D4D4D4;--ch-t-background: #1E1E1E;--ch-t-lighter-inlineBackground: #1e1e1ee6;--ch-t-editor-background: #1E1E1E;--ch-t-editor-foreground: #D4D4D4;--ch-t-editor-rangeHighlightBackground: #ffffff0b;--ch-t-editor-infoForeground: #3794FF;--ch-t-editor-selectionBackground: #264F78;--ch-t-focusBorder: #007FD4;--ch-t-tab-activeBackground: #1E1E1E;--ch-t-tab-activeForeground: #ffffff;--ch-t-tab-inactiveBackground: #2D2D2D;--ch-t-tab-inactiveForeground: #ffffff80;--ch-t-tab-border: #252526;--ch-t-tab-activeBorder: #1E1E1E;--ch-t-editorGroup-border: #444444;--ch-t-editorGroupHeader-tabsBackground: #252526;--ch-t-editorLineNumber-foreground: #858585;--ch-t-input-background: #3C3C3C;--ch-t-input-foreground: #D4D4D4;--ch-t-icon-foreground: #C5C5C5;--ch-t-sideBar-background: #252526;--ch-t-sideBar-foreground: #D4D4D4;--ch-t-sideBar-border: #252526;--ch-t-list-activeSelectionBackground: #094771;--ch-t-list-activeSelectionForeground: #fffffe;--ch-t-list-hoverBackground: #2A2D2E; }

AI agent with a spending limit for a treasury

This setup is used by DAOs or other organizations that want to utilize AI agents to manage their funds.

This setup uses Safe's allowance module(opens in a new tab). After activating it for a Safe Smart Account, you can set an allowance per token for a spender (the delegator). It can be a one-time allowance, or an allowance that resets after a certain time interval (for example, 100 USDC every day).

You can find an easy to run example for the allowance module in our example repository(opens in a new tab).

You can setup an allowance (spending limit) on a Safe Smart Account with the Safe Wallet(opens in a new tab) interface following this guide(opens in a new tab). Then, your agent can spend the allowance, as described in the last step.

Here are the important code snippets to get you up and running:

Pre-requisites

- A deployed Safe Smart Account
- The Smart Account should hold an amount of the ERC20 token for which the allowance will be given

Set and use a spending limit for the AI agent

Enable the Allowance module on your Safe

When you set a spending limit from Safe Wallet, the allowance module will be enabled automatically. You will use the Safe Protocol Kit . Here is a code example to enable it programmatically:

_16 import Safe from '@safe-global/protocol-kit' _16 import { getAllowanceModuleDeployment } from '@safe-global/safe-modules-deployments' _16 _16 const preExistingSafe = await Safe.init({ _16 provider: RPC_URL, _16 signer: OWNER_1_PRIVATE_KEY, _16 safeAddress: safeAddress _16 }) _16 _16 // Add Module _16 const allowanceModule = getAllowanceModuleDeployment({ network: '11155111' })! _16 const safeTransaction = await preExistingSafe.createEnableModuleTx(_16 allowanceModule.networkAddresses['11155111'] _16) _16 const txResponse = await preExistingSafe.executeTransaction(safeTransaction) _16 console.log(txResponse)

Set spending limit for AI agent

Now you can set a spending limit to your AI agent:

