Deploy a Solana Program with the CLI

Developers can deploy on-chainprograms (often called smart contracts elsewhere) with the Solana tools.

To learn about developing and executing programs on Solana, start with theintro to Solana programs and then dig into the details ofdeveloping on-chain programs .

To deploy a program, use the Solana tools to interact with the on-chain loader to:

- Initialize a program account
- Upload the program's shared object (the program binary.so
-) to the program
- account's data buffer
- (optional) Verify the uploaded program
- Finalize the program by marking the program account executable.

Once deployed, anyone can execute the program by sending transactions that reference it to the cluster.

How to deploy a program

To deploy a program, you will need the location of the program's shared object (the program binary.so):

solana program deploy < PROGRAM_FILEPATH

> Successful deployment will return the program id of the deployed program, for example:

Program Id: 3KS2k14CmtnuVv2fvYcvdrNgC94Y11WETBpMUGgXyWZL Specify the keypair in the deploy command to deploy to a specific program id:

solana program deploy --program-id < KEYPAIR_FILEPATH

< PROGRAM_FILEPATH

> If the program id is not specified on the command line the tools will first look for a keypair file matching the, or internally generate a new keypair.

A matching program keypair file is in the same directory as the program's shared object, and named-keypair.json. Matching program keypairs are generated automatically by the program build tools:

./path-to-program/program.so ./path-to-program/program-keypair.json

Showing a program account

To get information about a deployed program:

solana program show < ACCOUNT_ADDRESS

> An example output looks like:

Program Id: 3KS2k14CmtnuVv2fvYcvdrNgC94Y11WETBpMUGgXyWZL Owner: BPFLoaderUpgradeab1e11111111111111111111111 ProgramData Address: EHsACWBhgmw8iq5dmUZzTA1esRqcTognhKNHUkPi4q4g Authority: FwoGJNUaJN2zfVEex9BB11Dqb3NJKy3e9oY3KTh9XzCU Last Deployed In Slot: 63890568 Data Length: 5216

(0x1460) bytes * Program Id * is the address that can be referenced in an instruction'sprogram_id * field when invoking a program. * Owner * : The loader this program was deployed with. * ProgramData Address * is the account associated with the program account that * holds the program's data (shared object). * Authority * is the program's upgrade authority. * Last Deployed In Slot * is the slot in which the program was last deployed. * Data Length * is the size of the space reserved for deployments. The actual * space used by the currently deployed program may be less.

Redeploy a Solana program

A program can be redeployed to the same address to facilitate rapid development, bug fixes, or upgrades. If a program id is not provided, the program will be deployed to the default address at-keypair.json . This default keypair is generated during the first program compilation.

The command looks the same as the deployment command:

solana program deploy < PROGRAM_FILEPATH

By default, programs are deployed to accounts that match the size of the original program file. But, if the redeployed program is larger, the redeployment will fail. To avoid this, specify amax_len that is at least the size (in bytes) that the program is expected to become (plus some wiggle room).

solana program deploy --max-len 200000

< PROGRAM_FILEPATH

Extend a program

If a program has already been deployed, and a redeployment goes beyond themax_len of the account, it's possible to extend the program to fit the larger redeployment:

solana program extend < PROGRAM_ID

< ADDITIONAL_BYTES

Resuming a failed deploy

If program deployment fails, there will be a hanging intermediate buffer account that contains a non-zero balance. In order to recoup that balance you may resume a failed deployment by providing the same intermediate buffer to a new call todeploy .

Deployment failures will print an error message specifying the seed phrase needed to recover the generated intermediate buffer's keypair:

========== Recover the intermediate account's ephemeral keypair file with solana-keygen recover and the following 12-word seed phrase: ========== valley flat great hockey share token excess clever benefit traffic avocado athlete ========== To resume a deploy, pass the recovered keypair as the [BUFFER_SIGNER] to solana program deploy or solana program write-buffer'. Or to recover the account's lamports, pass it as the [BUFFER_ACCOUNT_ADDRESS] argument tosolana program drain`. ========== To recover the keypair:

solana-keygen recover -o < KEYPAIR_PATH

When asked, enter the 12-word seed phrase.

