Skip to main content

On this page

Wallet API v2: Quickstart

Overview

The v2 Wallet API allows you to create accounts on EVM compatible networks and the Solana network.

In this quickstart, you will learn how to:

- Create EVM and Solana accounts
- Fund your accounts with testnet tokens using CDP Faucets
- Send a transaction using `viem` for Typescript or `web3` for Python

Prerequisites

Setup all dependencies, export your keys to environment variables, and initialize a new project before you begin.

It is assumed you have:

- Node.js 22.x+ if using Typescript
- Python 3.10+ if using Python
- Created and signed in to an existing CDP account

Once you have setup the prerequisite dependencies, continue reading to create keys to authenticate your requests and initialize a new project.

Create keys

Sign in to the CDP Portal, create a CDP API key and generate a Wallet Secret. Keep these values handy as you will need them in the following steps.

For more information, see the CDP API Keys and Wallet Secret documentation.

Project setup

After creating your keys, initialize a new project and instantiate the CDP client.

Typescript **Python**

Initialize a new Typescript project by running:

```
mkdir cdp-sdk-example && cd cdp-sdk-example && npm init -y && npm pkg set type="module" && touch main.ts && touch .env
```

Add your CDP API key and wallet secret to the `.env` file:

.env

```
CDP_API_KEY_ID=your-api-key-id
CDP_API_KEY_SECRET=your-api-key-secret
CDP_WALLET_SECRET=your-wallet-secret
```

Now, install the CDP SDK and the dotenv packages:

```
npm install @coinbase/cdp-sdk dotenv
```

Finally, in `main.ts`, instantiate the CDP client:

main.ts

```
import { CdpClient } from "@coinbase/cdp-sdk";
import dotenv from "dotenv";

dotenv.config();

// Initialize the CDP client, which automatically loads
// the API Key and Wallet Secret from the environment
// variables.
const cdp = new CdpClient();
```

In this and in the following examples, you can run your code by running:

```
npx tsx main.ts
```

1. Create an account

The v2 Wallet API offers support for both EVM compatible accounts and Solana accounts.

EVM

To create an EVM account, see below:

Typescript   Python

main.ts

```
import { CdpClient } from "@coinbase/cdp-sdk";
import dotenv from "dotenv";

dotenv.config();

const cdp = new CdpClient();
const account = await cdp.evm.createAccount();
console.log(`Created EVM account: ${account.address}`);
```

After running the above snippet, you should see similar output:

```
Created EVM account: 0x3c0D84055994c3062819Ce8730869D0aDeA4c3Bf
```

Tip

You can also create accounts with human-readable names and retrieve them later using the `getOrCreateAccount` method.

See the Managing Accounts guide for more information.

Solana

To create a Solana account, see below:

Typescript **Python**

main.ts

```typescript
import { CdpClient } from "@coinbase/cdp-sdk";
import dotenv from "dotenv";

dotenv.config();

const cdp = new CdpClient();
const account = await cdp.solana.createAccount();
console.log(`Created Solana account: ${account.address}`);
```

After running the above snippet, you should see similar output:

```
Created Solana account:
2XBS6naS1v7pXEg25z43FGHnmEgEad53fmiZ9S6LPgKn
```

2. Fund account with test funds

Accounts do not have funds on creation. We provide a Faucet API to easily fund your account with testnet tokens.

Info

Before you request funds, ensure you read about rate limits when using CDP Faucets.

EVM

Typescript   Python

main.ts

```typescript
import { CdpClient } from "@coinbase/cdp-sdk";
import dotenv from "dotenv";

dotenv.config();

const cdp = new CdpClient();

const account = await cdp.evm.createAccount();
const faucetResponse = await cdp.evm.requestFaucet({
  address: account.address,
  network: "base-sepolia",
  token: "eth"
});
console.log(`Requested funds from ETH faucet:
https://sepolia.basescan.org/tx/${faucetResponse.transactionHash}`
);
```

After running the above, you should see similar output:

```
Requested funds from ETH faucet:
https://sepolia.basescan.org/tx/0x9e93a16f2ca67f35bcb1ea2933f19035
ae1e71ff3100d2abc6a22ce024d085ec
```

Solana

Typescript Python

main.ts

```typescript
import { CdpClient } from "@coinbase/cdp-sdk";
import dotenv from "dotenv";

dotenv.config();

const cdp = new CdpClient();

const account = await cdp.solana.createAccount();

const { signature } = await cdp.solana.requestFaucet({
  address: account.address,
  token: "sol"
});
console.log(`Requested funds from Solana faucet:
https://explorer.solana.com/tx/${signature}`);
```

After running the above, you should see similar output:

```
Requested funds from Solana faucet:
```

```
https://explorer.solana.com/tx/4KEPbhkRLTg2FJNqV5bbUd6zv1TNkksxF9P
DHw2FodrTha3jq2Cojn4hSKtjPWdrZiRDuYp7okRuc1oYvh3JkLuE
```

3. Send a transaction

EVM

Typescript **Python**

You can send transactions using the v2 Wallet API.

