Web3 Solana Blockchain - 02

As a developer, typical questions you might have include:

- How do I connect and talk to the Solana blockchain?
- How can I create my account on the Solana blockchain?
- How do I obtain some test tokens to test my applications?

Do not worry. We got you!

In this Note, we will be introducing four key concepts:

1. Solana clusters (networks) - this is similar to the different environments you may code in today
2. Connecting to the Solana devnet - this would like connecting to a test environment
3. Creating an account represented by a keypair - this is kind of like creating an API
4. Sending the token, or currency of Solana, called SOL into our account. This is called an airdrop which you will learn all about.

Clusters

As a developer, when we build an application on a blockchain, we can deploy our applications on various simulated blockchain networks called test networks. These networks are also known as clusters in Solana. Solana maintains several different clusters with different purposes.

The three popular types of clusters in Solana are:

1. **Devnet**: Devnet serves as a playground for anyone who wants to take Solana for a test drive, as a user, token holder, app developer, or validator. Think of this as the local development environment.
2. **Testnet**: Testnet is where we stress-test recent release features on a live cluster, particularly focused on network performance, stability and validator behavior. Think of this as a test environment.
3. **Mainnet Beta**: This is the main network where Solana apps can be deployed and interacted with, using real SOL. Think of this as the live environment.

SOL is Solana’s native **token**. Tokens are digitally native, programmable and secured by one’s crypto wallet and private key. The system can perform micropayments of factional SOLs, called **lamports** which have a value of 0.000000001 SOL.

Wallet / Keypair

A wallet is your keypair - this is what holds the SOL tokens. A keypair has a public and private key.

Keys

- **Public Key**: This publicly available key is how others can send you transactions.
- **Private Key**: This allows you to prove ownership and use the cryptocurrency associated with your public address. Never share this with anyone.

```
Important Functions for Extracting these keys

keypair.publickey
keypair.secretkey
```

Think of keys like a mailbox. The public key is the address of the box, where anyone can send drop mail (transactions) in the slot. Only the person with the private key can open the mailbox (prove ownership) and read the messages (use the transactions).

Key-pair Code

```
const getWalletBalance = async () => {

try {

// Connect to the Devnet

const connection = new Connection(clusterApiUrl("devnet"), "confirmed");

console.log("Connection object is:", connection);



// Make a wallet (keypair) from privateKey and get its balance

const myWallet = await Keypair.fromSecretKey(privateKey);

const walletBalance = await connection.getBalance(

new PublicKey(newPair.publicKey)

);

console.log(`Wallet balance: ${parseInt(walletBalance) / LAMPORTS_PER_SOL} SOL`);

} catch (err) {

console.log(err);
```

```
}

};
```

The code snippet you provided defines an asynchronous function called `getWalletBalance`. Let's break down what it does in a paragraph:

The function aims to retrieve the balance of a Solana wallet. To achieve this, it performs the following steps:

First, it establishes a connection to the Solana Devnet (a development network) using the `Connection` object. This connection allows the function to interact with the Solana blockchain and perform operations like retrieving balances.

Next, it generates a wallet (also known as a keypair) using the private key stored in the `privateKey` variable. A keypair consists of a private key (which should be kept secret) and a corresponding public key (used for identification and transactions).

After obtaining the keypair, the function retrieves the balance of the wallet by calling the `getBalance` method on the Solana connection object. It passes the public key derived from the keypair as an argument to get the specific wallet's balance.

Finally, the balance is logged to the console in a human-readable format by dividing the obtained balance (which is in lamports, the smallest unit of Solana currency) by the conversion factor `LAMPORTS_PER_SOL` to get the balance in SOL (the native cryptocurrency of Solana).

If any errors occur during the process, they will be caught in the `catch` block, and the error message will be logged to the console.

AirDrops Code

An airdrop process refers to the distribution of tokens or digital assets to a large number of recipients for free. In the context of blockchain technology, an airdrop involves "dropping" or distributing tokens to specific addresses or wallets as a way to promote a project, engage with the community, or incentivize token holders.

```
const airDropSol = async () => {

try {

// Connect to the Devnet and make a wallet from privateKey

const connection = new Connection(clusterApiUrl("devnet"), "confirmed");

const myWallet = await Keypair.fromSecretKey(privateKey);



// Request airdrop of 2 SOL to the wallet
```

```
console.log("Airdropping some SOL to my wallet!");

const fromAirDropSignature = await connection.requestAirdrop(

new PublicKey(myWallet.publicKey),

2 * LAMPORTS_PER_SOL

);

await connection.confirmTransaction(fromAirDropSignature);

} catch (err) {

console.log(err);

}

};
```

The code snippet you provided defines an asynchronous function called `airDropSol`. Let's break down what it does in a paragraph:

The function's purpose is to request an airdrop of 2 SOL (the native cryptocurrency of Solana) to a specific wallet. To accomplish this, it performs the following steps:

First, the function establishes a connection to the Solana Devnet (a development network) using the `Connection` object. This connection allows the function to interact with the Solana blockchain and execute operations such as requesting airdrops.

Next, it creates a wallet (also known as a keypair) by utilizing the private key stored in the `privateKey` variable. A keypair consists of a private key (which must be kept secret) and a corresponding public key (used for identification and transactions).

After generating the keypair, the function initiates an airdrop by invoking the `requestAirdrop` method on the Solana connection object. This method requires the recipient's public key (derived from the keypair) and specifies the amount of SOL to be airdropped. In this case, it requests an airdrop of 2 SOL, which is equivalent to 2 multiplied by the conversion factor `LAMPORTS_PER_SOL`.

During the airdrop process, the function logs a message indicating that an airdrop is being performed.

Finally, the function confirms the transaction by awaiting the `confirmTransaction` method on the Solana connection object. This step ensures that the airdrop transaction is included in a block and confirms its success.

If any errors occur during the process, they will be caught in the `catch` block, and the error message will be logged to the console.