Then issue a newdeploy command and specify the buffer:

solana program deploy --buffer < KEYPAIR_PATH

< PROGRAM_FILEPATH

Closing program and buffer accounts, and reclaiming their lamports

Both program and buffer accounts can be closed and their lamport balances transferred to a recipient's account.

If deployment fails there will be a left-over buffer account that holds lamports. The buffer account can either be used toresume a deploy or closed.

The program or buffer account's authority must be present to close an account, to list all the open program or buffer accounts that match the default authority:

solana program show --programs solana program show --buffers To specify a different authority:

solana program show --programs --buffer-authority < AUTHORITY_ADDRESS

solana program show --buffers --buffer-authority < AUTHORITY_ADDRESS

To close a single account:

solana program close < BADDRESS

To close a single account and specify a different authority than the default:

solana program close < ADDRESS

--buffer-authority < KEYPAIR_FILEPATH

To close a single account and specify a different recipient than the default:

solana program close < ADDRESS

--recipient < RECIPIENT_ADDRESS

To close all the buffer accounts associated with the current authority:

solana program close --buffers To show all buffer accounts regardless of the authority

solana program show --buffers --all

Set a program's upgrade authority

The program's upgrade authority must be present to deploy a program. If no authority is specified during program deployment, the default keypair is used as the authority. This is why redeploying a program in the steps above didn't require an authority to be explicitly specified.

The authority can be specified during deployment:

solana program deploy --upgrade-authority < UPGRADE_AUTHORITY_SIGNER

< PROGRAM_FILEPATH

Or after deployment and using the default keypair as the current authority:

solana program set-upgrade-authority < PROGRAM_ADDRESS

--new-upgrade-authority < NEW_UPGRADE_AUTHORITY

Or after deployment and specifying the current authority:

solana program set-upgrade-authority < PROGRAM_ADDRESS

--upgrade-authority < UPGRADE_AUTHORITY_SIGNER

--new-upgrade-authority < NEW_UPGRADE_AUTHORITY

By default,set-upgrade-authority requires a signature from the new authority. This behavior prevents a developer from giving upgrade authority to a key that they do not have access to. The--skip-new-upgrade-authority-signer-check option relaxes the signer check. This can be useful for situations where the new upgrade authority is an offline signer or a multisig.

Immutable programs

A program can be marked immutable, which prevents all further redeployments, by specifying the--final flag during deployment:

solana program deploy < PROGRAM_FILEPATH

--final Or anytime after:

solana program set-upgrade-authority < PROGRAM_ADDRESS

--final

Dumping a program to a file

The deployed program may be dumped back to a local file:

solana program dump < ACCOUNT_ADDRESS

< OUTPUT_FILEPATH

The dumped file will be in the same as what was deployed, so in the case of a shared object (the program binary.so), the dumped file will be a fully functional shared object. Note that thedump command dumps the entire data space, which means the output file will have trailing zeros after the shared object's data up tomax_len . Sometimes it is useful to dump and compare a program to ensure it matches a known program binary. The dumped file can be zero-truncated, hashed, and compared to the hash of the original program file.

solana dump < ACCOUNT_ADDRESS

dump.so cp original.so extended.so truncate -r dump.so extended.so sha256sum extended.so dump.so

Using an intermediary Buffer account

Instead of deploying directly to the program account, the program can be written to an intermediary buffer account. Intermediary accounts can be useful for things like multi-entity governed programs where the governing members first verify the intermediary buffer contents and then vote to allow an upgrade using it.

