio video allo schermo. Inoltre, viene chiamata la funzione `logVideoStreamInfo()` per registrare informazioni sullo stream video.

Nella seconda parte dell'immagine abbiamo definito un gestore per l'evento ***call*** del peer, così quando viene ricevuta una chiamata da un altro utente il codice esegue una risposta alla chiamata e crea un elemento video per visualizzare lo stream video dell'altro utente. Questo video viene aggiunto allo schermo chiamando la funzione ***addVideoStream()***.

Infine, viene chiamata la funzione ***logVideoStreamInfo()*** per registrare informazioni sullo stream video dell'altro utente.

```
64      socket.on("user-connected", (userId) => {
65        connectToNewUser(userId, stream);
66      );
67    });
68
69    const connectToNewUser = (userId, stream) => {
70      console.log("I call someone" + userId); cristiano, 2
71      const call = peer.call(userId, stream);
72      const video = document.createElement("video");
73      call.on("stream", (userVideoStream) => {
74        addVideoStream(video, userVideoStream);
75        logVideoStreamInfo(userVideoStream);
76      );
77    };
78
79    peer.on("open", (id) => {
80      console.log("my id is" + id);
81      socket.emit("join-room", ROOM_ID, id, user);
82    );
83
84
```

In questa sezione abbiamo definito un gestore per l'evento ***user-connected*** del socket, quando viene ricevuta una notifica di connessione di un nuovo utente, il codice chiama la funzione ***connectToNewUser()*** per stabilire una chiamata verso il nuovo utente, passando il proprio stream video.

```
84    const addVideoStream = (video, stream) => {
85      video.srcObject = stream;
86      video.addEventListener("loadedmetadata", () => {
87        video.play();
88        videoGrid.append(video); cristiano, 2 weeks
89      );
90    };
91
```

Poi è stato necessario creare un elemento video per visualizzare lo stream video dell'utente remoto il quale viene aggiunto allo schermo chiamando la funzione ***addVideoStream()***.

Quando il proprio peer va in stato di *open* e riceve un ID, il codice stampa l'ID del proprio peer sulla console e invia un evento *join-room* al server socket, passando l'ID della stanza, l'ID del proprio peer e il nome utente.

```
-- 92 let text = document.querySelector("#chat_message"); cristiano,
93 let send = document.getElementById("send");
94 let messages = document.querySelector(".messages");
95
96 send.addEventListener("click", (e) => {
97   if (text.value.length !== 0) {
98     socket.emit("message", text.value);
99     text.value = "";
100   }
101 });
102
103 text.addEventListener("keydown", (e) => {
104   if (e.key === "Enter" && text.value.length !== 0) {
105     socket.emit("message", text.value);
106     text.value = "";
107   }
108 });
109
```

Successivamente viene aggiunto un gestore per l'evento *click* sul pulsante di send così quando l'utente lo preme viene eseguito il blocco di codice nella funzione di callback.

All'interno della funzione di callback, viene verificato se il valore dell'input di testo *text.value* non è vuoto attraverso *text.value.length !== 0* e se l'input non è vuoto viene emesso un evento *message* al server socket utilizzando ***socket.emit()*** passando come argomento il valore dell'input di testo (*text.value*).

In seguito, il valore dell'input viene azzerato *text.value = ""* per cancellare il testo inserito dopo l'invio del messaggio così da poter inviare un nuovo messaggio partendo dal campo vuoto.

Infine, è stato aggiunto un gestore per l'evento *keydown* sull'input di testo che si verifica quando l'utente preme un tasto sulla tastiera.

All'interno della funzione di callback per l'evento *keydown*, viene verificato se il tasto premuto è *Enter* (*e.key === "Enter"*) viene quindi emesso un evento *message* al server socket utilizzando ***socket.emit()***, passando come argomento il valore dell'input di testo.

```

110  const inviteButton = document.querySelector("#inviteButton");
111  const muteButton = document.querySelector("#muteButton");
112  const stopVideo = document.querySelector("#stopVideo");
113  muteButton.addEventListener("click", () => {
114      const enabled = myVideoStream.getAudioTracks()[0].enabled;
115      if (enabled) {
116          myVideoStream.getAudioTracks()[0].enabled = false;
117          html = `<i class="fas fa-microphone-slash"></i>`;
118          muteButton.classList.toggle("clicked");
119          muteButton.innerHTML = html;
120      } else { cristiano, 2 weeks ago • added logic for message username
121          myVideoStream.getAudioTracks()[0].enabled = true;
122          html = `<i class="fas fa-microphone"></i>`;
123          muteButton.classList.toggle("clicked");
124          muteButton.innerHTML = html;
125      }
126  });
127

```

In questa sezione viene aggiunto un gestore per l'evento "click" sul pulsante di mute *muteButton*, quando l'utente clicca il pulsante, viene eseguito il blocco di codice nella funzione di callback all'interno della quale viene controllato lo stato attuale dell'audio del proprio stream video.

Attraverso il metodo ***getAudioTracks()*** su *myVideoStream* si ottiene l'array dei flussi audio e viene preso il primo elemento dell'array, viene quindi controllato se l'audio è abilitato attraverso il flag `enabled = true` o disabilitato attraverso il flag `enabled = false`.

- Se l'audio è abilitato, viene disabilitato impostando
myVideoStream.getAudioTracks()[0].enabled = false.
Viene aggiornato l'HTML del pulsante *muteButton* per mostrare l'icona di microfono disattivato e viene aggiunta o rimossa la classe "clicked" al pulsante utilizzando ***muteButton.classList.toggle("clicked")***.
- Se l'audio è disabilitato, viene abilitato impostando
myVideoStream.getAudioTracks()[0].enabled = true. Viene aggiornato l'HTML del pulsante *muteButton* per mostrare l'icona di microfono attivato e viene aggiunta o rimossa la classe "clicked" al pulsante utilizzando ***muteButton.classList.toggle("clicked")***.

```

128  stopVideo.addEventListener("click", () => {
129      const enabled = myVideoStream.getVideoTracks()[0].enabled;
130      if (enabled) {
131          myVideoStream.getVideoTracks()[0].enabled = false;
132          html = `<i class="fas fa-video-slash"></i>`;
133          stopVideo.classList.toggle("clicked");
134          stopVideo.innerHTML = html;
135      } else {
136          myVideoStream.getVideoTracks()[0].enabled = true;
137          html = `<i class="fas fa-video"></i>`;
138          stopVideo.classList.toggle("clicked");
139          stopVideo.innerHTML = html;
140      }
141  });

```

La funzione in questione ha lo scopo di gestire l'evento di clic sul button stopVideo.

Inizialmente, viene ottenuto lo stato corrente del video tramite la chiamata alla funzione `getVideoTracks()` su `myVideoStream` e l'accesso al primo elemento dell'array restituito. Questo stato viene memorizzato nella variabile `enabled` che si comporta come un flag, che può assumere i valori `true` o `false`, a seconda che il video sia abilitato o disabilitato.

Successivamente viene introdotto un ciclo `if` attraverso il quale viene verificato lo stato del flag e se il video è attualmente abilitato, cioè il valore di `enabled` è `true`, il blocco di codice all'interno delle parentesi graffe viene eseguito.

Altrimenti, se il video è disabilitato, cioè il valore di `enabled` è `false`, il blocco di codice all'interno delle parentesi graffe dell'`else` viene eseguito.

Nel blocco di codice che gestisce il caso in cui il video sia disabilitato, il video viene abilitato impostando la proprietà `enabled` del primo track video ottenuto da `myVideoStream` su `true`. Ciò significa che il video verrà riattivato. Successivamente, viene creata una stringa HTML che rappresenta un'icona di un video. Questa stringa HTML viene assegnata alla variabile `html`. Allo stesso modo, viene applicata la classe CSS "clicked" all'elemento HTML del pulsante "stopVideo". Infine, il contenuto HTML del pulsante "stopVideo" viene sostituito con il valore di `html`, che rappresenta l'icona del video.

```

143     inviteButton.addEventListener("click", (e) => {
144         Swal.fire({
145             title: "Send this link to people you want to meet with",
146             text: window.location.href,
147             icon: "info",
148             confirmButtonText: "OK",
149             position: "center",
150             allowOutsideClick: false,
151         });
152     });

```

In questa sezione abbiamo aggiunto un listener per il click sul pulsante inviteButton così quando l'utente clicca il button, viene avviata l'esecuzione della funzione.

All'interno della funzione, viene utilizzata la libreria SweetAlert (Swal) per creare una modale attraverso la funzione ***Swal.fire***.

```

154     socket.on("createMessage", (message, userName) => {
155         messages.innerHTML =
156             messages.innerHTML +
157             `<div class="message">
158                 <b><i class="fa fa-user" aria-hidden="true"></i><span> ${
159                     userName === user ? "me" : userName
160                 }</span> </b>
161                 <span>${message}</span>
162             </div>`;
163
164
165     });

```

Questa parte del codice si occupa di visualizzare il messaggio nella pagina, la variabile *messages* rappresenta l'elemento HTML in cui vengono mostrati i messaggi. Inizialmente, si accede al contenuto HTML corrente di *messages* utilizzando *messages.innerHTML*.

```

167 // Hide-show chat function
168
169 var toggleButton = document.getElementById("toggleButton");
170 const endCallButton = document.getElementById("endCallButton");
171
172 var sidebar = document.getElementById("sidebar");
173
174 toggleButton.addEventListener("click", function () {
175   sidebar.classList.toggle("open");
176 });
177
178 // Change color to chat button when clicked
179
180 var toggleButton = document.getElementById("toggleButton");
181
182 toggleButton.addEventListener("click", function () {
183   toggleButton.classList.toggle("clicked");
184 });

```

In questa parte del codice viene aggiunto un gestore per l'evento click sul pulsante di toggle chiamato toggleButton. Quando l'utente clicca sul pulsante, viene eseguito il blocco di codice nella funzione di callback, all'interno della quale, è stato utilizzato il metodo ***classList.toggle()*** per aggiungere o rimuovere la classe open dall'elemento sidebar. Questo comporta il cambiamento dello stato di visualizzazione della barra laterale, nascondendola se la classe è presente e mostrandola se la classe è assente.

