Why solana?

Until now, we've gone through the following -

1. What are blockchains, how do they work under the hood
2. Public and Private keys, how you can use them to `sign` transactions that miners use to `verify` and credit/debit balances

In today's class, we'll understand about one of the biggest use-case that blockchains like Solana/ETH solve for - Programs/Smart contracts.

Programs/Smart contracts

ETH was one of the first blockchains to introduce the `concept` of decentralized `state` / `programs`. These are popularly known as `smart contracts` on the ETH blockchain.

▼ Here is a simple ETH smart contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Counter {
    uint public count;

    // Constructor to initialize count
    constructor() {
        count = 0;
    }

    // Function to increment the count
```

```
    }
    // Function to decrement the count
    function decrement() public {
        require(count > 0, "Count cannot be negative");
        count -= 1;
    }

    // Function to get the current count
    function getCount() public view returns (uint) {
        return count;
    }
}
```


▼ Here is a simple Node.js HTTP server that does something similar

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware to parse JSON bodies
app.use(express.json());

// Initialize count
let count = 0;

// Route to increment the count
app.post('/increment', (req, res) => {
  count += 1;
  res.json({ count });
});

// Route to decrement the count
app.post('/decrement', (req, res) => {
  if (count > 0) {
    count -= 1;
    res.json({ count });
  } else {
    res.status(400).json({ error: 'Count cannot be negative' });
  }
});

// Route to get the current count
```

```
app.get('/count', (req, res) => {
  res.json({ count });
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```


HTTP Servers are deployed on cloud providers like `GCP, Azure`

Smart contracts/programs are deployed on the `blockchain`

The way solana programs work is significantly different from other blockchains. Lets understand how.

Accounts on Solana

Accounts

On the Solana blockchain, an "account" is a fundamental data structure used to store various types of information.

1. **Data Storage**: Accounts on Solana are used to store data required by programs (smart contracts) or to maintain state
2. **Lamports**: Accounts hold a balance of Solana's native cryptocurrency, lamports. Lamports are used to pay for transaction fees and to rent the space that the account occupies on the blockchain.

3. **Programs:** On Solana, programs are special accounts that contain executable code. These accounts are distinct from regular data accounts in that they are designed to be executed by the blockchain when triggered by a transaction.

Account with `data` and `lamports` but no data -
https://explorer.solana.com/address/4GQsAP5jYi5ysGF1GEnWiV3zJHZLRcLWhLCSuim6aAkL

Account with `lamports` but no data -
https://solscan.io/account/Eg4F6LW8DD3SvFLLigYJBFvRnXSBiLZYYJ3KEePDL95Q

Program
https://solscan.io/account/TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA

Install solana cli

You can install the solana cli locally by running the following command

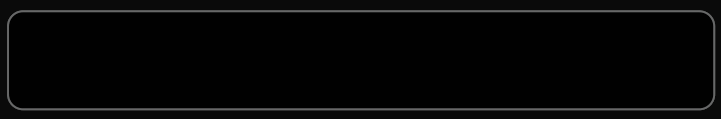
```
sh -c "$(curl -sSfL https://release.anza.xyz/stable/install)"
```

For Windows people -
https://github.com/solana-labs/solana/releases

Unzip and you should see all the .exe files

Web2 Data model

In the web2 world, you store `data` in SQL/NoSQL databases. Here is an example of how you might create tables for a `token balance` app



Data model on Solana

Solana stores all the data of the `same app` / `same program` in various accounts.

Transaction vs Instruction

Transactions

A transaction in Solana is a bundle that includes one or more instructions. Transactions are used to submit operations or changes to the network. They can be simple, such as transferring SOL between accounts, or complex

Instructions

The core operations that the transaction will execute.

> There are more concepts like `recentBlockchash` and `signers` , `writeable` that we we will eventually get to.

How to create an account with some data

```
const solanaWeb3 = require('@solana/web3.js');
const fs = require("fs")
const { Keypair, Connection, SystemProgram, Transaction, sendAndConfirmTrc

