Getting Started with Solana:

A Developer's Handbook

Part 1

Punar Dutt Rajput

Rinki

C#Corner

Getting Started with Solana:

A Developer's Handbook

Punar Dutt Rajput, Rinki

Author – Punar Dutt Rajput
Co-Author – Rinki
Publisher – C# Corner
Editorial Team – Deepak Tewatia, Baibhav Kumar
Publishing Team – Praveen Kumar
Promotional & Media – Rohit Tomar

Background and Expertise

Punar Dutt Rajput is a seasoned software engineer with extensive experience in blockchain technologies and Microsoft platforms. Having worked with various blockchains such as Solana and NEAR Protocol, he brings a wealth of knowledge and hands-on expertise to the field of decentralized technologies. In addition to blockchain, he is proficient in Microsoft technologies, particularly ASP.NET Core.

He holds both a Bachelor's and a Master's degree in Computer Applications from Chhatrapati Shahu Ji Maharaj University. His academic background, combined with his practical experience, fuels his passion for advancing and exploring new horizons in blockchain technology. Dedicated to sharing his insights and innovations, continues to contribute to the evolving landscape of decentralized applications and digital solutions.

Rinki is a seasoned software developer specializing in .NET technologies, including C# and ASP.NET MVC. With a robust background in blockchain development, she has worked extensively with NEAR and Solana blockchains. She holds a Master of Computer Applications degree from Maharana Pratap Engineering College. Passionate about innovation and technology, she brings a wealth of knowledge and experience to the world of blockchain through her insightful writing.

Table of Contents:

1

Introduction to Solana

Overview

This chapter delves into Solana's innovative blockchain features, contrasts them with Ethereum, and outlines practical applications in trading, gaming, DeFi, and real-time data. Readers will gain insights into Solana's speed, scalability, and cost-effectiveness, positioning it as a pivotal player in decentralized technology and finance.

Introduction

In the ever-evolving world of blockchain technology, Solana emerges as a game-changer. By leveraging innovative consensus mechanisms like proof-of-history (PoH) and parallel processing, Solana achieves thousands of transactions per second (TPS) at minimal cost. In this chapter, we will explore Solana's core features, its advantages over traditional platforms like Ethereum, and the potential it holds for revolutionizing decentralized ecosystems.

What is Solana?

Solana is a high-performance blockchain platform designed for decentralized applications (dApps) and crypto-native projects. Launched in March 2020 by Solana Labs, it aims to provide scalability without sacrificing decentralization or security. At its core, Solana utilizes a unique blend of innovative technologies to achieve its goals, including a proof-of-history (PoH) consensus mechanism, proof-of-stake (PoS) validation, and a horizontally scalable architecture.

Solana distinguishes itself through its emphasis on scalability and speed. Traditional blockchain networks often face scalability issues, leading to congestion and high transaction fees. Solana addresses this challenge by leveraging its novel consensus mechanism, PoH, which timestamps transactions before they are processed. This approach enables Solana to achieve high throughput, with the network capable of processing thousands of transactions per second (TPS) at a low cost.

Key Features of Solana

Let us explore some of its key functionalities:

Scalability:

Solana's architecture is designed for horizontal scalability, allowing the network to handle increasing transaction volumes without compromising performance. This scalability is achieved through its unique consensus mechanism and parallel processing capabilities.

Speed:

Solana boasts one of the fastest transaction processing times among blockchain platforms, with transactions confirmed in seconds rather than minutes or hours. This speed is critical for applications requiring real-time interaction and responsiveness.

Low Cost:

By optimizing its protocol for efficiency, Solana offers low transaction fees compared to other blockchain networks. This affordability makes it an attractive option for developers and users alike..

Security:

Despite its focus on speed and scalability, Solana maintains a high level of security through its PoS validation and robust network architecture. By decentralizing control and incentivizing validators, Solana ensures the integrity and resilience of its blockchain.

Prerequisites for Learning Solana

Before getting started with Solana, let’s see the prerequisites-

Understanding of Blockchain Basics: Before diving into Solana, it's essential to have a solid understanding of how blockchain technology works, including concepts like decentralization, consensus mechanisms, smart contracts, and cryptographic principles.