La seconda parte del codice riguarda il cambio di colore del pulsante di chat quando viene cliccato, all'interno della funzione di callback viene utilizzato il metodo ***classList.toggle()*** per aggiungere o rimuovere la classe clicked dall'elemento toggleButton, questo comporta il cambio di colore del pulsante quando viene cliccato, grazie agli stili associati alla classe clicked.

```
186 endCallButton.addEventListener("click", () => {
187     // stop the connection
188     peer.destroy();      cristiano, 2 weeks ago • added logic to c
189
190     // Remove all the videos from videoGrid
191     const videos = document.querySelectorAll("#video-grid video");
192     videos.forEach(video => video.remove());
193
194     // Hide videoGrid section
195     videoGrid.style.display = "none";
196 })
```

Questa parte di codice gestisce l'evento di clic sul pulsante "endCallButton" per terminare la chiamata e ripulire l'interfaccia video.

All'interno della funzione di callback inizialmente viene chiamato il metodo **destroy()** sull'oggetto *peer* per interrompere la connessione peer-to-peer, poi vengono selezionati tutti gli elementi video presenti nella sezione con l'ID *video-grid* utilizzando **document.querySelectorAll("#video-grid video")**. Questi elementi rappresentano i video degli utenti partecipanti alla chiamata.

Viene iterato su ciascun elemento video utilizzando il metodo **forEach()**, all'interno del quale ogni elemento video viene rimosso dalla visualizzazione chiamando il metodo **remove()** su di esso.

Infine, viene impostata la proprietà **display** della sezione con l'ID "video-grid" su "none" per nascondere l'intera sezione dei video.

```

199   window.addEventListener("offline", function() {
200     const errorMessageDiv = document.getElementById("error-container");
201     errorMessageDiv.innerHTML = "<div class='message-error'> CONNECTION LOST </div><br><div class='message-error'> CONNECTION LOST </div>";
202     errorMessageDiv.style.display = "block"; // show the element
203
204     const videoGridDiv = document.getElementById("video-grid");
205     videoGridDiv.style.display = "none"; // hide div "video-grid"
206   });
207
208   window.addEventListener("online", function() { [Cristiano Marzano, 9 minutes ago • changed position]
209     const errorMessageDiv = document.getElementById("error-container");
210     errorMessageDiv.style.display = "none"; // hide this element
211
212     const videoGridDiv = document.getElementById("video-grid");
213     videoGridDiv.style.display = "block"; // show again div "video-grid"
214   });
215
216 }
217 });

```

Questa sezione gestisce gli eventi "offline" e "online" della finestra del browser per gestire la connessione Internet dell'utente e mostrare un messaggio di errore appropriato utilizzando la funzione **window.addEventListener(*offline*)**, questo evento si verifica quando la connessione Internet dell'utente viene persa.

All'interno della funzione di callback viene modificato il contenuto HTML dell'elemento *errorMessageDiv* utilizzando la proprietà *innerHTML*.

Abbiamo impostato un messaggio di errore con una classe CSS ***message-error*** per indicare la perdita di connessione Internet impostando la proprietà *display* dell'elemento ***errorMessageDiv*** su ***block*** per mostrare l'elemento nella pagina.

In questo caso viene selezionato l'elemento HTML con l'ID ***video-grid*** per essere assegnato alla variabile ***videoGridDiv*** la cui proprietà *display* diventa ***none*** per nascondere l'intero contenitore dei video nella pagina.

In seguito abbiamo aggiunto un gestore per l'evento *online* alla finestra del browser utilizzando **window.addEventListener(*online*)** che si verifica quando la connessione Internet dell'utente viene ripristinata.

All'interno della funzione di callback, vengono eseguite le seguenti operazioni:

- Viene selezionato l'elemento HTML con l'ID ***error-container*** utilizzando e viene assegnato alla variabile ***errorMessageDiv***.
- Viene impostata la proprietà *display* dell'elemento ***errorMessageDiv*** su ***none*** per nascondere il messaggio di errore.

Infine, viene impostata la proprietà *display* dell'elemento ***videoGridDiv*** su ***block*** per mostrare nuovamente l'intero contenitore dei video nella pagina.

3 DOCKER

Docker è una piattaforma open-source che permette di creare, distribuire e gestire applicazioni in modo isolato utilizzando i container, i quali consentono di impacchettare un'applicazione e tutte le sue dipendenze in un ambiente isolato che può essere eseguito su qualsiasi sistema operativo che supporta Docker.

Questo rende l'esecuzione delle applicazioni più affidabile e prevedibile, in quanto ogni container ha il proprio ambiente isolato e indipendente, senza interferenze con altre applicazioni o risorse del sistema.

Struttura di Docker:

- Docker Engine

Docker Engine è il cuore di Docker, responsabile dell'esecuzione dei container. Si compone di due parti fondamentali:

- Demone Docker (Docker Daemon): gestisce i container, le immagini, le reti e i volumi
- il client Docker (Docker client): permette agli utenti di interagire con il demone attraverso l'interfaccia della riga di comando o l'API.

- Docker Image

Un'immagine Docker rappresenta un pacchetto leggero e autonomo che contiene tutto il necessario per eseguire un'applicazione.

Le immagini sono create utilizzando un Dockerfile, cioè un file di configurazione che definisce le istruzioni per la costruzione dell'immagine.

Queste istruzioni includono il sistema operativo di base, le dipendenze, i comandi di installazione e le configurazioni necessarie.

- Docker Container

Un container Docker è un'istanza in esecuzione di un'immagine Docker.

Ogni container è isolato dagli altri e dal sistema ospite ma può comunicare con essi tramite interfacce specifiche, inoltre possono essere avviati, arrestati, messi in pausa o cancellati senza influire sugli altri container o sull'host di sistema.

I container sono leggeri, veloci da avviare e possiedono un ambiente isolato che garantisce la portabilità dell'applicazione.

- Docker Hub

Docker Hub è un registro pubblico di immagini Docker, fornito da Docker. Funge da repository centrale in cui gli sviluppatori possono condividere e distribuire le proprie immagini Docker.

3.1 CREAZIONE IMMAGINE

Il primo passaggio è stato quello di creare l'immagine del progetto attraverso il comando seguente:

- docker build -t livelink:v1.0 .

:v1.0 è opzionale e rappresenta il tag ma è preferibile utilizzarlo così da distinguere le versioni.

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    GITLENS

● cristiano@MacBook-Air-di-Cristiano LiveLink % docker build -t livelink:v1.0 .
[+] Building 2.3s (13/13) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 540B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:14
=> [auth] library/node:pull token for registry-1.docker.io
=> [1/7] FROM docker.io/library/node:14sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> [internal] load build context
=> => transferring context: 18.54kB
=> CACHED [2/7] WORKDIR /app
=> CACHED [3/7] COPY package*.json .
=> CACHED [4/7] RUN npm install
=> CACHED [5/7] COPY ./views ./views
=> CACHED [6/7] COPY ./public ./public
=> [7/7] COPY server.js .
=> exporting to image
=> => exporting layers
=> => writing image sha256:83d9e71647c03e04e232f3b1491c12b2aff5cd095f979b1b64860baab9c5934d
=> => naming to docker.io/library/livelink:v1.0
○ cristiano@MacBook-Air-di-Cristiano LiveLink %

```

Una volta fatto ciò è bastato lanciare il comando:

- Docker run livelink:v1.0

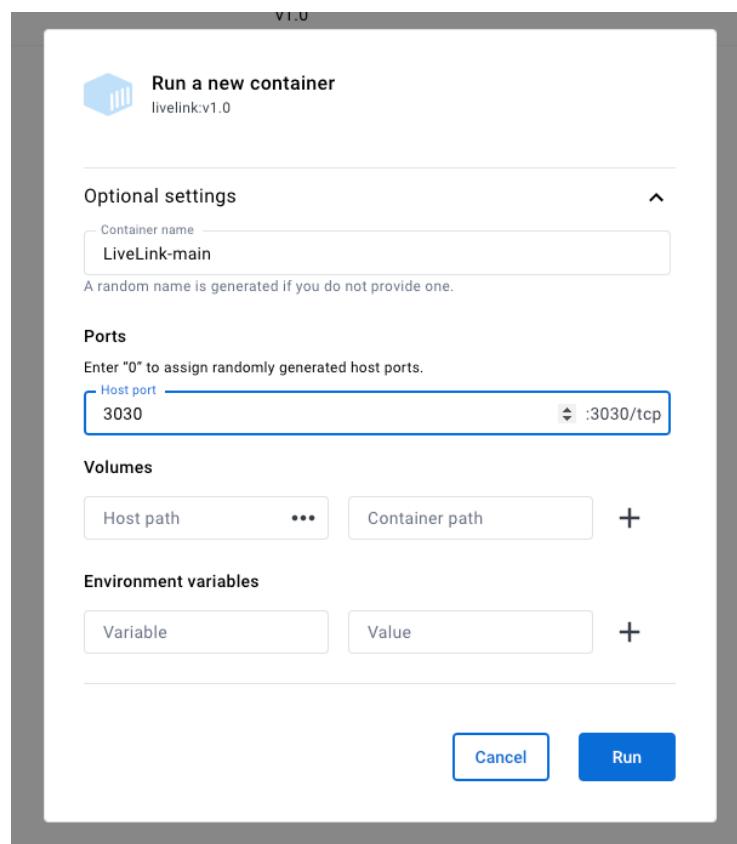
per far partire il container basato sull'immagine Docker che avevamo creato un attimo prima.