_55 import { getAllowanceModuleDeployment } from '@safe-global/safe-modules-deployments' _55 import Safe from '@safe-global/protocol-kit' _55 import { getAllowanceModuleDeployment } from '@safe-global/safe-modules-deployments' _55 import { OperationType, MetaTransactionData } from '@safe-global/types-kit' _55 _55 const ERC20_TOKEN_ADDRESS = '0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238' _55 const preExistingSafe = await Safe.init({ _55 provider: RPC_URL, _55 signer: OWNER_1_PRIVATE_KEY, _55 safeAddress: safeAddress _55 }) _55 _55 const allowanceModule = getAllowanceModuleDeployment({ network: '11155111' })! _55 _55 const allowanceModuleAddress = allowanceModule.networkAddresses['11155111'] _55 _55 const callData1 = encodeFunctionData({ _55 abi: allowanceModule.abi, _55 functionName: 'addDelegate', _55 args: [AGENT_ADDRESS] _55 }) _55 // agent can spend 1 USDC per day: _55 const callData2 = encodeFunctionData({ _55 abi: allowanceModule.abi, _55 functionName: 'setAllowance', _55 args: [_55 AGENT_ADDRESS, // delegate _55 ERC20_TOKEN_ADDRESS, // token _55 1_000_000, // allowance amount (1 USDC) _55 1_440, // reset time in minutes (1440 mins = 1 day) _55 0 // reset base (fine to set zero) _55] _55 }) _55 _55 const safeTransactionData1: MetaTransactionData = { _55 to: allowanceModuleAddress, _55 value: '0', _55 data: callData1, _55 operation: OperationType.Call _55 } _55 _55 const safeTransactionData2: MetaTransactionData = { _55 to: allowanceModuleAddress, _55 value: '0', _55 data: callData2, _55 operation: OperationType.Call _55 } _55 _55 const safeTransaction = await preExistingSafe.createTransaction({ _55 transactions: [safeTransactionData1, safeTransactionData2], _55 onlyCalls: true _55 }) _55 _55 const txResponse = await preExistingSafe.executeTransaction(safeTransaction) _55 console.log(txResponse)

Let the AI agent use the spending limit

Now your agent has a spending limit, either set programmatically or from Safe Wallet.

Here is how the agent can spend it:

_72 import { _72 createPublicClient, _72 http, _72 encodeFunctionData, _72 zeroAddress, _72 createWalletClient, _72 } from 'viem' _72 import { privateKeyToAccount } from 'viem/accounts' _72 const ERC20_TOKEN_ADDRESS = '0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238' _72 _72 const allowanceModule = getAllowanceModuleDeployment({ network: '11155111' })! _72 _72 const allowanceModuleAddress = allowanceModule.networkAddresses[_72 '11155111' _72] as 0x{string} _72 _72 const publicClient = createPublicClient({ transport: http(RPC_URL!) }) _72 _72 // Read allowance module to get current nonce _72 const allowance = await publicClient.readContract({ _72 address: allowanceModuleAddress, _72 abi: allowanceModule.abi, _72 functionName: 'getTokenAllowance', _72 args: [safeAddress, AGENT_ADDRESS, ERC20_TOKEN_ADDRESS] _72 }) _72 _72 const amount = 1 // You might want to adapt the amount _72 _72 // generate hash _72 const hash = await publicClient.readContract({ _72 address: allowanceModuleAddress, _72 abi: allowanceModule.abi, _72 functionName: 'generateTransferHash', _72 args: [_72 safeAddress, _72 ERC20_TOKEN_ADDRESS, _72 AGENT_ADDRESS, _72 amount, _72 zeroAddress, _72 0, _72 allowance[4] // nonce _72] _72 }) _72 _72 const agentAccount = privateKeyToAccount(_72 AGENT_PRIVATE_KEY as 0x{string} _72) _72 const signature = await agentAccount.sign({ _72 hash: hash as unknown as 0x{string} _72 }) _72 _72 const { request } = await publicClient.simulateContract({ _72 address: allowanceModuleAddress, _72 abi: allowanceModule.abi, _72 functionName: 'executeAllowanceTransfer', _72 args: [_72 safeAddress, _72 ERC20_TOKEN_ADDRESS, _72 AGENT_ADDRESS, _72 amount, _72 zeroAddress, _72 0, _72 AGENT_ADDRESS, _72 signature _72], _72 account: agentAccount _72 }) _72 _72 const walletClient = createWalletClient({ transport: http(RPC_URL!) }) _72 _72 const tx = await walletClient.writeContract(request) _72 console.log(tx) In this example, your agent will get a daily spending limit of 10 USDC.

Next steps

You can find more info in the example repository or in the documentation about the allowance module.

If you have a technical question, feel free to reach out on Stack Exchange(opens in a new tab) with the safe-core tag.

Multiple Agent setup Introduction Was this page helpful?

Report issue