Programming Skills: While not strictly necessary, having programming skills, especially in languages like Rust or JavaScript, can facilitate a deeper understanding of Solana's technical aspects, such as smart contract development and interacting with the Solana blockchain.

Solana Vs Ethereum

While Ethereum remains the leading platform for smart contracts and decentralized applications, it faces challenges related to scalability and congestion. Solana seeks to address these issues with its scalable architecture and high-performance capabilities.

Here's a comparative analysis of Solana and Ethereum across key parameters:

Parameters	Solana	Ethereum
Scalability	Achieves thousands of Transactions Per Second (TPS) with horizontal scalability.	Limited scalability, leading to network congestion during periods of high demand.
Transaction Speed	Transactions are confirmed in seconds.	Longer confirmation times, especially during peak usage periods.
Cost	Low transaction fees, making it cost-effective for users and developers.	High gas fees during network congestion, impact usability and affordability.
Ecosystem	Rapidly growing ecosystem with diverse dApps (Decentralized Applications) and projects.	An established ecosystem with a wide range of dApps, DeFi protocols, and NFT platforms.
Learning Curve	Innovative concepts and technologies may be challenging to grasp initially, especially for beginners in blockchain development.	Slightly more approachable for newcomers.

When to Use Solana?

Solana is particularly well-suited for use cases that demand high throughput, real-time responsiveness, and cost-effective transaction processing. Here are some scenarios where Solana can be used:

High-Frequency Trading: Solana's low latency and high throughput make it an ideal platform for high-frequency trading applications where split-second transaction execution is crucial.

Gaming and NFT Marketplaces: Solana's speed and scalability are advantageous for gaming platforms and NFT marketplaces, where rapid transaction processing and interaction are essential for a seamless user experience.

Decentralized Finance (DeFi): Solana's low transaction fees and fast confirmation times make it attractive for various DeFi applications, including decentralized exchanges (DEXs), lending protocols, and yield farming platforms.

Real-Time Data Applications: Solana's ability to process transactions in seconds makes it suitable for applications requiring real-time data processing, such as IoT (Internet of Things) networks, supply chain management, and streaming services.

Large-Scale Tokenization: Projects looking to tokenize assets at scale can benefit from Solana's scalability and low cost, enabling efficient token creation, transfer, and management on the blockchain.

2

Setting Up Your Development Environment

Overview

This chapter guides you through the essential steps to establish a robust development setup for Solana. You will learn how to configure your environment for seamless integration with Solana's ecosystem, ensuring that your tools operate efficiently. By the end of this chapter, you will have the knowledge to create and deploy smart contracts and applications smoothly. Additionally, you will become proficient in testing and experimenting with your ideas, enabling you to develop innovative solutions across various domains, such as finance and digital assets.

Introduction

A strong development setup is super important for building stuff on Solana. When you set it up right, it makes everything smoother. Plus, it makes sure your tools work well with Solana's system, so you can easily use them to create and launch your smart contracts and apps. Having the right setup also gives you all the tools you need to test and try out your ideas quickly. By getting to know Solana's system and how to work with it, you can make cool stuff for lots of different things, like finance or unique digital items.

We can install Solana on three operating systems Windows(WSL), Linux, and MacOS. For now, we will use Windows(WSL).

Install WSL

WSL, or Windows Subsystem for Linux, is a compatibility layer for Windows that enables you to run Linux distributions directly on your Windows machine. It allows developers to use Linux tools, utilities, and command-line interfaces on a Windows environment without the need for virtual machines or dual-booting. In the context of Solana development, WSL is often used to create a Linux-like environment on Windows systems, providing seamless compatibility with Solana's tooling and libraries.

For running the Solana in Windows, we need to install WSL in your machine. Follow this article to install WSL

Install Rust

Rust is a programming language known for its speed, reliability, and safety features. It's popular in the development of blockchain projects like Solana because it allows developers to write efficient and secure code. In the context of setting up a Solana development environment, Rust is used to write smart contracts that run on the Solana blockchain. It's a key tool in building decentralized applications on Solana.

