Deployer

op-deployer simplifies the process of deploying the OP Stack. It works similarly to Terraform(opens in a new tab) . Like Terraform, you define a declarative config file called an "intent," then run a command to apply the intent to your chain.op-deployer will compare the state of your chain against the intent, and make whatever changes are necessary for them to match. In its current state, it is intended to deploy new standard chains that utilize the Superchain wide contracts.

Installation

The recommended way to installop-deployer is to download the latest release from the monorepo's release page(opens in a new tab) . To install a release, download the binary for your platform then extract it somewhere on yourPATH . The rest of this tutorial will assume that you have installedop-deployer using this method.

To alternatively install from source, follow the steps below. You will need to install the Go toolchain andjust as prerequisites.

Clone the monorepo

:

Run the following command to clone the monorepo:

git

clone

https://github.com/ethereum-optimism/optimism.git

Build the binary

:

Run the following commands to build the binary:

cd

op-deployer just

build The binary will be in thebin directory.

Deployment usage

The base use case forop-deployer is deploying new OP Chains. This process is broken down into three steps:

init

: configure your chain

To get started withop-deployer , create an intent file that defines your desired chain configuration. Use the built-inop-deployer utility to generate this file:

./bin/op-deployer init --l1-chain-id 11155111 --l2-chain-ids--workdir .deployer This command will create a directory called.deployer in your current working directory containing the intent file and an emptystate.json file.state.json is populated with the results of your deployment, and never needs to be edited directly.

Your intent file will need to be modified to your parameters, but it will initially look something like this:

⚠ Do not use the default addresses in the intent for a production chain! They are generated from thetest... junk mnemonic.Any funds they hold will be stolen on a live chain. deploymentStrategy =

"live"

Deploying a chain to a live network i.e. Sepolia

l1ChainID

11155111

The chain ID of the L1 chain you'll be deploying to

fundDevAccounts

true

Whether or not to fund dev accounts using the test... junk mnemonic on L2.

l1ContractsLocator

"tag://op-contracts/v1.6.0"

L1 smart contracts versions

l2ContractsLocator

"tag://op-contracts/v1.7.0-beta.1+l2-contracts"

L2 smart contracts versions

Delete this table if you are using the shared Superchain contracts on the L1

If you are deploying your own SuperchainConfig and ProtocolVersions contracts, fill in these details

[superchainRoles] proxyAdminOwner =

"0xb9cdf788704088a4c0191d045c151fcbe2db14a4" protocolVersionsOwner =

"0x85d646ed26c3f46400ede51236d8d7528196849b" guardian =

"0x8c7e4a51acb17719d225bd17598b8a94b46c8767"

List of L2s to deploy. op-deployer can deploy multiple L2s at once

[[chains]]

Your chain's ID, encoded as a 32-byte hex string

id

"0x0000000000000000000000000000000000000000000000000000000000003039"

Update the fee recipient contract

baseFeeVaultRecipient

"0x0000000000000000000000000000000000000000" l1FeeVaultRecipient =

"0x0000000000000000000000000000000000000000" sequencerFeeVaultRecipient =

"0x0000000000000000000000000000000000000000" eip1559Denominator =

50 eip1559Elasticity =

6

Various ownership roles for your chain. When you use op-deployer init, these roles are generated using the

test... junk mnemonic. You should replace these with your own addresses for production chains.

[chains.roles] l1ProxyAdminOwner =

"0x1a66b55a4f0139c32eddf4f8c60463afc3832e76" l2ProxyAdminOwner =

"0x7759a8a43aa6a7ee9434ddb597beed64180c40fd" systemConfigOwner =

"0x8e35d9523a0c4c9ac537d254079c2398c6f3b35f" unsafeBlockSigner =

"0xbb19dce4ce51f353a98dbab31b5fa3bc80dc7769" batcher =

"0x0e9c62712ab826e06b16b2236ce542f711eaffaf" proposer =

"0x86dfafe0689e20685f7872e0cb264868454627bc" challenger =

"0xf1658da627dd0738c555f9572f658617511c49d5" By default,op-deployer will fill in all other configuration variables with those that match thestandard configuration(opens in a new tab). You can override these default settings by adding them to your intent file using the table below:

[globalDeployOverrides] l2BlockTime =

1

1s L2blockTime is also standard, op-deployer defaults to 2s

You can also do chain by chain configurations in thechains table.

apply

: deploy your chain

Hardware wallets are not supported, but you can use ephemeral hot wallets since this deployer key has no privileges. Now that you've created your intent file, you can apply it to your chain to deploy the L1 smart contracts:

op-deployer apply --workdir .deployer --l1-rpc-url --private-key This command will deploy the OP Stack to L1. It will deploy all L2s specified in the intent file. Superchain configuration will be set to the Superchain-wide defaults - i.e., your chain will be opted into theSuperchain pause(opens in a new tab) and will use the sameprotocol versions(opens in a new tab) address as other chains on the Superchain.

inspect

: generate genesis files and chain information

To add your chain to theSuperchain Registry(opens in a new tab) you will need to provide the chain artifacts. To get these chain artifacts, you will need to write the output of these commands to new files. Inspect thestate.json file by navigating to your working directory. With the contracts deployed, generate the genesis and rollup configuration files by running the following commands:"

cd .deployer op-deployer --workdir .deployer inspect genesis > genesis.json op-deployer --workdir .deployer inspect rollup > rollup.json Now that you have yourgenesis.json androllup.json you can spin up a node on your network. You can also use the following inspect subcommands to get additional data:

op-deployer --workdir .deployer inspect l1 # outputs all L1 contract addresses for an L2 chain op-deployer --workdir .deployer inspect deploy-config # outputs the deploy config for an L2 chain op-deployer --workdir .deployer inspect l2-semvers # outputs the semvers for all L2 chains

Bootstrap usage

You can also useop-deployer to deploy the contracts needed to run theinit ...apply flow on new chains. This process, called 'bootstrapping,' is useful when you want to useop-deployer with L3s, new testnets, or other custom settlement chains.

OPCM bootstrap

To deploy OPCM to a new chain, run the following command:

```
op-deployer

bootstrap

opcm \ --l1-rpc-url

< mainnet-rpc-ur l

    \ --private-key

< deployer-private-ke y

    \ --artifacts-locator

tag://op-contracts/v1.6.0
```

Next steps

- For more details, check out the tool and documentation in theop-deployer repository(opens in a new tab)
- .
- For more information on OP Contracts Manager, refer to theOPCM documentation
- .

op-deployer proxyd