```

○ cristiano@MacBook-Air-di-Cristiano LiveLink % docker run livelink:v1.0
> zoom@1.0.0 start /app
> node server.js

```

Lo stesso risultato è stato ottenuto runnando il container direttamente da Docker Desktop nel modo seguente:



Un procedimento simile è applicato ai branch che si vogliono containerizzare, un esempio è mostrato nella figura sottostante dove sono state create prima le immagini dei relativi branch e poi i rispettivi container con i relativi tag.

Images [Give feedback](#)

Local Hub Artifactory [EARLY ACCESS](#)

1.98 GB / 3.18 GB in use 16 images Last refresh: 0 seconds ago [↻](#)

<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	cristiano0121/livelink 943c182b2c2d	v1.0	Unused	3 days ago	931.1 MB	▶ ⋮ trash
<input type="checkbox"/>	livelink 943c182b2c2d	v1.0	Unused	3 days ago	931.1 MB	▶ ⋮ trash
<input type="checkbox"/>	registry.heroku.com/livelink-docker/web 943c182b2c2d	latest	Unused	3 days ago	931.1 MB	▶ ⋮ trash
<input type="checkbox"/>	livelink 63b19313b5f6	logs	In use	17 days ago	923.8 MB	▶ ⋮ trash
<input type="checkbox"/>	livelink 8c88444417c5	interface	In use	23 days ago	923.32 MB	▶ ⋮ trash
<input type="checkbox"/>	docker/desktop-vpnkit-controller 556098075b3d	dc331cb22850be0cdd97c84a9cfecaf44a1afb	Unused	1 month ago	36.22 MB	▶ ⋮ trash
<input type="checkbox"/>	hubproxy.docker.internal:5555/docker/deskt 23c4f114c8bbe	kubernetes-v1.25.9-cni-v1.1.1-critools-v1.25.0	Unused	2 months ago	402.21 MB	▶ ⋮ trash
<input type="checkbox"/>	registry.k8s.io/kube-apiserver dc245db8c2fa	v1.25.9	Unused	2 months ago	127.89 MB	▶ ⋮ trash
<input type="checkbox"/>	registry.k8s.io/kube-controller-manager 3ea2571fcc83	v1.25.9	Unused	2 months ago	117.23 MB	▶ ⋮ trash

Showing 16 items

RAM 1.41 GB Disk 52.20 GB avail. of 62.67 GB  v4.20.1

Containers [Give feedback](#)

Container CPU usage [ⓘ](#) Container memory usage [ⓘ](#) Show charts [▼](#)

0.00% / 400% (4 cores allocated) 0B / 7.5GB

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	CPU (%)	Actions
<input type="checkbox"/>	 minikube 930f696ca4e1	gcr.io/k8s-minikube/kicbase:v0.0	Exited		8 days ago	0%	▶ ⋮ trash
<input type="checkbox"/>	 livelink-logs 884dd5833ec0	livelink:logs	Exited	3030:3030	17 days ago	0%	▶ ⋮ trash
<input type="checkbox"/>	 livelink-interface 173dd4e04878	livelink:interface	Exited	3030:3030	22 days ago	0%	▶ ⋮ trash
<input type="checkbox"/>	 main d19ba7264b27	livelink:v1.0	Exited	3030:3030	14 seconds ago	0%	▶ ⋮ trash

3.2 CREAZIONE REPOSITORY (DOCKERHUB)

La creazione di un repo su DockerHub si è resa necessaria per poter contenere l'immagine del nostro progetto.

<https://hub.docker.com/repository/docker/cristiano0121/livelink/general>

The screenshot shows the Docker Hub interface for creating a new repository. At the top, there's a search bar and navigation links for Explore, Repositories, Organizations, Help, and Upgrade. The user is signed in as 'cristiano0121'. Below the header, it says 'Using 0 of 1 private repositories.' with a 'Get more' link. The main form is titled 'Create repository'. It has a 'Namespace' dropdown set to 'cristiano0121', a 'Repository Name *' input field, and a 'Description' text area. To the right, a 'Pro tip' box contains the text: 'You can push a new image to this repository using the CLI' followed by two command examples: 'docker tag local-image:tagname new-repo:tagname' and 'docker push new-repo:tagname'. Below the form, there's a 'Visibility' section with 'Public' (selected) and 'Private' options. The 'Private' option is described as 'Only visible to you'. At the bottom right of the form are 'Cancel' and 'Create' buttons.

The screenshot shows the Docker Hub repository details for 'cristiano0121/livelink'. The top navigation bar includes 'Explore', 'Repositories', 'Organizations', 'Help', 'Upgrade', and the user profile 'cristiano0121'. The repository path 'cristiano0121/livelink' is shown in the breadcrumb. It says 'Using 0 of 1 private repositories.' with a 'Get more' link. Below the header, there are tabs for General, Tags, Builds, Collaborators, Webhooks, and Settings. The 'General' tab is selected. On the left, there's a 'Description' section with a note that the repository does not have a description. On the right, there's a 'Docker commands' section with the command 'docker push cristiano0121/livelink:tagname'. Below these are 'Tags' and 'Automated Builds' sections. The 'Tags' section lists one tag: 'v1.0' (Image type, pushed a minute ago). The 'Automated Builds' section notes that manually pushing images to Hub? requires connecting to GitHub or Bitbucket. There's also a note about automated builds being available with Pro, Team and Business subscriptions. At the bottom, there's a 'Repository overview' section with a 'Add overview' button.

3.3 PUBBLICAZIONE IMMAGINE SUL REPO

Una volta creato il repo si carica l'immagine al suo interno, ciò può essere fatto utilizzando i seguenti due comandi:

- docker tag livelink:v1.0 cristiano0121/livelink:v1.0

il tag viene utilizzato per associare l'immagine al nome corretto del repository DockerHub e al tag desiderato.

- docker push cristiano0121/livelink:v1.0

per pushare direttamente l'immagine nel repo.

```
● cristiano@MacBook-Air-di-Cristiano LiveLink % docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
livelink        logs     83d9e71647c0   18 minutes ago  923MB
livelink        v1.0     83d9e71647c0   18 minutes ago  923MB
livelink        interface 8c88444417c5   24 hours ago   923MB
● cristiano@MacBook-Air-di-Cristiano LiveLink % docker tag livelink:v1.0 cristiano0121/livelink:v1.0
● cristiano@MacBook-Air-di-Cristiano LiveLink % docker push cristiano0121/livelink:v1.0
The push refers to repository [docker.io/cristiano0121/livelink]
1cc75c17977a: Pushed
b51827080995: Pushed
b663a9462c7c: Pushed
4e149338e0ff: Mounted from cristiano0121/livelink-app
fa381da7e54e: Mounted from cristiano0121/livelink-app
897fd94fa93b: Mounted from cristiano0121/livelink-app
0d5f5a015e5d: Mounted from cristiano0121/livelink-app
3c777d951de2: Mounted from cristiano0121/livelink-app
f8a91dd5fc84: Mounted from cristiano0121/livelink-app
cb81227abde5: Mounted from cristiano0121/livelink-app
e01a454893a9: Mounted from cristiano0121/livelink-app
c45660adde37: Mounted from cristiano0121/livelink-app
fe0fb3ab4a0f: Mounted from cristiano0121/livelink-app
f1186e5061f2: Mounted from cristiano0121/livelink-app
b2dba7477754: Mounted from cristiano0121/livelink-app
v1.0: digest: sha256:1c785106e94bd31454936109515cf624a0fe615c499d1129073ae54e9754d8ad size: 3463
○ cristiano@MacBook-Air-di-Cristiano LiveLink %
```

3.4 COLLEGARE HEROKU A DOCKER

Come prima è necessario autenticarsi con le proprie credenziali su heroku utilizzando il seguente comando

- heroku login

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    GITLENS
● cristiano@MacBook-Air-di-Cristiano LiveLink % heroku login
  > Warning: heroku update available from 7.68.2 to 8.1.7.
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser?04ed79cc-8896-4730-b0a9-79a3191ee65d?requestor=SFMyNTY.g2g0b0AAAAw5My40Ni45N14xNjduBgCg75mHiAFjAAFRgA.AHn2znfPv0tlnsKT1wE30XwB4Nlt0UwE2Ghktqc0KQ
Logged in as cristiano@studenti.poliba.it
Logged in as cristiano@MacBook-Air-di-Cristiano LiveLink %
```

Il passaggio successivo è stato quello di collegarsi all'app su Heroku lanciando il comando seguente:

- heroku git:remote -a livelink-docker

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    GITLENS
● cristiano@MacBook-Air-di-Cristiano LiveLink % heroku git:remote -a livelink-docker
  set git remote heroku to https://git.heroku.com/livelink-docker.git
○ cristiano@MacBook-Air-di-Cristiano LiveLink %

```

Per pushare l'immagine docker all'interno del repository Heroku è necessario eseguire i seguenti comandi in sequenza :

- docker tag cristiano0121/livelink:v1.0 registry.heroku.com/livelink-docker/web
- docker images (facoltativo)
- docker push registry.heroku.com/livelink-docker/web
- heroku container:release web --app livelink-docker
- heroku open --app livelink-docker

