Cluster Code Service

Cluster Code Service is the backend component of a cloud development workspace platform, similar to GitHub Workspaces or Gitpod. It enables serving and managing development containers using DevPod, and it handles authentication, container setup, and messaging infrastructure.

This project is built with **Golang**, **PostgreSQL**, and **RabbitMQ**, and is fully Dockerized for easy deployment.

🧩 Features

- Dev container orchestration backend using DevPod
- WebSocket-based real-time communication
- User and machine configuration sync
- RabbitMQ messaging system integration
- Fully containerized with Docker Compose

🛠️ Tech Stack

- **Language**: Go (Golang)
- **Database**: PostgreSQL
- **Message Broker**: RabbitMQ
- **Containers**: Docker & Docker Compose
- **Dev Containers**: DevPod

⚙️ Setup

1. Configure Environment Variables

Create a `.env` file at the root of the project:

```
cp .env.example .env
```

Edit the `.env` file and fill in the required values.

2. Start the Services

Run the following command to start all services using Docker Compose:

```
docker compose up -d
```

3. Run Database Migrations

Inside the container, run:

```
go run cmd/commands/main.go migrate:up
```

4. Import Machine Configurations

```
go run cmd/commands/main.go import-machine-configs
```

5. Sync Users

Run the same import command again to sync users:

```
go run cmd/commands/main.go import-machine-configs
```

> Note: This command currently handles both machine configuration and user sync.

6. Add a Provider

Insert a provider record manually into the PostgreSQL database using a DB client such as:

- `psql`
- `pgAdmin`
- `DBeaver`

Make sure required fields like name, URL, credentials, etc., are properly configured.

7. Create RabbitMQ Exchange

Create an exchange in RabbitMQ with the following name:

```
apps.clusterix-files-v2
```

You can create this via the RabbitMQ Management UI or programmatically using RabbitMQ libraries.

🔌 WebSocket Usage

Step 1: Obtain a Short Token

Send a `POST` request to authenticate using your main token:

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

Response:

```
{
  "short_token": "abc123..."
}
```

Step 2: Connect to WebSocket

Use the short token to connect to the WebSocket server:

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

Step 3: Receive Messages

After connecting, the server will send structured messages like this:

```
{
  "event_type": "some_event_type",
  "channel": "some_channel",
  "data": {
    // payload data
  }
}
```

📦 Message Structure (Go)

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

🤝 Contributing

We welcome contributions! To contribute:

1. Fork the repository
2. Create a new branch
3. Submit a Pull Request

Please make sure your changes are well-documented and tested.

🪪 License

This project is licensed under the MIT License.