Deploy do token Editora KONKIN

Preparações

Antes de dar deploy é necessário se atentar ou alterar algumas coisas no smart contract

Contas para as taxas da Editora e Liquidez

A conta que vai receber as taxas da editora e a conta que vai receber as taxas para liquidez precisam ser informadas em:

```
address EDITOR_ADDRESS;
```

```
address LIQUIDITY_ADDRESS;
```

> Atente que o endereço da conta precisa estar *checksummed*. O endereço
>
> `0x82ee32f3c8ce259e725e2f2dcac2738016a80d11`
>
> quando ***checksummed*** é:
>
> `0x82EE32F3c8CE259e725E2f2dCAc2738016a80d11`
>
> Desse modo, é possível informar apenas a conta, por exemplo:
>
> ```
> address EDITOR_ADDRESS = 0x82EE32F3c8CE259e725E2f2dCAc2738016a80d11;
> ```
>
> ```
> address LIQUIDITY_ADDRESS =  outro_endereço_checksummed;
> ```

Nome e símbolo

O nome do token pode ser alterado em

```
string private _name;
```

O símbolo do token pode ser alterado em

```
string private _symbol;
```

O nome e o símbolo é privado ao smart contract, mas pode ser acessado fora do smart contract pelas funções `name()` e `symbol()`

Total Supply, número de decimais e limite de tokens por transação

O total supply pode ser alterado em

```
uint256 private _tTotal;
```

`_tTotal` é o total de **tokens**, `_rTotal` é o total de **reflections**, nesse último somente o smart contract realiza alterações. O padrão para o total de tokens é `19501019 * 10**8`, onde '`10**8`' é o número de decimais.

O total supply e os número de decimais são acessados fora do contrato pelas funções `totalSupply()` e `decimals()`.

A função `decimals()` informa a quantidade de decimais para auxiliar na visualização da quantidade de tokens na metamask ou pancakeswap, por exemplo. Isso porque na EVM / solidity não há *floats* (números racionais).

O número de decimais pode ser alterado no `_tTotal` e em seguida na função `decimals()`

O padrão da função `decimals()` é:

```
function decimals() public pure returns (uint8) {
    return 8;
}
```

O limite de tokens por transação pode ser alterado em

```
uint256 _maxTxAmount;
```

O padrão é 50%: `(19501019 / 2) * 10**8`.

O limite pode ser alterado para alguma outra porcentagem após o deploy com a função `setMaxTxPercent()`

Entendendo o `constructor()`

O que está dentro do bloco do construtor do token é executado apenas uma vez no momento do deploy, alguns detalhes precisam de atenção.

Dono do smart contract

O deployer do token vai receber o total de tokens no momento do deploy. A conta do dono do token é excluída das taxas, dessa forma o dono do token pode transferir e distribuir a quantidade de tokens como julgar melhor.

`_rOwned[conta]` significa a quantidade de *reflections* que determinada conta possui.

Router da pancakeswap v2

A pancakeswap possui um router na mainnet e um router na testnet. Dependendo da rede do deploy é necessário alterar o router no smart contract.

Há alguns comentários com os endereços do router da pancakeswap:

- o router da mainnet é `0x10ED43C718714eb63d5aA57B78B54704E256024E`
- o router da testnet é `0xD99D1c33F9fC3444f8101754aBC46c52416550D1`

Escolhida a rede, basta substituir o endereço em

```
IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(aqui_vai_o_endereço_do_router);
```

O router da pancake no smart contract, por padrão, é o da ***testnet***.

Exclusão das taxas e dividendos

A conta do dono do token no momento do deploy é excluída das taxas, mas ainda continua a receber *reflections*.

As contas de burn, da editora e liquidez são excluídas tanto das taxas, quanto dos dividendos.

Compilação

- Compilador solidity: versão 0.8.13 ou alguma versão 0.8.x estável mais recente
- Otimização na compilação: é possível usar, o contrato vai ficar menor em tamanho e o compilador vai tentar reduzir ao máximo o custo de gas, valores altos podem ocasionar bugs no smart contract. O valor padrão para otimização é 200

Deploy

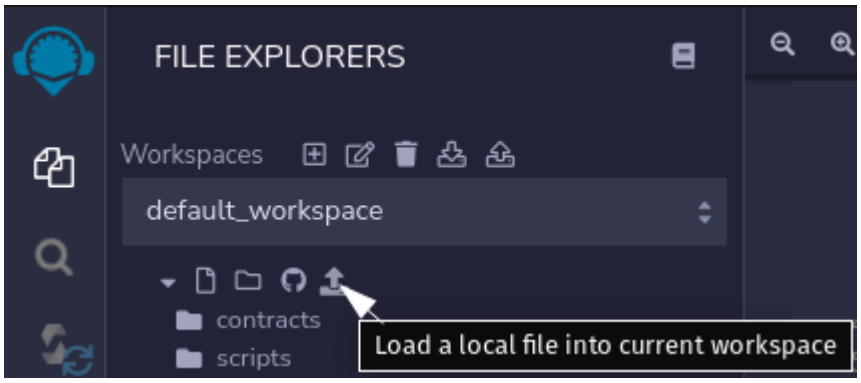
Após verificar se está tudo correto com as contas que as taxas serão recebidas, o nome e símbolo do token, o total supply e limite de transação, o router da pancakeswap para a rede de deploy, só resta realizar o deploy do token.