// Connect to Solana devnet
const connection = new Connection(solanaWeb3.clusterApiUrl('devnet'), 'confi

// Generate a new keypair for the data account
const dataAccount = Keypair.generate();
const payer = Keypair.fromSecretKey(new Uint8Array(JSON.parse(fs.readFileS

async function createAccount() {
  // Create a transaction to create and fund the account
  const tx = new Transaction().add(
    SystemProgram.createAccount({
      fromPubkey: payer.publicKey,
      newAccountPubkey: dataAccount.publicKey,
      lamports: await connection.getMinimumBalanceForRentExemption(100C
      space: 1000, // Space in bytes to allocate for data
      programId: SystemProgram.programId,
    })
  );

  // Send the transaction to the network
  const txId = await sendAndConfirmTransaction(connection, tx, [payer, dataA

  console.log(`Created account with transaction ID: ${txId}`);
```

```
}
createAccount();
```


Creating a token

Creating `your own token` (100x coin lets say) requires understanding the `Token Program` that is written by the engineers at Solana - https://github.com/solana-labs/solana-program-library

Specifically, the way to create a `token` requires you to

1. Create a token mint
2. Create an `associated token account` for this mint and for a specific user
3. Mint tokens to that user.

Token mint

It's like a `bank` that has the athority to create more coins. It can also have the authority to `freeze coins` .

Associated token account

Before you can ask other people to send you a token, you need to create an `associated token account` for that token and your public key

Reference - https://spl.solana.com/token

Programs, Accounts and the Token Program 1 of 11

- Create a new cli wallet

```
solana-keygen new
```

- Set the RPC url

```
solana config set --url https://api.devnet.solana.com
```

- Airdrop yourself some SOL

```
solana airdrop 1
```

- Check your balance

```
solana balance
```

- Create token mint

```
spl-token create-token
```

- Verify token mint on chain

  - Check the token on solana fm https://solana.fm/address/ChNkv9iW5pZJ1YAsNswC2CrdMUkFJBUbRWinjdLvKpXA/transactions?cluster=devnet-solana
  - Use the `getAccountInfo` to see the `data` and `lamports` in the account

- Check the supply of the token

```
spl-token supply AQoKYV7tYpTrFZN6P5oUufbQKAUr9mNYGe1TTJC9wajM
```

- Create an associated token account

```
spl-token create-account ChNkv9iW5pZJ1YAsNswC2CrdMUkFJBUbRWinjdLv
```

- Mint some tokens to yourself

```
spl-token mint  ChNkv9iW5pZJ1YAsNswC2CrdMUkFJBUbRWinjdLvKpXA 100
```

▼ Check your balances in the explorer

▼ Import the token in Phantom and see the balances

Equivalent code in JS

▼ Create a new cli wallet

```
solana-keygen new
```

▼ Set the RPC url

```
solana config set --url https://api.devnet.solana.com
```

▼ Create an empty JS file

```
npm init -y
touch index.js
```

▼ Install dependencies

```
npm install @solana/web3.js @solana/spl-token
```

▼ Write a function to airdrop yourself some solana

```
const {Connection, LAMPORTS_PER_SOL, clusterApiUrl, PublicKey} = require(

const connection = new Connection(clusterApiUrl('devnet'));

async function airdrop(publicKey, amount) {
    const airdropSignature = await connection.requestAirdrop(new PublicKey
    await connection.confirmTransaction({signature: airdropSignature})
```

```
}
```



```
airdrop("GokppTzVZi2LT1MSTWoEprM4YLDPy7wQ478Rm3r77yEw", LAMPORTS_
    console.log('Airdrop signature:', signature);
});
```

Check your balance