```

● cristiano@MacBook-Air-di-Cristiano LiveLink % docker tag cristiano0121/livelink:v1.0 registry.heroku.com/livelink-docker/web
● cristiano@MacBook-Air-di-Cristiano LiveLink % docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
cristiano0121/livelink    logs     83d9e71647c0  7 hours ago  923MB
cristiano0121/livelink    v1.0     83d9e71647c0  7 hours ago  923MB
livelink              logs     83d9e71647c0  7 hours ago  923MB
livelink              v1.0     83d9e71647c0  7 hours ago  923MB
registry.heroku.com/livelink-docker/web  latest   83d9e71647c0  7 hours ago  923MB
registry.heroku.com/livelink/web       latest   83d9e71647c0  7 hours ago  923MB
livelink              interface 8c8844417c5   31 hours ago  923MB
● cristiano@MacBook-Air-di-Cristiano LiveLink % docker push registry.heroku.com/livelink-docker/web
Using default tag: latest
The push refers to repository [registry.heroku.com/livelink-docker/web]
1cc75c17977a: Mounted from livelink/web
b51827080995: Mounted from livelink/web
b663a9462c7c: Mounted from livelink/web
4e149338e0ff: Mounted from livelink/web
fa381da7e54e: Mounted from livelink/web
897fd94fa93b: Mounted from livelink/web
0d5f5a015e5d: Mounted from livelink/web
3c777d951de2: Mounted from livelink/web
f8a91dd5fc84: Mounted from livelink/web
cb81227abde5: Mounted from livelink/web
e01a454893a9: Mounted from livelink/web
c45660adde37: Mounted from livelink/web
fe0fb3bab4a0f: Mounted from livelink/web
f1186e5061f2: Mounted from livelink/web
b2db47477754: Mounted from livelink/web
latest: digest: sha256:1c785106e94bd31454936109515cf624a0fe615c499d1129073ae54e9754d8ad size: 3463
● cristiano@MacBook-Air-di-Cristiano LiveLink % heroku container:release web --app livelink-docker
Releasing images web to livelink-docker... done
● cristiano@MacBook-Air-di-Cristiano LiveLink % heroku open --app livelink-docker

```

Si può avere la certezza che l'app è stata deployata correttamente aprendo la sezione overview di Heroku:

The screenshot shows the Heroku Overview page for a specific application. At the top, there's a navigation bar with the Heroku logo and a link to 'Jump to Favorites, Apps, Pipelines, Spaces...'. Below the navigation, there are three main sections: 'Metrics (last 24hrs)', 'Latest activity', and 'Installed add-ons'.

- Metrics (last 24hrs):** Shows Response Time (26s), Throughput (< 1 rps), and Memory (5%).
- Latest activity:** Lists recent events:
 - c.marzano1@studenti.poliba.it: Deployed web (83d9e71647c0) Today at 1:47 AM · v3
 - c.marzano1@studenti.poliba.it: Enable Logplex Yesterday at 7:15 PM - v2
 - c.marzano1@studenti.poliba.it: Initial release Yesterday at 7:15 PM - v1
- Installed add-ons:** Shows '\$0.00/month' and a button to 'Configure Add-ons'.

In the center, there's a box stating 'There are no add-ons for this app' with a note: 'You can add add-ons to this app and they will show here. [Learn more](#)'.

Below these sections is the 'Dyno formation' section, which costs '\$7.00/month' and is currently set to 'ON'. It shows the command 'web npm start'.

At the bottom, there's a 'Collaborator activity' section with a link to 'Manage Access'.

4 KUBERNETES

Kubernetes è un sistema open-source di orchestrazione dei container che offre un insieme di strumenti e risorse per automatizzare il deployment, la scalabilità e la gestione di applicazioni containerizzate.

Il concetto fondamentale di Kubernetes è il **pod**, che rappresenta una singola istanza di un'applicazione o di un servizio all'interno di un cluster Kubernetes.

I *pod* sono gruppi di uno o più container che vengono gestiti come un'entità coesa e condividono risorse come lo spazio di archiviazione, l'indirizzo IP e i volumi.

Kubernetes fornisce un ambiente in cui i pod possono essere creati, pianificati, distribuiti, monitorati e scalati in modo efficiente.

Una delle caratteristiche distintive di Kubernetes è la sua capacità di scalare le applicazioni in modo dinamico, infatti, è possibile definire le regole di scaling automatico in base a metriche predefinite, come l'utilizzo della CPU o il numero di richieste al secondo, consentendo alle applicazioni di adattarsi alle variazioni di carico di lavoro in modo autonomo.

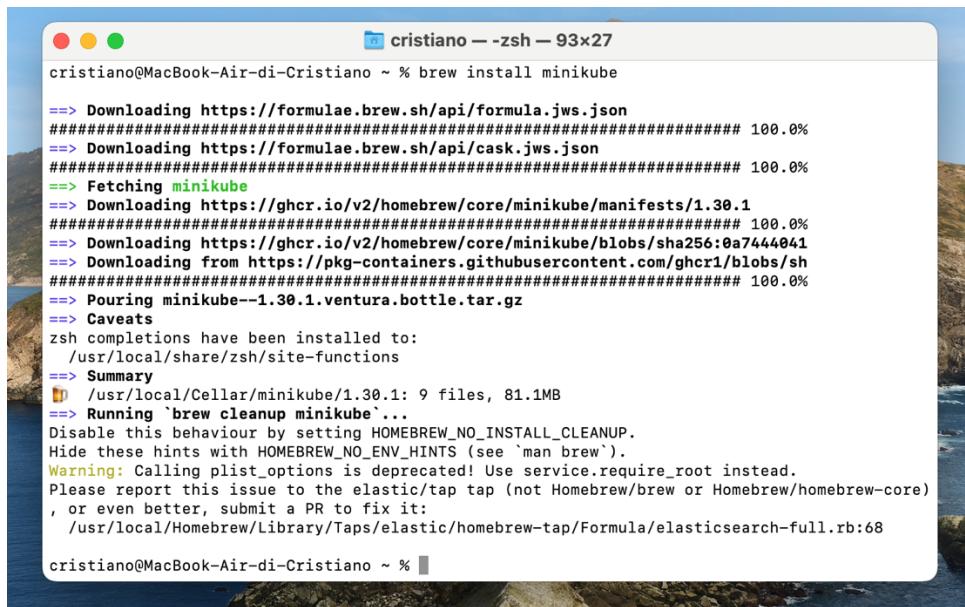
Kubernetes gestisce anche la distribuzione delle applicazioni in modo dichiarativo attraverso i "manifesti", cioè file di configurazione YAML o JSON che descrivono lo stato desiderato dell'applicazione, inclusi i pod, i servizi, le variabili d'ambiente, le risorse e le dipendenze.

Questi manifesti vengono utilizzati per creare e gestire le risorse nel cluster Kubernetes, consentendo di definire l'infrastruttura dell'applicazione in modo riproducibile.

4.1 CONFIGURAZIONE DI KUBERNETES

Il primo passaggio la configurazione di Kubernetes è stato quello di installare tramite Homebrew (solo se si usa MacOS) minikube lanciando il comando:

- brew install minikube



```
cristiano@MacBook-Air-di-Cristiano ~ % brew install minikube

==> Downloading https://formulae.brew.sh/api/formula.jws.json
#####
# 100.0%
==> Downloading https://formulae.brew.sh/api/cask.jws.json
#####
# 100.0%
==> Fetching minikube
==> Downloading https://ghcr.io/v2/homebrew/core/minikube/manifests/1.30.1
#####
# 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/minikube/blobs/sha256:0a744401
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:0a744401
#####
# 100.0%
==> Pouring minikube--1.30.1.ventura.bottle.tar.gz
==> Caveats
zsh completions have been installed to:
  /usr/local/share/zsh/site-functions
==> Summary
  /usr/local/Cellar/minikube/1.30.1: 9 files, 81.1MB
==> Running 'brew cleanup minikube'...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
Warning: Calling plist_options is deprecated! Use service.require_root instead.
Please report this issue to the elastic/tap tap (not Homebrew/brew or Homebrew/homebrew-core)
, or even better, submit a PR to fix it:
  /usr/local/Homebrew/Library/Taps/elastic/homebrew-tap/Formula/elasticsearch-full.rb:68

cristiano@MacBook-Air-di-Cristiano ~ %
```

In seguito, abbiamo lanciato il seguente comando per avviare minikube:

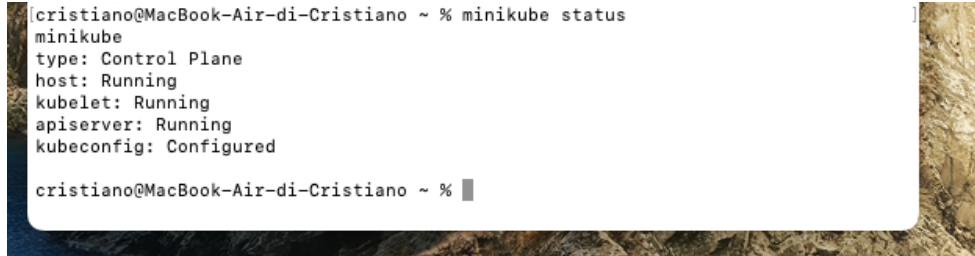
- minikube start



```
Last login: Sun Jun 18 03:19:27 on ttys005
cristiano@MacBook-Air-di-Cristiano ~ % minikube start
😄 minikube v1.30.1 on Darwin 13.2.1
👉 Using the docker driver based on existing profile
👍 Starting control plane node minikube in cluster minikube
🌐 Pulling base image ...
🐳 Restarting existing docker container for "minikube" ...
🌐 Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
🌐 Configuring bridge CNI (Container Networking Interface) ...
🔎 Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
💡 Enabled addons: storage-provisioner, default-storageclass
🎉 Done! kubectl is now configured to use "minikube" cluster and "default" name
space by default
cristiano@MacBook-Air-di-Cristiano ~ %
```

Infine, per verificarne lo stato abbiamo lanciato il comando

- `minikube status`

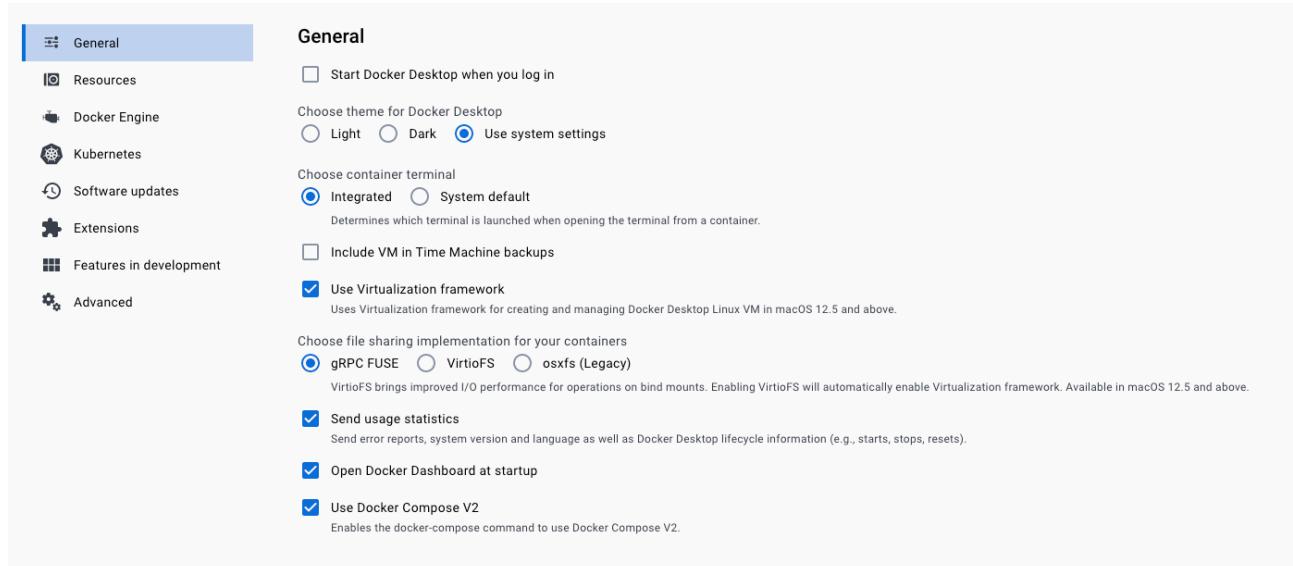


