

# Golang SDK

[suave-geth provides an SDK](#) with a set of tools to interact with SUAVE, including deploying contracts, sending transactions, and waiting for transaction receipts.

## Installation

Make sure you have Go installed on your system and your workspace is set. Then import the SDK package into your Go project:

```
import
```

"github.com/ethereum/go-ethereum/suave/sdk" Then, introduce a replace statement in your go.mod file to point to the SUAVE Geth fork:

```
replace github.com/ethereum/go-ethereum =
```

```
github.com/flashbots/suave-geth This is a temporary setup, and the SDK will later be spawned into its own library.
```

## Usage

### Creating a Client

To start interacting with Ethereum smart contracts, instantiate a new client:

```
client := sdk.NewClient(rpcClient, privateKey, executionNodeAddress) * rpcClient * : Your Ethereum RPC client. *
privateKey * : An ECDSA private key for signing transactions. * executionNodeAddress * : The address of a SUAVE Kettle.
Use 0x03493869959c866713c33669ca118e774a30a0e5 * if working on Rigil.
```

One way to instantiate a golang RPC client using the go-ethereum rpc module "github.com/ethereum/go-ethereum/rpc" is:

## exNodeNetAddr

```
"http://localhost:8545" rpcClient ,
```

```
—
```

```
:= rpc.Dial(exNodeNetAddr)
```

### Deploying a Contract

Deploy a smart contract to the network:

```
transactionResult, err := sdk.DeployContract(bytecode, client) * bytecode * : The compiled bytecode of the smart
contract. * client * : An instance of yourClient * .
```

### Sending a Transaction

Interact with a contract by sending a transaction:

```
transactionResult, err := contract.SendTransaction(methodName, args, confidentialData) * methodName * : The name
of the contract method to call. * args * : Arguments for the method call. * confidentialData * : Confidential data bytes for the
transaction.
```

### Transaction Result

After sending a transaction, you can query the result and receipt:

```
receipt, err := transactionResult.Wait() Wait for the transaction to be mined and get the receipt.
```

```
hash := transactionResult.Hash() Retrieve the hash of the transaction.
```

## Client Methods

The following are key methods available on the Client type:

- `RPC()`
- : Retrieves the underlying RPC client.
- `SignTxn(*types.LegacyTx)`
- : Signs a transaction with the provided private key.
- `SendTransaction(*types.LegacyTx)`
- : Sends a signed transaction to the network.[Edit this page](#) [Previous](#) [Forge](#) [Next](#) [TypeScript SDK](#)