📡 WebSocket Client Guide

This document provides a detailed guide for integrating and working with the WebSocket API in **Cluster Code Service**.

🔍 Overview

The WebSocket API allows clients to receive real-time updates and events from the Cluster Code Service backend. Clients must authenticate with a short-lived token and subscribe to a channel to start receiving relevant events.

🧪 Test Environment (HTML Example)

You can use the following basic HTML page to test your WebSocket integration:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>WebSocket Test Client</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        #messages { border: 1px solid #ccc; padding: 10px; height: 200px; overflow-y: scr
        .message { margin-bottom: 10px; }
    </style>
</head>
<body>

<h2>WebSocket Test Client</h2>

<div id="messages"></div>

<script>
    const token = "your-short-token-here"; // Replace with actual short token
    const wsUrl = `ws://localhost:8070/ws?token=${token}`;

    const socket = new WebSocket(wsUrl);
    const messagesDiv = document.getElementById("messages");
```

```
    socket.onopen = () => {
        console.log("Connected to WebSocket");

        const subscribeMessage = {
            action: "subscribe",
            channel: "cluster_code_16" // Replace with your target channel
        };
        socket.send(JSON.stringify(subscribeMessage));
    };

    socket.onmessage = (event) => {
        console.log("Received:", event.data);
        let data;
        try {
            data = JSON.parse(event.data);
        } catch (e) {
            console.error("Failed to parse message JSON", e);
            return;
        }

        if (data.channel === "cluster_code_16") {
            const messageEl = document.createElement("div");
            messageEl.className = "message";
            messageEl.textContent = `Event: ${data.event_type}, Data: ${JSON.stringify(da
            messagesDiv.appendChild(messageEl);
            messagesDiv.scrollTop = messagesDiv.scrollHeight;
        }
    };

    socket.onclose = () => {
        console.log("WebSocket connection closed");
    };

    socket.onerror = (err) => {
        console.error("WebSocket error", err);
    };
</script>

</body>
</html>
```

🛡️ Authentication Flow

1. Get Short Token

To initiate a WebSocket session, you need a short-lived token obtained by calling:

```
POST /api/v1/auth
Authorization: Bearer {main_token}
```

Response:

```
{
    "success": true,
    "data": {
        "token": "short_token"
    }
}
```

Use this token in the WebSocket connection.

🌐 Connect to WebSocket

After obtaining a short token, connect to the WebSocket endpoint:

```
ws://{host}:{port}/ws?token={short_token}
```

> Use `wss://` for secure connections (SSL/TLS).

📥 Subscribing to Channels

After the connection is open, you must subscribe to a specific channel to receive messages.

Subscription Format

```
{
  "action": "subscribe",
  "channel": "cluster_code_16"
}
```

📤 Receiving Messages

Once subscribed, the server will send structured messages when events occur:

Message Example

```
{
  "event_type": "workspace_created",
  "channel": "cluster_code_16",
  "data":  {"id":11,"title":"test workspace","color":"#124376","ide":"vscode","url":"","r
}
```

Go Struct (Server Side)

```go
type Message struct {
    EventType constants.EventType `json:"event_type"`
    Channel   string              `json:"channel"`
    Data      interface{}         `json:"data"`
}
```

🧑‍💻 Message Handling Tips

- Always **validate and parse** incoming JSON.
- Use the `event_type` field to distinguish message types (e.g., `log_update`, `status_change`, `container_ready`).
- The `channel` should match the one you subscribed to.
- Expect different payload shapes under `data` depending on the event type.

🔌 Connection Handling

onopen

Establish connection and send subscription.

onmessage

Receive and process incoming events.

onclose

Handle connection drops gracefully and consider reconnecting.

onerror

Log or report errors for diagnostics.

📌 Best Practices

- Refresh short tokens before expiry (they are temporary).
- Use secure connections (`wss://`) in production environments.
- Avoid subscribing to too many channels over one connection.
- Implement retry logic for unstable networks.

🧾 Summary

Step	Action
🔐 Get short token	`POST /api/v1/auth` with main token
🌐 Connect	`ws://host/ws?token=short_token`
📣 Subscribe	`{ "action": "subscribe", "channel": "..." }`
📬 Receive messages	Messages follow the `Message` struct format

🧠 Need Help?

If you're having trouble connecting or receiving events, ensure:

- The backend is running and listening on the correct port.
- The token has not expired.
- Your channel name is correct and mapped on the backend.