```
[cristiano@MacBook-Air-di-Cristiano ~ % minikube status
minikube
  type: Control Plane
  host: Running
  kubelet: Running
  apiserver: Running
  kubeconfig: Configured
cristiano@MacBook-Air-di-Cristiano ~ % ]
```

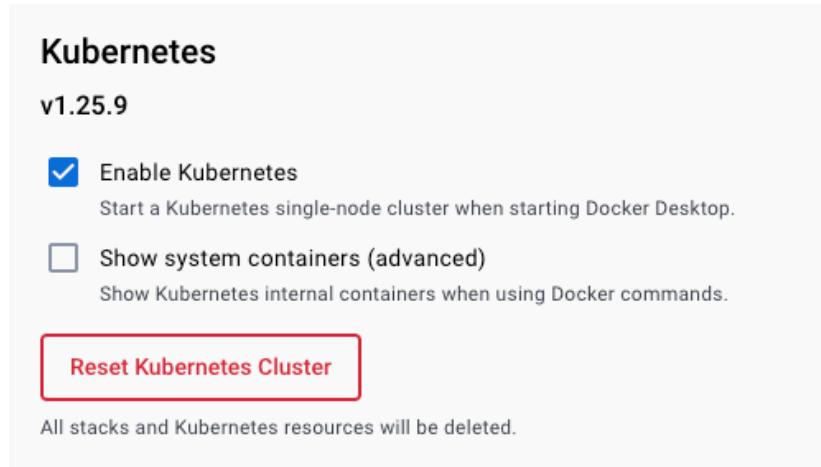
4.2 KUBERNETES E DOCKER DESKTOP

Il primo passo della configurazione di Kubernetes con Docker Desktop è stato quello di inizializzarlo attraverso Docker Desktop nel seguente modo:

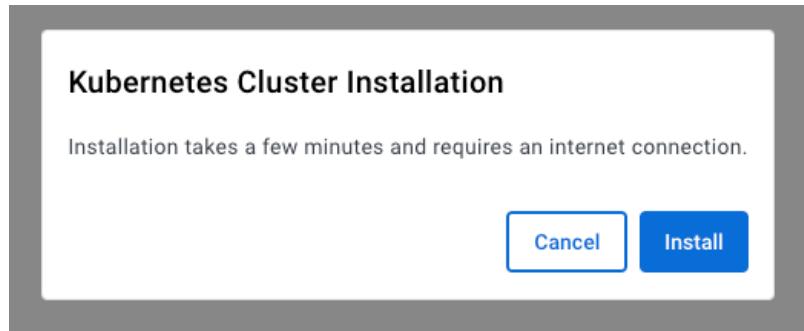
- ci siamo recati nelle impostazioni di Docker desktop e nel menù laterale abbiamo selezionato la voce Kubernetes.



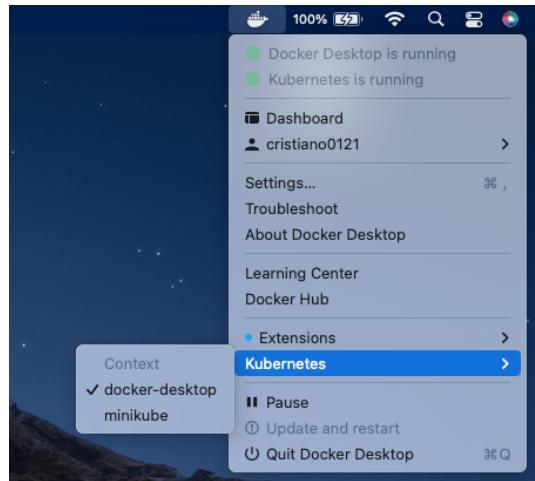
- è stato necessario spuntare la checkbox **Enable Kubernetes**



- dopo aver confermato la scelta premendo il bottone in basso a destra apply & restart ci è stato chiesto di installare il cluster Kubernetes e noi abbiamo acconsentito.



- infine, è stato necessario selezionare Docker Desktop tra le possibili soluzioni per utilizzare Kubernetes.

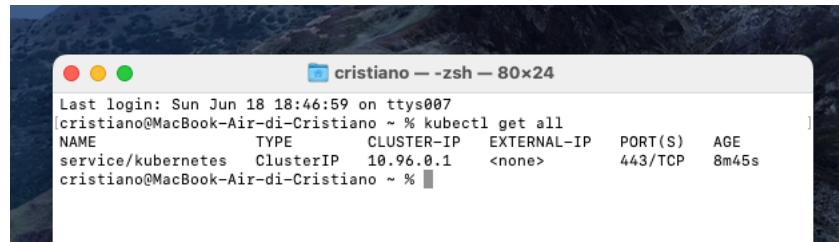


Il passo successivo al collegamento con Docker Desktop è stato necessario verificare che Kubernetes stesse funzionando correttamente e non ci fossero errori attraverso il comando

- `kubectl get all`

il quale viene utilizzato per ottenere informazioni su tutti gli oggetti Kubernetes presenti nel cluster corrente.

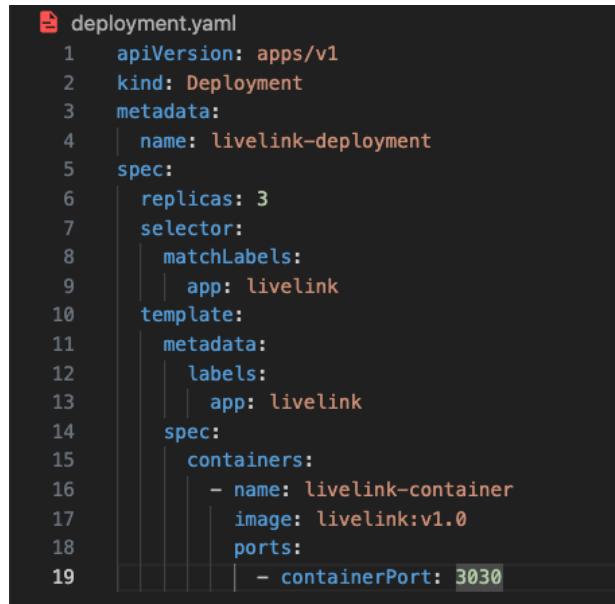
Come output restituisce una tabella che elenca diversi tipi di oggetti, come pod, servizi, deployment, replica sets e altri, insieme alle relative informazioni di stato.



```
Last login: Sun Jun 18 18:46:59 on ttys007
[cristiano@MacBook-Air-di-Cristiano ~ % kubectl get all
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)      AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP     8m45s
cristiano@MacBook-Air-di-Cristiano ~ %
```

Inoltre, è stato necessario creare due nuovi file fondamentali per il corretto funzionamento di Kubernetes:

- `Deployment.yaml`



```
deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: livelink-deployment
5  spec:
6    replicas: 3
7    selector:
8      matchLabels:
9        app: livelink
10   template:
11     metadata:
12       labels:
13         app: livelink
14     spec:
15       containers:
16         - name: livelink-container
17           image: livelink:v1.0
18           ports:
19             - containerPort: 3030
```

- Service.yaml

```
service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: livelink-service
5  spec:
6    selector:
7      app: livelink
8    ports:
9      - protocol: TCP
10     port: 80
11     targetPort: 3030
12   type: LoadBalancer
```

Il passaggio successivo è stato quello di verificare se esistessero già risorse collegate a Kubernetes lanciando il seguente comando:

- kubectl get pods

che viene utilizzato per ottenere informazioni sui pod presenti nel cluster Kubernetes corrente.

Come output viene restituita una tabella che elenca i pod presenti nel cluster insieme alle relative informazioni di stato.

In questo caso non abbiamo la tabella popolata perché eravamo ancora in fase di configurazione.

```
[cristiano@MacBook-Air-di-Cristiano ~ % kubectl get pods
No resources found in default namespace.
cristiano@MacBook-Air-di-Cristiano ~ % ]
```

Infatti, il passo successivo è stato quello di iniziare a cerarne uno con il seguente comando:

- kubectl apply -f deployment.yaml

che viene utilizzato per applicare o aggiornare le risorse di un deployment nel cluster Kubernetes utilizzando il file di configurazione YAML.

Quando viene eseguito il comando, Kubernetes legge il file di configurazione YAML specificato che contiene la definizione del deployment che descrive come Kubernetes deve creare, gestire e scalare le repliche di un'applicazione all'interno del cluster.

È importante notare che *kubectl apply* è un comando dichiarativo, il che significa che Kubernetes si preoccupa di portare lo stato effettivo del cluster allineato con lo stato desiderato definito nel file di configurazione YAML.

```
[cristiano@MacBook-Air-di-Cristiano LiveLink % kubectl apply -f deployment.yaml ]  
deployment.apps/livelink-deployment created  
cristiano@MacBook-Air-di-Cristiano LiveLink %
```

Lanciando nuovamente il seguente comando si possono vedere le risorse che sono state create.

- `kubectl get pods`

```
[cristiano@MacBook-Air-di-Cristiano LiveLink % kubectl get pods  
NAME READY STATUS RESTARTS AGE  
livelink-deployment-778c49c95f-5slh4 1/1 Running 0 2m11s  
livelink-deployment-778c49c95f-kcvzs 1/1 Running 0 2m11s  
livelink-deployment-778c49c95f-qsz4n 1/1 Running 0 2m11s  
cristiano@MacBook-Air-di-Cristiano LiveLink %
```

Il passaggio successivo è stato quello di creare il service.yaml utilizzando lo stesso comando che abbiamo lanciato per il file di deployment.yaml e che ha lo stesso significato.

- `kubectl apply -f service.yaml`

In questo caso Kubernetes legge il file YAML e crea o aggiorna il servizio corrispondente nel cluster in base alle specifiche fornite.

Se il servizio esiste già, Kubernetes gestisce automaticamente l'aggiornamento delle configurazioni o delle regole di selezione dei pod.

```
[cristiano@MacBook-Air-di-Cristiano LiveLink % kubectl apply -f service.yaml ]  
service/livelink-service created  
cristiano@MacBook-Air-di-Cristiano LiveLink %
```

Lanciando il comando

- `kubectl get all`

si ottengono informazioni su tutti gli oggetti Kubernetes presenti nel cluster corrente sottoforma di una tabella che elenca diversi tipi di oggetti, come pod, servizi, deployment, replica sets insieme alle relative informazioni di stato.