```
solana balance
```

Create token mint

```
const { createMint } = require('@solana/spl-token');
const { Keypair, Connection, clusterApiUrl,  TOKEN_PROGRAM_ID } = require(

const payer = Keypair.fromSecretKey(Uint8Array.from([102,144,169,42,220,87

const mintAthority = payer;

const connection = new Connection(clusterApiUrl('devnet'));

async function createMintForToken(payer, mintAuthority) {
    const mint = await createMint(
        connection,
        payer,
        mintAuthority,
        null,
        6,
        TOKEN_PROGRAM_ID
    );
    console.log('Mint created at', mint.toBase58());
    return mint;
}

async function main() {
    const mint = await createMintForToken(payer, mintAthority.publicKey);
}

main();
```

Verify token mint on chain

- Check the token on solana fm

https://solana.fm/address/GrNkv...YAsNswC2CrdMUkFJBUbRWinjdLvKpXA/transactions?cluster=devnet-solana
- Use the `getAccountInfo` to see the `data` and `lamports` in the account

▼ Create an associated token account, mint some tokens

```
const { createMint, getOrCreateAssociatedTokenAccount, mintTo } = requir
const { Keypair, Connection, clusterApiUrl,  TOKEN_PROGRAM_ID, PublicKey }

const payer = Keypair.fromSecretKey(Uint8Array.from([102,144,169,42,220,87

const mintAthority = payer;

const connection = new Connection(clusterApiUrl('devnet'));

async function createMintForToken(payer, mintAuthority) {
    const mint = await createMint(
        connection,
        payer,
        mintAuthority,
        null,
        6,
        TOKEN_PROGRAM_ID
    );
    console.log('Mint created at', mint.toBase58());
    return mint;
}

async function mintNewTokens(mint, to, amount) {
    const tokenAccount = await getOrCreateAssociatedTokenAccount(
        connection,
        payer,
        mint,
        new PublicKey(to)
      );

      console.log('Token account created at', tokenAccount.address.toBase58
      await mintTo(
        connection,
        payer,
```

```
        mint,
        tokenAccount.address,
        payer,
        amount
      )
      console.log('Minted', amount, 'tokens to', tokenAccount.address.toBase58
}

async function main() {
    const mint = await createMintForToken(payer, mintAthority.publicKey);
    await mintNewTokens(mint, mintAthority.publicKey, 100);
}

main();
```

▼ Check your balances in the explorer

▼ Import the token in Phantom and see the balances

Equivalent code in rust/python/go

Solana has libraries similar to `@solana/web3.js` in Rust, Python that would let you do the same thing.

In the end, they all are sending requests to an RPC server.

PDAs

When you created an associated token account , you actually created a PDA - https://github.com/solana-labs/solana-program-library/blob/master/associated-token-account/program/src/lib.rs#L71

JS - https://github.com/solana-labs/solana-program-library/blob/ab830053c59c9c35bc3a727703aacf40c1215132/token/js/src/st

ate/mint.ts#L171

Token-22 program

Ref - https://spl.solana.com/token-2022

A token program on the Solana blockchain, defining a common implementation for fungible and non-fungible tokens.

The Token-2022 Program, also known as Token Extensions, is a superset of the functionality provided by the Token Program.

▼ Create token mint


```
spl-token create-token  --program-id TokenzQdBNbLqP5VEhdkAS6EPFLC1P
```

▼ Create an associated token account

```
spl-token create-account 8fTM5XYRaoTJU9PLUuyakF3EypQ4RXL5HxKtiw2z9p
```

▼ Mint the tokens

```
spl-token mint 8fTM5XYRaoTJU9PLUuyakF3EypQ4RXL5HxKtiw2z9pQQ 100
```

Token-22 with metadata

https://cdn.100xdevs.com/metadata.json

▼ Create a token with metadata enabled

```
spl-token --program-id TokenzQdBNbLqP5VEhdkAS6EPFLC1PHnBqCXEpPxuE
```

▼ Create metadata

```
spl-token initialize-metadata pXfZ6Hg2s78m1iSRVsdzos9TmfkqkQdv5MmQr
```

▼ Create ATA

```
spl-token create-account pXfZ6Hg2s78m1iSRVsdzos9TmfkqkQdv5MmQrr7
```

▼ Mint

```
spl-token mint 1000
```

▼ Check out the token in your wallet

Assignment

1. Show all the tokens that the user has in our web based wallet
2. Create a token launchpad website that lets users launch tokens (take things like decimals, freeze athority as inputs from the user)