Create Automation-Compatible Contracts

Learn how to make smart contracts that are compatible withAutomation.

Considerations and Best Practices

Before you deploy contracts to use with Chainlink Automation, read theBest Practices guide. These best practices are important for using Chainlink Automation securely and reliably. You can also read more about the Chainlink Automation architecturehere .

Automation compatible contracts

A contract is Automation-compatible when it follows a specified interface that allows the Chainlink Automation Network to determine if, when, and how the contract should be automated.

The interface you use will depend on the type of trigger you want to use:

- If you want a log event to trigger your upkeep, use theILogAutomation interface.
- If you want to use onchain state in a custom calculation to trigger your upkeep, useAutomationCompatibleInterface interface.
- If you want to call a function just based on time, you don't need an interface. Consider instead using atime-based upkeep .
- If you want to use Automation with Data Streams, useStreamsLookupCompatibleInterface interface.

You can learn more about these interfaceshere .

Example Automation-compatible contract using custom logic trigger

Custom logic Automation compatible contracts must meet the following requirements:

- ImportAutomationCompatible.sol. You can refer to theChainlink Contracts on GitHub to find the latest version.
- Use theAutomationCompatibleInterface from the library to ensure yourcheckUpkeepandperformUpkeepfunction definitions match the definitions expected by the Chainlink Automation Network.
- Include acheckUpkeepfunction that contains the logic that will be executed offchain to see ifperformUpkeepshould be executed.checkUpkeepcan use onchain data and a specifiedcheckDataparameter to perform complex calculations offchain and then send the result toperformUpkeepasperformData.
- Include aperformUpkeepfunction that will be executed onchain whencheckUpkeepreturnstrue.

Use these elements to create a compatible contract that will automatically increment a counter after everyupdateIntervalseconds. After you register the contract as an upkeep, the Chainlink Automation Network frequently simulates yourcheckUpkeepoffchain to determine if theupdateIntervaltime has passed since the last increment (timestamp). WhencheckUpkeepreturns true, the Chainlink Automation Network callsperformUpkeeponchain and increments the counter. This cycle repeats until the upkeep is cancelled or runs out of funding.

// SPDX-License-Identifier: MITpragmasolidity^0.8.7;// AutomationCompatible.sol imports the functions from both ./AutomationBase.sol and// ./interfaces/AutomationCompatibleInterface.solimport{AutomationCompatibleInterface}from"@chainlink/contracts/src/v0.8/automation/AutomationCompatible.sol";/ *** @dev Example contract, use the Forwarder as needed for additional security. * * @notice important to implement {AutomationCompatibleInterface}** *// * THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION. */contractCounterisAutomationCompatibleInterface{/ * Public counter variable* /uint256publiccounter;/ ** Use an interval in seconds and a timestamp to slow execution of Upkeep /uint256publicimmutable interval;uint256publiclastTimeStamp;constructor(uint256updateInterval) {interval=updateInterval;lastTimeStamp=block.timestamp;counter=0;}functioncheckUpkeep(bytescalldata/* checkData /)externalviewoverridereturns(boolupkeepNeeded,bytesmemory/* performData /){upkeepNeeded=(block.timestamp-lastTimeStamp)>interval;// We don't use the checkData in this example. The checkData is defined when the Upkeep was registered.}functionperformUpkeep(bytescalldata/* performData */)externaloverride{if((block.timestamp-lastTimeStamp)>interval){lastTimeStamp=block.timestamp;counter=counter+1;}// We don't use the performData in this example. The performData is generated by the Automation Node's call to your checkUpkeep function}} Open in Remix What is Remix? Compile and deploy your own Automation Counter onto asupported Testnet .

1. In the Remix example, select the compile tab on the left and press the compile button. Make sure that your contract compiles without any errors. Note that the Warning messages in this example are acceptable and will not block the deployment.
2. Select theDeploytab and deploy theCountersmart contract in theinjected web3environment. When deploying the contract, specify theupdateIntervalvalue. For this example, set a short interval of 60. This is the interval at which theperformUpkeepfunction will be called.
3. After deployment is complete, copy the address of the deployed contract. This address is required to register your upkeep in theAutomation UI . The example in this document usescustom logic automation.

To see more complex examples, go to theQuick Starts page.

Now register yourupkeep .

Vyper example

Note on arrays

Make sure the checkdata array size is correct. Vyper does not support dynamic arrays.

You can find aKeepersConsumerexamplehere . Read theapeworx-starter-kitREADME to learn how to run the example.