Por questões de facilidade de uso, vou exemplificar o deploy usando o Remix IDE

{clearpage}

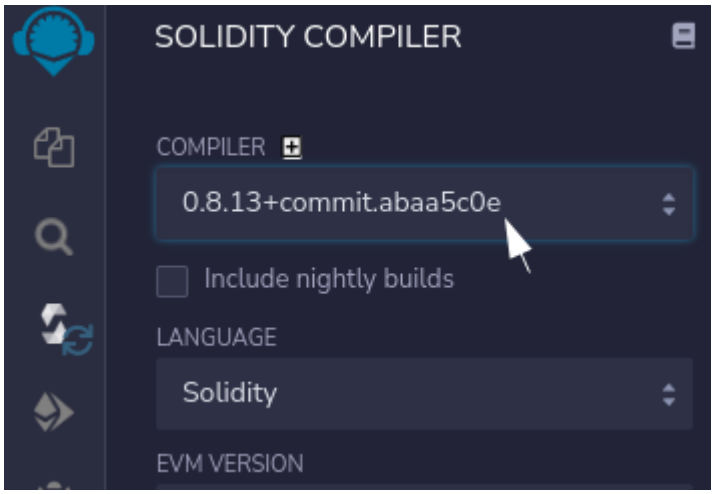
1. Importando o smart contract no remix

Ao acessar o remix IDE, é possível importar um arquivo local no workspace atual:

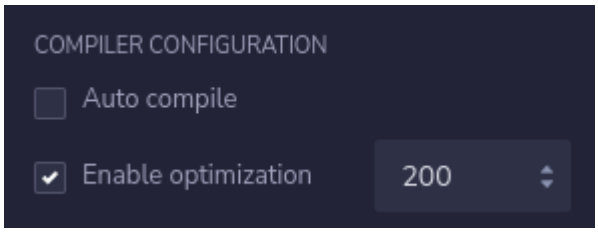


2. Compilando

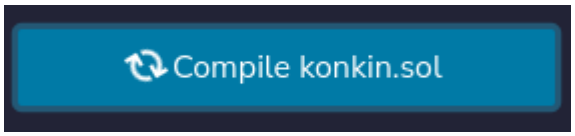
Para compilar o smart contract é preciso selecionar a versão do compilador `0.8.13` ou mais recente:



Também é possível especificar a otimização:



Ao clicar em *Compile konkin.sol*



o indicador na aba do compilador deve ficar assim:

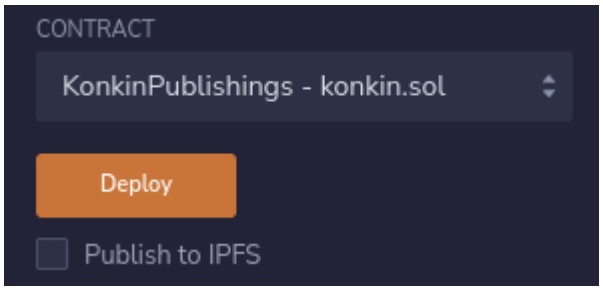
3. Conectando à rede BSC

Com o contrato compilado falta apenas mais algumas etapas para finalmente fazer o deploy. Na aba de ***Deploy & Run Transactions***

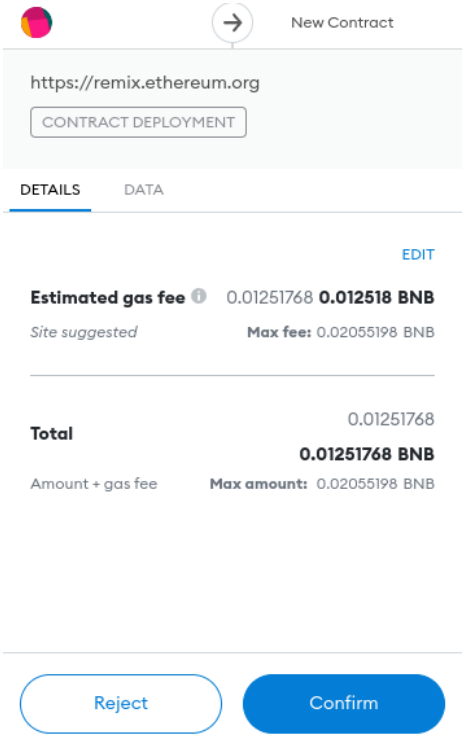
- O *environment* **Injected Web3** precisa estar selecionado no remix:

  - Com o *Injected Web3* selecionado, vai ser solicitado que uma conta seja conectada ao remix, pode ser usado a metamask para conectar uma conta, por exemplo.
  - A conta conectada precisa ter ao menos o saldo para pagar a gas fee para deploy do token

- Agora é só selecionar o contrato correto e clicar em *Deploy*:



– Será solicitado a autorização na sua carteira e provavelmente será estimado quanto de gasfee será necessário para lançar o token



Feito o deploy é só guardar o endereço do smart contract, é possível copiar no painel lateral do remimx

ou no console:

em `logs[0].from`