solana program write-buffer < PROGRAM_FILEPATH

Buffer accounts are managed by an authority. To create a buffer and specify a different authority than the default:

solana program write-buffer < PROGRAM_FILEPATH

--buffer-authority < BUFFER_AUTHORITY_SIGNER

Only the buffer authority may write to the buffer, so the--buffer-authority above must be asigner , and not an address. This requirement limits usage with offline signers. To use an offline address as a buffer authority, the buffer account must be initialized and written with an online keypair, and then the buffer authority must be assigned usingsolana program set-buffer-authority :

solana program set-buffer-authority < BUFFER_ADDRESS

--new-buffer-authority < NEW_BUFFER_AUTHORITY

Unlike program accounts buffer accounts cannot be marked immutable, so they don't support the--final option.

The buffer account, once entirely written, can be passed todeploy to deploy the program:

solana program deploy --program-id < PROGRAM_ADDRESS

--buffer < BUFFER_ADDRESS

Note, the buffer's authority must match the program's upgrade authority. During deployment, the buffer account's contents are copied into the program-data account and the buffer account is set to zero. The lamports from the buffer account are refunded to a spill account.

Buffers also supportshow anddump just like programs do.

Upgrading program using offline signer as authority

Some security models require separating the signing process from the transaction broadcast, such that the signing keys can be completely disconnected from any network, also known asoffline signing .

This section describes how a program developer can use offline signing to upgrade their program, unlike theprevious section , which assumes the machine is connected to the internet, aka online signing.

Note that only theupgrade command can be performed in offline mode. The initial program deploymentmust be performed from an online machine, and only subsequent program upgrades can leverage offline signing.

Assuming your program has been deployed and its upgrade authority has been changed to an offline signer, a typical setup would consist of 2 different signers:

- online signer (fee payer for uploading program buffer and upgrading program)
- offline signer (program upgrade authority)

The general process is as follows:

1. (online) create buffer and write new program to it
2. (online) set buffer authority to offline signer
3. (optional, online) verify the buffer's on-chain contents
4. (offline) sign a transaction to upgrade the program
5. (online) use this signature to broadcast the upgrade transaction

(1) (use online machine) create buffer

solana program write-buffer < PROGRAM_FILEPATH

(2) (use online machine) set buffer authority to offline signer

solana program set-buffer-authority < BUFFER_PUBKEY

--new-buffer-authority < OFFLINE_SIGNER_PUBKEY

(3) (optional) You may verify that the uploaded program matches the built binary. Seedumping a program to a file for more information and details.

(4) (use offline machine) get a signature for your intent to upgrade program

solana program upgrade < BUFFER_PUBKEY

< PROGRAM_ID

--sign-only --fee-payer < ONLINE_SIGNER_PUBKEY

--upgrade-authority < OFFLINE_SIGNER

--blockhash < VALUE

(5) (use online machine) use this signature to build and broadcast the upgrade transaction on-chain

solana program upgrade < BUFFER_PUBKEY

< PROGRAM_ID

--fee-payer < ONLINE_SIGNER

--upgrade-authority < OFFLINE_SIGNER_PUBKEY

--blockhash < VALUE

--signer < OFFLINE_SIGNER_PUBKEY

: < OFFLINE_SIGNER_SIGNATURE

Note:

- typically, the output of the previous command(s) will contain some values useful in subsequent commands, e.g.--program-id
- ,--buffer
- ,--signer
- you need to specify matching (or corresponding) values for params with same names (--fee-payer
- ,--program-id
- ,--upgrade-authority
- ,--buffer
- ,--blockhash
-) in offline/online modes
- you should pre-fill every value except forblockhash
- ahead of time, and once you are ready to act - you'll need to
- look up a recentblockhash
- and paste it in to generate the offline transaction signature. Theblockhash
- expires
- after ~60 seconds. If you didn't make it in time - just get another fresh hash and repeat until you succeed, or
- consider usingdurable transaction nonces
- . Previous Solana CLI: Staking Next Solana CLI: Durable Transaction Nonces