Using the Automate SDK

Using Automate SDK

The SDK is suitable when you need to integrate task creation into your development environment or automated scripts. It's also useful for complex setups that require conditional logic before task submission.

Use theautomate-sdk to easily create a new task:

```

Copy yarn install @gelatonetwork/automate-sdk

```

Typescript Functions are still in private beta, make sure to use the beta version of the automate-sdk to have access to it

Typescript Function

Import the sdk and create task, passing your typescript function CID & arguments:

```

Copy import{ AutomateSDK,Web3Function }from"@gelatonetwork/automate-sdk";

constautomate=newAutomateSDK(chainId,deployer);

const{taskId,tx}=awaitautomate.createBatchExecTask({ name:"Web3Function - Eth Oracle", web3FunctionHash:cid, web3FunctionArgs:{ oracle:oracle.address, currency:"ethereum", }, trigger:{ // Run every minutes type:TriggerType.TIME, interval:60*1000, }, }); awaittx.wait();

```

You can specifycron trigger this way:

```

Copy trigger:{ type:TriggerType.CRON, cron:"0 8 * * *",// Run every day at 8:00 }

```

event trigger like this:

```

Copy trigger:{ type:TriggerType.EVENT, filter:{ // Listen to PriceUpdated event on Oracle contract address:oracle.address, topics:[[oracle.getEventTopic("PriceUpdated")]], }, blockConfirmations:0,// Trigger immediately },

```

Andblock trigger this way:

```

Copy trigger:{ type:TriggerType.BLOCK, }

```

If your task utilizes secrets, you can set them after the task has been created.

```

Copy constweb3Function=newWeb3Function(chainId,deployer);

constsecrets={ API_KEY:"..."// Set your secret environment variables } awaitweb3Function.secrets.set(secrts,taskId);

```

Solidity Function

Repeat the installation step as shown above, then import and instantiate the SDK:

```

Copy constautomate=newAutomateSDK(chainId,signer);

```

UsecreateTask to automate your function calls:

```

Copy interfaceCreateTaskOptions{ name:string;// your task name

// Function to execute execAddress:string;// address of your target smart contract execSelector:string;// function selector to execute on your target smart contract execAbi?:string;// ABI of your target smart contract

// Proxy caller dedicatedMsgSender:boolean;// task will be called via a dedicated msg.sender which you can whitelist (recommended: true)

// Optional: Pre-defined / static target smart contract inputs execData?:string;// exec call data

// Optional: Dynamic target smart contract inputs (using a resolver) resolverAddress?:string;// resolver contract address resolverData?:string;// resolver call data (encoded data with function selector) resolverAbi?:string;// your resolver contract ABI

// Optional: Time based task params interval?:number;// execution interval in seconds startTime?:number;// start timestamp in seconds or 0 to start immediately (default: 0)

// Optional: Single execution task singleExec?:boolean;// task cancels itself after 1 execution if true.

// Optional: Payment params useTreasury?:boolean;// use false if your task is self-paying (default: true) }

constparams:CreateTaskOptions={ name, execAddress, execSelector, interval };

const{taskId,tx}:TaskTransaction=awaitautomate.createTask(params);

```

Examples

Deploy a contract & automate your function call:

```

Copy // Deploying Counter contract constcounterFactory=awaithre.ethers.getContractFactory("Counter"); constcounter=awaitcounterFactory.deploy(GELATO_ADDRESSES[chainId].automate); awaitcounter.deployed();

// Call Counter.increaseCount(42) every 10 minutes const{taskId,tx}:TaskTransaction=awaitautomate.createTask({ execAddress:counter.address, execSelector:counter.interface.getSighash("increaseCount(uint256)"), execData:counter.interface.encodeFunctionData("increaseCount",[42]), execAbi:counter.interface.format("json")asstring, interval:10*60,// execute every 10 minutes name:"Automated counter every 10min", dedicatedMsgSender:true });

```

Use a Checker to automate your function call:

If you need more configurable execution condition and/or dynamic input data, you can create a task using Checker

```

Copy // Prepare Task data to automate constcounter=newContract(COUNTER_ADDRESSES,counterAbi,signer); constresolver=newContract(COUNTER_RESOLVER_ADDRESSES,counterResolverAbi,signer); constselector=counter.interface.getSighash("increaseCount(uint256)"); constresolverData=resolver.interface.getSighash("checker()");

// Create task const{taskId,tx}:TaskTransaction=awaitautomate.createTask({ execAddress:counter.address, execSelector:selector, resolverAddress:resolver.address, resolverData:resolverData, name:"Automated counter using resolver", dedicatedMsgSender:true });

```

Automated Transaction

TheCreateTaskOptions interface is used for configuring Automated Transactions with the same structure and options as defined above.

The only difference is you need to configure your automated transaction without the need for a checker function.

Example :

```

Copy interfaceCreateTaskOptions{ name:string;// Name your task execAddress:string;// Address of your target smart contract execSelector:string;// Function selector to call on your target smart contract execAbi?:string;// ABI of your target smart contract (optional) execData?:string;// Call data for the function execution (optional) }

```

Single Execution Task

If you want to have Gelato call your function only once. If so, setsingleExec flag totrue when callingcreateTask .

```

Copy const{taskId,tx}:TaskTransaction=awaitautomate.createTask({ execAddress:counter.address, execSelector:selector, resolverAddress:counter.address, resolverData:resolverData, dedicatedMsgSender:true, name:"Automated counter using resolver", dedicatedMsgSender:true, singleExec:true });

```

Previous Using a Smart Contract Next Security Considerations Last updated3 months ago On this page *Using Automate SDK * Typescript Function * Solidity Function * Automated Transaction * Single Execution Task