Note that in order to wait for transaction confirmation, you will need to have `viem` installed:

```
npm install viem
```

In the example below, we:

1. Create a new EVM account.
2. Request ETH from the faucet.
3. Use the v2 Wallet API to send a transaction.
4. Wait for transaction confirmation.

main.ts

```
import { CdpClient } from "@coinbase/cdp-sdk";
import { http, createPublicClient, parseEther } from "viem";
import { baseSepolia } from "viem/chains";
import dotenv from "dotenv";

dotenv.config();

const cdp = new CdpClient();

const publicClient = createPublicClient({
  chain: baseSepolia,
  transport: http(),
});

// Step 1: Create a new EVM account.
const account = await cdp.evm.createAccount();
console.log("Successfully created EVM account:", account.address);

// Step 2: Request ETH from the faucet.
const { transactionHash: faucetTransactionHash } = await
cdp.evm.requestFaucet({
  address: account.address,
  network: "base-sepolia",
  token: "eth",
});

const faucetTxReceipt = await
publicClient.waitForTransactionReceipt({
  hash: faucetTransactionHash,
});
console.log("Successfully requested ETH from faucet:",
faucetTxReceipt.transactionHash);

// Step 3: Use the v2 Wallet API to send a transaction.
const transactionResult = await cdp.evm.sendTransaction({
  address: account.address,
  transaction: {
    to: "0x0000000000000000000000000000000000000000",
    value: parseEther("0.000001"),
  },
  network: "base-sepolia",
});

// Step 4: Wait for the transaction to be confirmed
const txReceipt = await publicClient.waitForTransactionReceipt({
  hash: transactionResult.transactionHash,
});

console.log(
  `Transaction sent! Link:
https://sepolia.basescan.org/tx/${transactionResult.transactionHas
h}`
);
```

Solana

Typescript **Python**

You can send transactions on Solana using the `@solana/web3.js` v1 library.

```
npm install @solana/web3.js@1
```

In the example below, we:

1. Create a new Solana account.
2. Request SOL from the faucet.

3. Wait for funds to become available.
4. Send the transaction to a specified address.

main.ts

```
import {
  Connection,
  PublicKey,
  SystemProgram,
  Transaction,
} from "@solana/web3.js";
import { CdpClient } from "@coinbase/cdp-sdk";
import dotenv from "dotenv";

dotenv.config();

const cdp = new CdpClient();

const connection = new
Connection("https://api.devnet.solana.com");

async function createAccount() {
  const account = await cdp.solana.createAccount();
  console.log(`Created account: ${account.address}`);
  return account;
}

async function requestFaucet(address: string) {
  await cdp.solana.requestFaucet({
    address,
    token: "sol",
  });
}
```

```
async function waitForBalance(address: string) {
  let balance = 0;
  let attempts = 0;
  const maxAttempts = 30;

  while (balance === 0 && attempts < maxAttempts) {
    balance = await connection.getBalance(new PublicKey(address));
    if (balance === 0) {
      console.log("Waiting for funds...");
      await new Promise(resolve => setTimeout(resolve, 1000));
      attempts++;
    } else {
      console.log("Account funded with", balance / 1e9, "SOL");
    }
  }

  if (balance === 0) {
    throw new Error("Account not funded after multiple attempts");
  }
}

async function sendTransaction(address: string) {
  // Amount of lamports to send (default: 1000 = 0.000001 SOL)
  const lamportsToSend = 1000;
  const fromAddress = new PublicKey(address)
  const toAddress = new
PublicKey("EeVPcnRE1mhcY85wAh3uPJG1uFiTNya9dCJjNUPABXzo");

  const { blockhash } = await connection.getLatestBlockhash();

  const transaction = new Transaction();
  transaction.add(
    SystemProgram.transfer({
      fromPubkey: fromAddress,
      toPubkey: toAddress,
      lamports: lamportsToSend,
   })
  );

  transaction.recentBlockhash = blockhash;
  transaction.feePayer = fromAddress;

  const serializedTx = Buffer.from(
    transaction.serialize({ requireAllSignatures: false })
  ).toString("base64");

  const { signature: txSignature } = await
cdp.solana.signTransaction({
    address,
    transaction: serializedTx,
  });
  const decodedSignedTx = Buffer.from(txSignature, "base64");

  console.log("Sending transaction...");
  const txSendSignature = await
connection.sendRawTransaction(decodedSignedTx);

  const latestBlockhash = await connection.getLatestBlockhash();

  console.log("Waiting for transaction to be confirmed...");
  const confirmation = await connection.confirmTransaction({
    signature: txSendSignature,
    blockhash: latestBlockhash.blockhash,
    lastValidBlockHeight: latestBlockhash.lastValidBlockHeight,
  });

  if (confirmation.value.err) {
```

```
    throw new Error(`Transaction failed:
${confirmation.value.err.toString()}`);
  }

  console.log(`Sent SOL:
https://explorer.solana.com/tx/${txSendSignature}?
cluster=devnet`);
}

async function main() {
  const account = await createAccount();
  await requestFaucet(account.address);
  await waitForBalance(account.address);
  await sendTransaction(account.address);
}

main().catch(console.error)
```

What to read next

- v2 Wallet Accounts: An overview of the types of accounts supported by the v2 Wallet API.
- Using Smart Accounts: A step-by-step guide on how to create and use smart accounts.
- v2 Wallet Security: Learn about the security features of the v2 Wallet API.
- Faucets: Learn more on supported testnet assets and their associated rate limits.

*Last updated on **May 7, 2025***

Get help on Discord

Request a feature