### Usare le Docker Engine API con Go

Docker fornisce un'API potente per interagire con il suo engine in modo programmatico. In questo articolo, vedremo come utilizzare le Docker Engine API con Go e il relativo SDK.

## Installazione del Client Docker per Go

Per interagire con Docker da Go, è necessario installare il pacchetto `github.com/docker/docker/client`. Possiamo farlo utilizzando il modulo di Go:

```
go get github.com/docker/docker
```

Assicuriamoci inoltre di avere Docker in esecuzione sul nostro sistema.

## Creare un Client Docker in Go

Per iniziare, dobbiamo creare un'istanza del client Docker:

```
package main

import (
        "context"
        "fmt"
        "github.com/docker/docker/api/types"
        "github.com/docker/docker/client"
)

func main() {
        cli, err := client.NewClientWithOpts(client.FromEnv)
        if err != nil {
                panic(err)
        }
        defer cli.Close()

        version, err := cli.ServerVersion(context.Background())
        if err != nil {
                panic(err)
        }

        fmt.Println("Docker Server Version:", version.Version)
}
```

Il codice sopra crea un client Docker utilizzando l'ambiente di sistema e stampa la versione del server Docker.

## Elencare i Container in Esecuzione

Possiamo elencare i container in esecuzione con il metodo `ContainerList`:

```
func listContainers(cli *client.Client) {
        containers, err := cli.ContainerList(context.Background(),
types.ContainerListOptions{})
        if err != nil {
```

```
                panic(err)
        }

        for _, container := range containers {
                fmt.Println(container.ID[:10], container.Image, container.State)
        }
}
```

## Creare ed Avviare un Container

Per creare un nuovo container, utilizziamo il metodo `ContainerCreate` e successivamente lo avviamo con `ContainerStart`:

```
import "github.com/docker/docker/api/types/container"

func createAndStartContainer(cli *client.Client) {
        resp, err := cli.ContainerCreate(
                context.Background(),
                &container.Config{
                        Image: "nginx",
                },
                nil, nil, nil, "my-nginx-container",
        )
        if err != nil {
                panic(err)
        }

        err = cli.ContainerStart(context.Background(), resp.ID, types.ContainerStartOptions{})
        if err != nil {
                panic(err)
        }

        fmt.Println("Container avviato con ID:", resp.ID)
}
```

## Conclusione

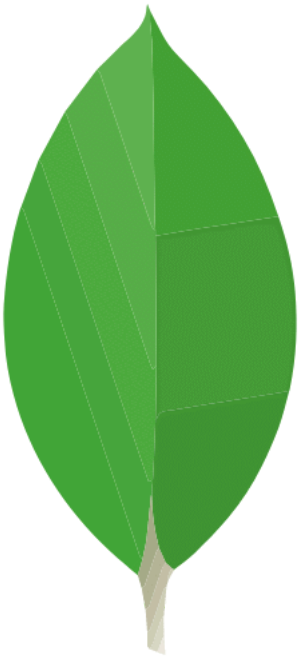
Abbiamo visto come utilizzare le Docker Engine API con Go e il relativo SDK per interagire con Docker in modo programmatico. Questo ci permette di gestire container e risorse in modo flessibile e automatizzato.

## Applicazioni Correlate

- 

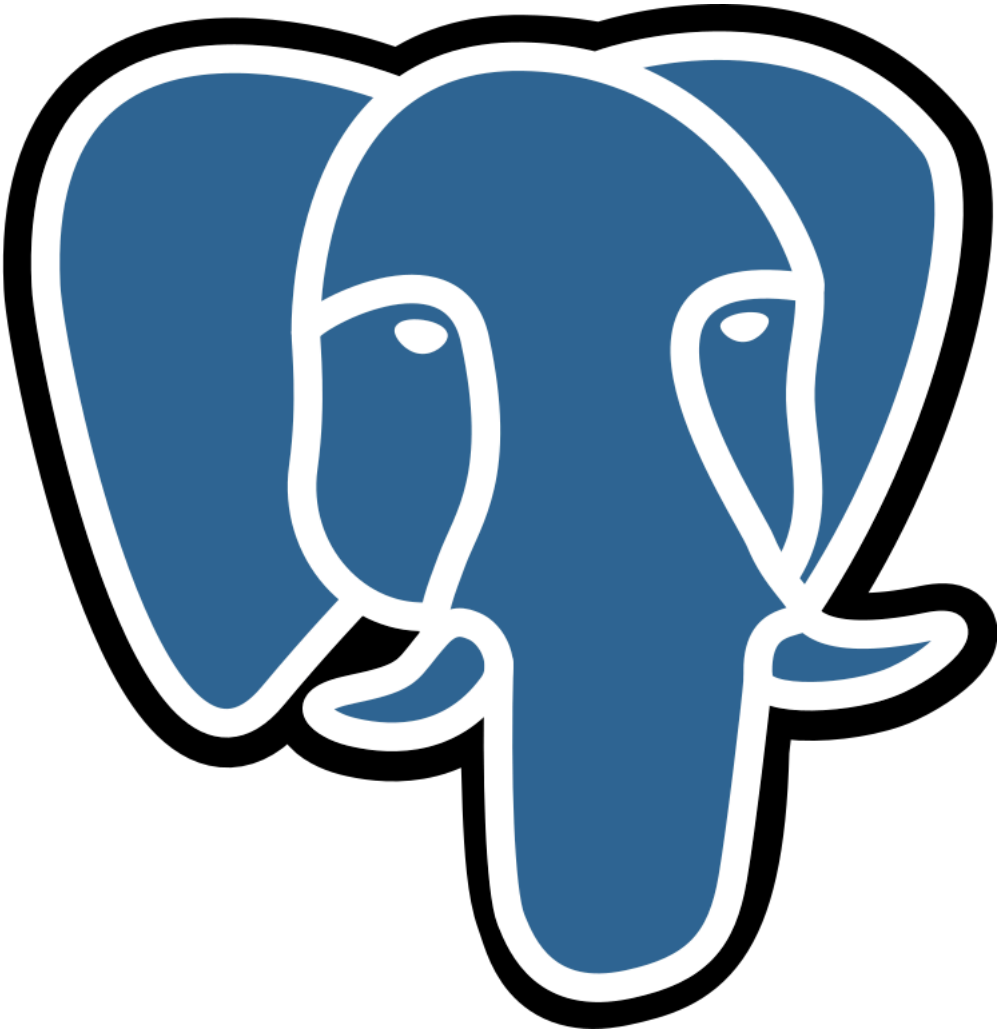
  **Go Placeholder Image**

  Applicazione in Go per la creazione di immagini segnaposto.
  DockerDocker ComposeGoJavaScript



### Go MongoDB App

Applicazione basata su MongoDB ed implementata in Go con il driver ufficiale.
DockerDocker ComposeGoMongoDB



## Go PostgreSQL App

Applicazione basata su PostgreSQL e sviluppata in Go.
DockerDocker ComposeGoPostgreSQL