```
[cristiano@MacBook-Air-di-Cristiano LiveLink % kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/livelink-deployment-778c49c95f-5slh4   1/1     Running   0          6m30s
pod/livelink-deployment-778c49c95f-kcvzs   1/1     Running   0          6m30s
pod/livelink-deployment-778c49c95f-qsz4n   1/1     Running   0          6m30s

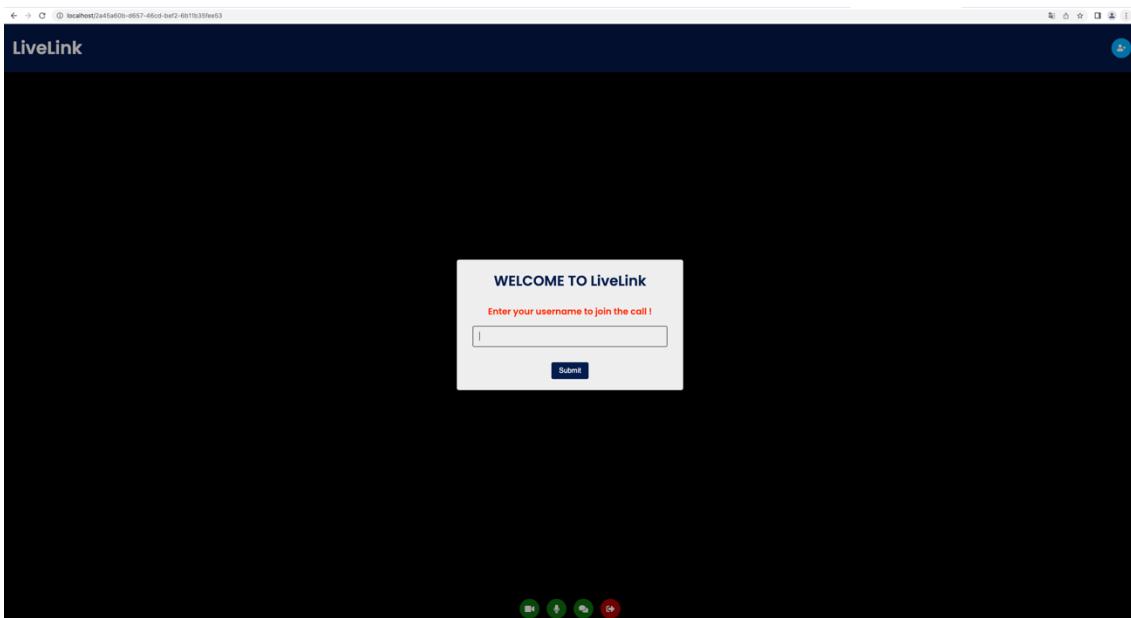
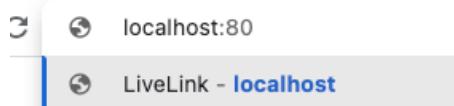
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)         AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP       32m
service/livelink-service   LoadBalancer  10.98.5.159   localhost     80:31599/TCP   2m11s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/livelink-deployment   3/3     3           3          6m30s

NAME                DESIRED   CURRENT   READY   AGE
replicaset.apps/livelink-deployment-778c49c95f   3        3        3       6m30s
cristiano@MacBook-Air-di-Cristiano LiveLink % ]
```

Dopo tutti questi passaggi l'app è in esecuzione e possiamo verificarlo collegandoci al seguente indirizzo web

- Localhost:80



5 LOG

Nella seconda fase del progetto abbiamo posto l'attenzione sui log e sulla loro analisi, partendo dal lato sviluppo è stato necessario scrivere una funzione che intercettasse i log che noi stessi abbiamo ritenuto più significativi e che li incanalasse in un file di testo così da poter essere analizzati da software appositi.

La funzione scritta è la seguente:

```
199 function logVideoStreamInfo(stream) {
200   const logs = [];
201
202   const videoResolution = stream.getVideoTracks()[0].getSettings().width + "x" + stream
203   if (videoResolution !== "undefinedxundefined") [
204     logs.push("Video resolution: " + videoResolution);    Cristiano Marzano, 3 hours
205   ]
206
207   const frameRate = stream.getVideoTracks()[0].getSettings().frameRate;
208   if (frameRate !== undefined) {
209     logs.push("Frame rate: " + frameRate);
210   }
211
212   const audioLatency = stream.getAudioTracks()[0].getSettings().latency;
213   if (audioLatency !== undefined) {
214     logs.push("Audio Latency: " + audioLatency + " seconds");
215   }
216
217   const noiseSuppression = stream.getAudioTracks()[0].getSettings().noiseSuppression;
218   if (noiseSuppression !== undefined) {
219     logs.push("Noise suppression: " + noiseSuppression);
220   }
221 }
```

Nella prima parte dell'immagine, all'interno della funzione **logVideoStreamInfo()** vengono dichiarate diverse variabili:

- "logs": rappresenta un array che conterrà le informazioni registrate.
- videoResolution: serve per recuperare le informazioni relative al video dallo stream passato come argomento ottenendo la traccia video e da essa vengono estratte le informazioni come la larghezza e l'altezza del video.

Se la risoluzione video non è "undefinedxundefined", viene creata una stringa con la risoluzione e viene aggiunta all'array *logs* con la relativa etichetta **Video resolution**.

- frameRate: vengono recuperate le informazioni relative al frame rate dello stream video, inoltre, se il frame rate è definito, allora, viene aggiunto anch'esso all'array *logs* con l'etichetta **Frame rate**.
- audioLatency: vengono recuperate le informazioni relative alla traccia audio dello stream.
Se la latenza audio è definita, viene aggiunta all'array *logs* con l'etichetta **Audio Latency**.
- noiseSuppression: riguarda le informazioni relative alla soppressione del rumore audio dalla traccia audio dello stream, se la soppressione del rumore è definita, allora, viene aggiunta all'array *logs* con l'etichetta **Noise suppression**.

Dopo aver raccolto queste informazioni, la funzione crea una stringa di log unendo tutto il materiale ottenuto così da poterlo visualizzare sulla console del browser.

```

244  const logText = logs.join("\n");
245  console.log(logText);
246
247  // Write logs to a text file
248  if (logs.length > 0) {      Cristiano Marzano, 4 hours ago • cgan
249    const filename = "log.txt";
250    const element = document.createElement("a");
251    element.setAttribute(
252      "href",
253      "data:text/plain;charset=utf-8," + encodeURIComponent(logText)
254    );
255    element.setAttribute("download", filename);
256    element.style.display = "none";
257    document.body.appendChild(element);
258    element.click();
259    document.body.removeChild(element);
260  }
261 }
262

```

Successivamente, la funzione crea un elemento `<a>` nel documento HTML e imposta i suoi attributi per consentire il download del file di log.

L'elemento `<a>` viene quindi aggiunto al corpo del documento, viene simulato un click sull'elemento per avviare il download e infine l'elemento viene rimosso dal documento così facendo il file di log viene scaricato dal browser dell'utente.

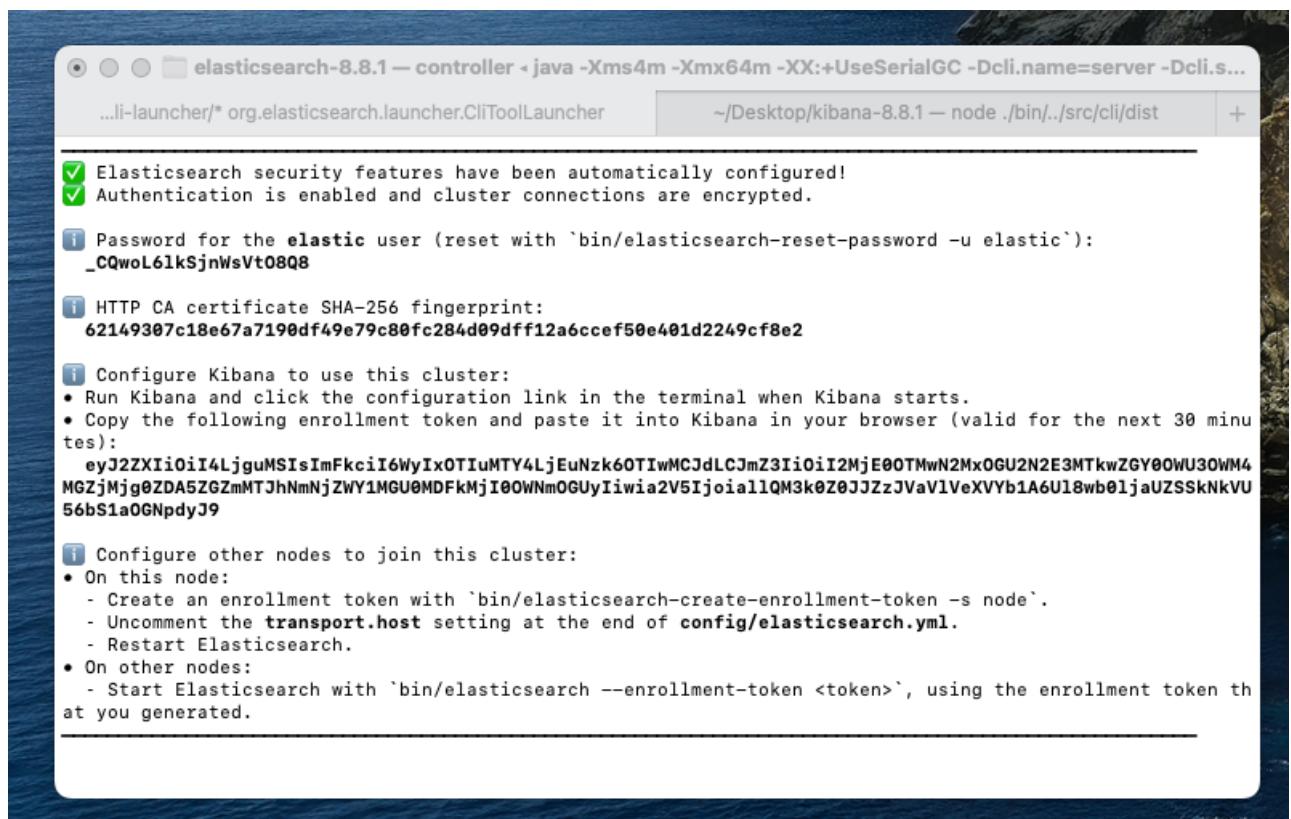
5.1 CONFIGURAZIONE DI ELASTICSEARCH & KIBANA

Dopo aver installato Elasticsearch e Kibana direttamente dal sito ufficiale di Elasticsearch (<https://www.elastic.co/downloads/elasticsearch> & <https://www.elastic.co/downloads/kibana>), il primo passaggio è stato quello di inizializzare e lanciare i due ambienti.

Partendo da Elasticsearch ci siamo prima posizionati nella directory di elasticsearch che nel nostro caso specifico era `cristiano@MacBook-Air-di-Cristiano elasticsearch-8.8.1` e poi è stato necessario lanciare il comando:

- `bin/elasticsearch`

dopo qualche istante è apparsa una sezione relativa alla configurazione di elasticsearch che conteneva i dati di collegamento e accesso a Kibana.



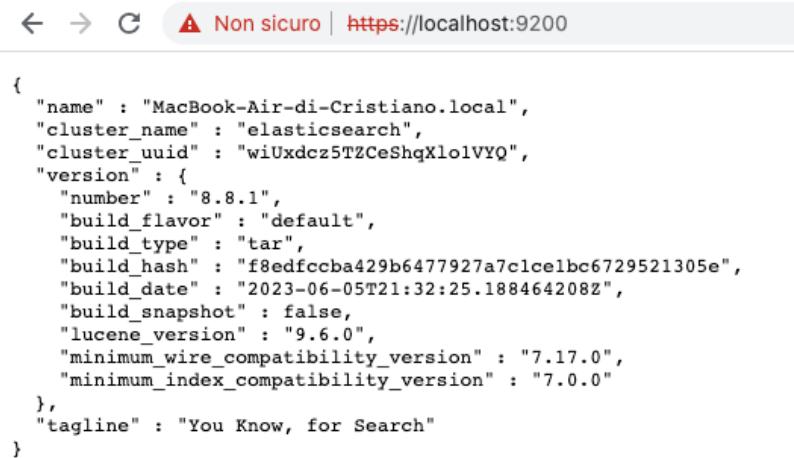
The screenshot shows a terminal window with two tabs. The active tab displays the Elasticsearch configuration output. It includes a success message about security features being automatically configured, and instructions for the elastic user password and HTTP CA certificate fingerprint. It also provides steps for configuring Kibana and other nodes, including enrollment tokens and transport host settings. The terminal window has a dark blue background with white text and a light blue header bar.

```
elasticsearch-8.8.1 -- controller < java -Xms4m -Xmx64m -XX:+UseSerialGC -Dcli.name=server -Dcli.s...  
...li-launcher/* org.elasticsearch.launcher.CliToolLauncher ~/Desktop/kibana-8.8.1 -- node ./bin/../src/cli/dist  
  
✓ Elasticsearch security features have been automatically configured!  
✓ Authentication is enabled and cluster connections are encrypted.  
  
ℹ Password for the elastic user (reset with `bin/elasticsearch-reset-password -u elastic`):  
_CQwoL6lkSjnWsVt08Q8  
  
ℹ HTTP CA certificate SHA-256 fingerprint:  
62149307c18e67a7190df49e79c80fc284d09dff12a6cce50e401d2249cf8e2  
  
ℹ Configure Kibana to use this cluster:  
• Run Kibana and click the configuration link in the terminal when Kibana starts.  
• Copy the following enrollment token and paste it into Kibana in your browser (valid for the next 30 minutes):  
eyJ2ZXIiOiI4LjguMSIsImFkciI6WyIxOTIuMTY4LjEuNzk6OTIwMCJdLCJmZ3IiOii2MjE00TMwN2MxOGU2N2E3MTkwZGY0OWU30WM4  
MGZjMjg0ZDA5ZGzmMTJhNmNjZWY1MGU0MDFkMjI00WNmOGUyIiwiava2V5Ijoia1lQM3k0Z0JJZzJVaV1VeXVYb1A6U18wb01jaUZSSkNkVU  
56bS1aOGNpdyJ9  
  
ℹ Configure other nodes to join this cluster:  
• On this node:  
- Create an enrollment token with `bin/elasticsearch-create-enrollment-token -s node`.  
- Uncomment the transport.host setting at the end of config/elasticsearch.yml.  
- Restart Elasticsearch.  
• On other nodes:  
- Start Elasticsearch with `bin/elasticsearch --enrollment-token <token>`, using the enrollment token that you generated.
```

Per vedere che Elasticsearch fosse stato lanciato correttamente e stesse quindi funzionando è stato necessario collegarsi al sito web:

- <https://localhost:9200/>

Avendo come risposta il seguente file JSON

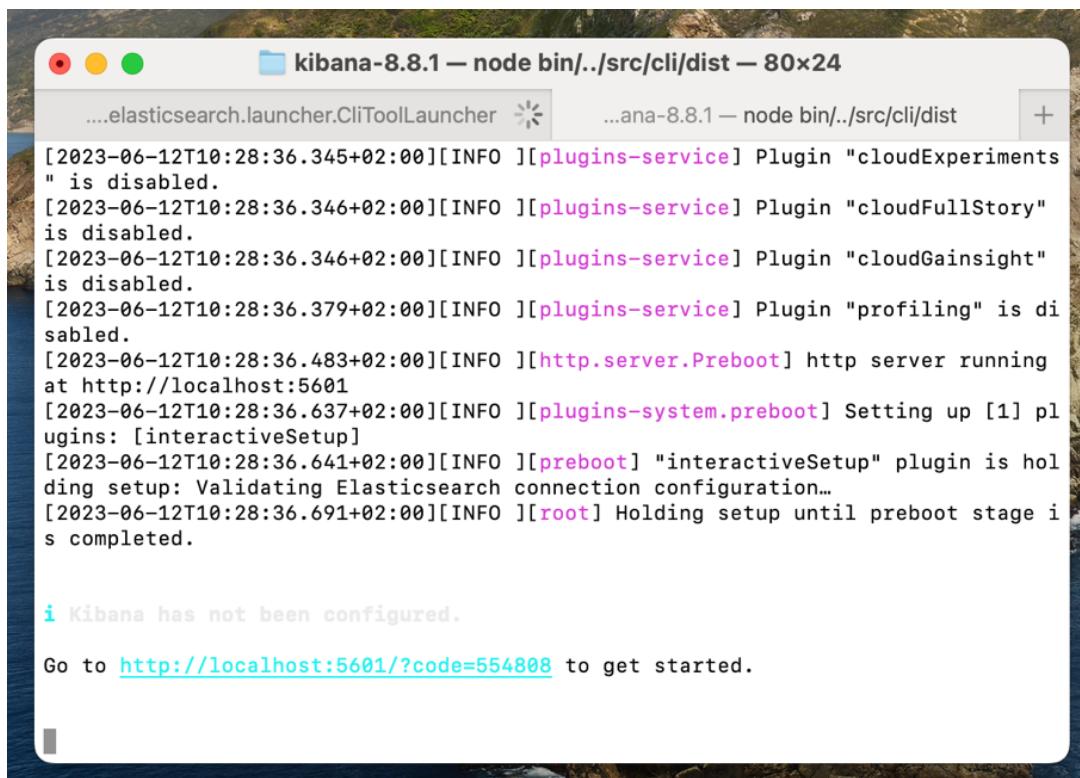


```
{  
  "name" : "MacBook-Air-di-Cristiano.local",  
  "cluster_name" : "elasticsearch",  
  "cluster_uuid" : "wiUxdcz5T2CeShqXlo1VYQ",  
  "version" : {  
    "number" : "8.8.1",  
    "build_flavor" : "default",  
    "build_type" : "tar",  
    "build_hash" : "f8edfccba429b6477927a7c1ce1bc6729521305e",  
    "build_date" : "2023-06-05T21:32:25.188464208Z",  
    "build_snapshot" : false,  
    "lucene_version" : "9.6.0",  
    "minimum_wire_compatibility_version" : "7.17.0",  
    "minimum_index_compatibility_version" : "7.0.0"  
  },  
  "tagline" : "You Know, for Search"  
}
```

A questo punto dopo aver appurato che Elasticsearch fosse stato configurato correttamente ci siamo dedicati alla configurazione di Kibana il quale è stato lanciato anch'esso posizionandosi nella sua directory che in questo caso specifico era cristiano@MacBook-Air-di-Cristiano kibana-8.8.1 attraverso il comando:

- ./bin/kibana

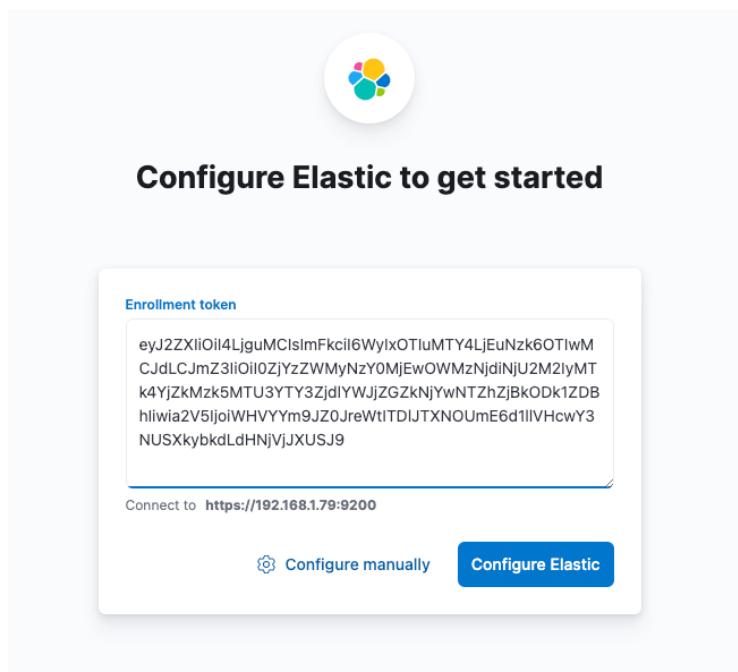
qualche istante sono apparse le seguenti istruzioni per accedere all'interfaccia grafica di Kibana.



Una volta controllato che Kibana funzionasse correttamente ci siamo collegati al seguente indirizzo web per utilizzare la sua interfaccia grafica.

- <http://localhost:5601/>

Come prima cosa ci è stato chiesto il token fornito da Elasticsearch



Poi è stato necessario inserire le credenziali fornite dalla sezione di configurazione di elasticsearch inserendo come nome utente quello di default: elastic e come password: _CQwoL6IkSjnWsVtO8Q8

Two side-by-side screenshots of the Kibana interface. The left screenshot shows the "Welcome to Elastic" screen with fields for "Username" and "Password", and a "Log in" button. The right screenshot shows the "Configure Elastic to get started" screen with a progress bar showing "Saving settings" and "Starting Elastic" completed, and "Completing setup" pending.

Una volta finalizzato correttamente l'accesso siamo riusciti ad accedere all'interfaccia grafica di Kibana così da poter gestire correttamente le nostre analisi relative ai log.

The screenshot shows the Elastic Integrations interface. At the top, there's a navigation bar with the Elastic logo, a search bar, and user icons. Below the navigation is a breadcrumb trail: 'Integrations > Browse integrations'. The main title is 'Integrations' with the subtitle 'Choose an integration to start collecting and analyzing your data.' Below this, there are two tabs: 'Browse integrations' (selected) and 'Installed integrations'. To the left is a sidebar with a tree view of categories and their counts: All categories (318), APM (1), AWS (35), Azure (24), Cloud (3), Containers (15), Custom (34), Database (34), Elastic Stack (30), Elasticsearch SDK (9), Enterprise Search (36), Google Cloud (18). A link 'Display beta integrations' is also present. On the right, there are several integration cards with icons and descriptions:

- APM**: Collect performance metrics from your applications with Elastic APM.
- Elastic Defend**: Protect your hosts and cloud workloads with threat prevention, detection, and deep security data visibility.
- Web crawler**: Add search to your website with the Enterprise Search web crawler.
- 1Password**: Collect logs from 1Password with Elastic Agent.
- AbuseCH**: Ingest threat intelligence indicators from URL Haus, Malware Bazaar, and Threat Fox feeds with Elastic Agent.
- ActiveMQ Logs**: Collect and parse logs from ActiveMQ instances with Filebeat.
- ActiveMQ Metrics**: Collect metrics from ActiveMQ instances with Metricbeat.
- Aerospike Metrics**: Collect metrics from Aerospike servers with Metricbeat.
- Akamai**: Collect logs from Akamai with Elastic Agent.

5.2 CONFIGURAZIONE DEL TD-AGENT (FLUENTD)

Il Td-agent è basato su Fluentd, un progetto open-source per la raccolta e l'elaborazione dei log, abbiamo deciso di utilizzare drettamente lui invece di Fluentd perché fornisce un pacchetto preconfigurato che semplifica l'installazione e la configurazione di Fluentd. Utilizzando il td-agent è possibile raccogliere file di log per trasformarli, normalizzarli e inviarli a software specifici come database, sistemi di archiviazione o piattaforme di analisi dei log, nel nostro caso ad Elasticsearch il quale dopo averli elaborati avrebbe dovuto mandarli a Kibana per la visualizzazione grafica da parte dell'utente.

Il primo passaggio è stato quello di installare il td-agent utilizzando su MacOS Homebrew il seguente comando:

- brew install td-agent

```
cristiano@MacBook-Air-di-Cristiano ~ % brew install td-agent
==> Downloading https://s3.amazonaws.com/packages.treasuredata.com/4/macosx/td-a
#####
#                                           100.0%
==> Installing Cask td-agent
==> Running installer for td-agent; your password may be necessary.
Package installers may write to any location; options such as `--appdir` are ignored.
installer: Package name is Td-agent v4.4.1
installer: Upgrading at base path /
installer: The upgrade was successful!
🍺 td-agent was successfully installed!
cristiano@MacBook-Air-di-Cristiano ~ %
```

Il passaggio successivo è stato quello di installare il plugin fluent-plugin-elasticsearch per connettere Fluentd ad Elasticsearch utilizza il seguente comando:

- sudo gem install fluent-plugin-elasticsearch

```
cristiano@MacBook-Air-di-Cristiano ~ % sudo fluent-gem install fluent-plugin-elasticsearch
[Password:
Successfully installed fluent-plugin-elasticsearch-5.3.0
Parsing documentation for fluent-plugin-elasticsearch-5.3.0
Done installing documentation for fluent-plugin-elasticsearch after 0 seconds
1 gem installed
cristiano@MacBook-Air-di-Cristiano ~ %
```

Nel file di configurazione del td-agent è stato necessario apportare qualche piccola modifica per settare nel modo corretto la comunicazione con elasticsearch, infatti sono state aggiunte due sezioni:

- La prima è stata aggiunta per indicare al td-agent dove andare a recuperare il file di log da inviare ad Elasticsearch.

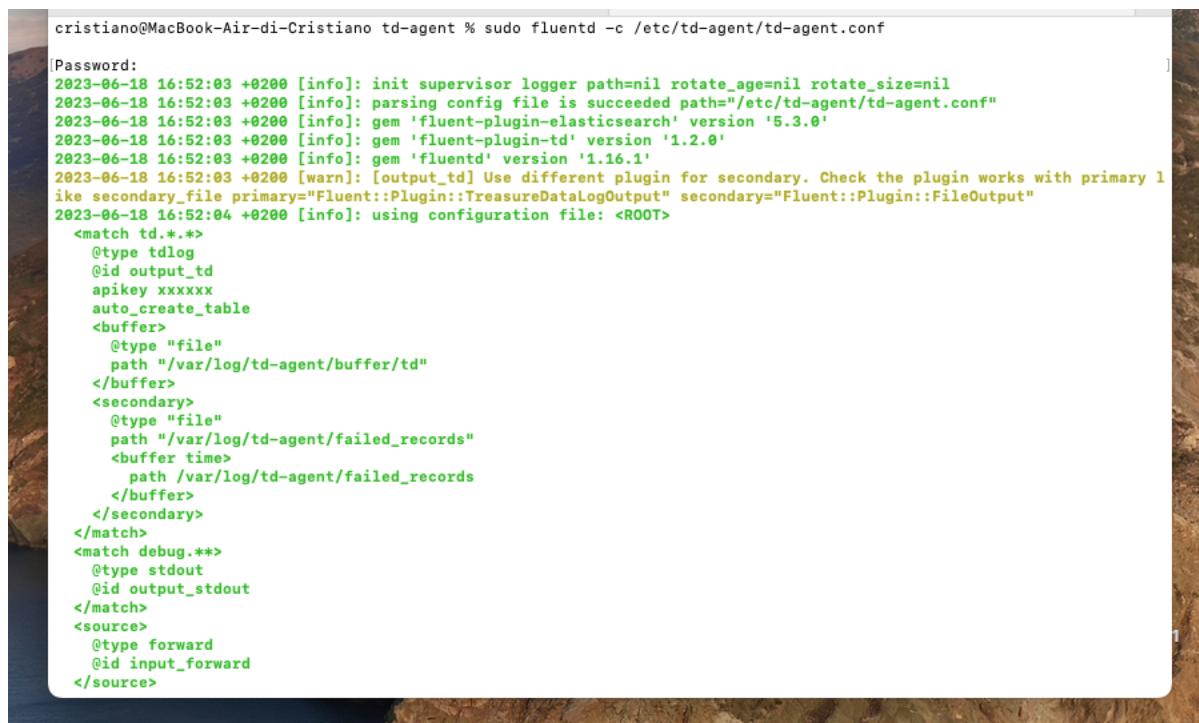
```
84 #####  
85 ## Source descriptions:  
86 ##  
87 <source>  
88   @type tail  
89   path /Users/cristiano/Downloads/*.txt  
90   pos_file /var/log/td-agent/td-agent.log.pos  
91   format json  
92   time_format %Y-%m-%d %H:%M:%S  
93   tag log  
94 </source>
```

- La seconda parte di codice, invece, è servita per indicare a quale output inviare i log recuperati dalla prima sezione.

```
70 #####  
71 ## Output descriptions:  
72 ##  
73 <match *log*>  
74   @type elasticsearch  
75   hosts localhost  
76   port 9200  
77   username elastic  
78   password _CQwoL6lkSjnWsVt08Q8  
79   index_name video  
80   type_name log  
81   verify_certificate false  
82 </match>  
83
```

L'ultimo passaggio è stato quelli di lanciare il td-agent attraverso il seguente comando:

- sudo fluentd -c /etc/td-agent/td-agent.conf



```
cristiano@MacBook-Air-di-Cristiano td-agent % sudo fluentd -c /etc/td-agent/td-agent.conf
Password:
2023-06-18 16:52:03 +0200 [info]: init supervisor logger path=nil rotate_age=nil rotate_size=nil
2023-06-18 16:52:03 +0200 [info]: parsing config file is succeeded path="/etc/td-agent/td-agent.conf"
2023-06-18 16:52:03 +0200 [info]: gem 'fluent-plugin-elasticsearch' version '5.3.0'
2023-06-18 16:52:03 +0200 [info]: gem 'fluent-plugin-td' version '1.2.0'
2023-06-18 16:52:03 +0200 [info]: gem 'fluentd' version '1.16.1'
2023-06-18 16:52:03 +0200 [warn]: [output_td] Use different plugin for secondary. Check the plugin works with primary like secondary_file primary="Fluent::Plugin::TreasureDataLogOutput" secondary="Fluent::Plugin::FileOutput"
2023-06-18 16:52:04 +0200 [info]: using configuration file: <ROOT>
<match td.*.*>
  @type tdlog
  @id output_td
  apikey xxxxxx
  auto_create_table
  <buffer>
    @type "file"
    path "/var/log/td-agent/buffer/td"
  </buffer>
  <secondary>
    @type "file"
    path "/var/log/td-agent/failed_records"
    <buffer time>
      path /var/log/td-agent/failed_records
    </buffer>
  </secondary>
</match>
<match debug.*.*>
  @type stdout
  @id output_stdout
</match>
<source>
  @type forward
  @id input_forward
</source>
```

LINK

LiveLink: <https://livelink-docker.herokuapp.com>

Docker Repo: <https://hub.docker.com/repository/docker/cristiano0121/livelink/general>

GitHub Repo: <https://github.com/Cristiano-21/LiveLink>