We need to install Rust as we write our programs in the Rust language. To install Rust in your system first open the WSL terminal then paste the following command.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
```

So we have installed the Rust successfully. Now restart your terminal or run the following command to refresh your new path setting.

```
source ~/.bashrc
```

Run the following command to check whether Rust is installed or not.

```
rustc --version
```

Install Solana CLI

Solana CLI is like a toolbox for working with Solana. It's a set of commands you can use in your computer's terminal to do all sorts of things on the Solana blockchain. You can create new projects, deploy smart contracts, send transactions, and much more, all from the command line. It's the main way developers interact with Solana's system directly from their computers.

Run the following command in your terminal to install Solana CLI

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

You have the option to substitute "stable" with the release tag that corresponds to the software version you want (such as v1.18.1), or you can opt for one of the three symbolic channel names: stable, beta, or edge.

Now run the following command to check the version of Solana.

```
solana --version
```

Output

```
solana-cli 1.17.25 (src:d0ed878d; feat:3580551090, client:SolanaLabs)
```

We have successfully installed the Solana CLI, Let's move to the next step by installing the 'Anchor' framework.

Install Anchor

To set up and maintain different versions of the Anchor framework, we'll use AVM, the Anchor Version Manager. Because AVM is installed through cargo (Rust's package manager), the installation process remains consistent across all operating systems. Once installed, AVM enables us to easily install the specific version of the Anchor framework we need.

Install AVM

Run the following command in your terminal to install avm.

```
cargo install --git https://github.com/coral-xyz/anchor avm --locked --
force
```

Install Anchor using AVM

Restart your terminal and run the following command to install Anchor.

```
avm install latest
avm use latest
```

Setup a localhost blockchain cluster

The Solana CLI includes a built-in test validator, enabling you to run a complete blockchain cluster directly from your command line interface.

Now open a new tab of the terminal and run the following command to run the localhost cluster.

```
solana-test-validator
```

Now return to the first terminal tab and configure the Solana CLI with the local cluster by running the following command.

```
solana config set --url localhost
```

To view the configuration of the Solana cli run the following command.

```
solana config get
```

Create a file system wallet

In Solana, a file system wallet is like a digital wallet that lives on your computer. It's a secure place where you can keep your Solana cryptocurrency and interact with the Solana blockchain. Instead of relying on a third-party service, like an online wallet, a file system wallet stores your cryptocurrency keys directly on your device. This gives you more control and security over your funds. You can use your file system wallet to send and receive Solana tokens, participate in decentralized finance (DeFi) activities, and more, all from your computer. It's an essential tool for anyone looking to engage with the Solana ecosystem.

Let's create a file system wallet to use during the development. Run the following command in the terminal.

```
solana-keygen new
```

The solana-keygen command automatically generates a new file system wallet, which is initially saved at ~/.config/solana/id.json. You have the option to specify a different location for the output file by using the --outfile /path option.

Set your new wallet as the default

Run the following command to set the wallet as default.

```
solana config set -k ~/.config/solana/id.json
```

Airdrop SOL token in your wallet

After setting a new wallet as default, you can request a free SOL token by running the following command.

```
solana airdrop 2
```

Now check the current balance in your wallet.

```
solana balance
```

3

Understanding Solana’s Architecture

Overview

This chapter explores Solana, a blockchain platform known for processing thousands of transactions per second. This capability supports various applications, from dApps to digital asset exchanges. We will examine the consensus mechanism and architecture that enable Solana's impressive performance.

Introduction

This chapter explores Solana, a blockchain platform known for processing thousands of transactions per second. This capability supports various applications, from dApps to digital asset exchanges. We will examine the consensus mechanism and architecture that enable Solana's impressive performance.
We will explore how high throughput and low latency transactions are approved and added to the blockchain by exploring the inner workings of Solana's consensus process, which combines Proof of Stake (PoS) and Proof of History (PoH).

Consensus Mechanism

Solana's consensus mechanism is a fundamental aspect of its architecture, combining Proof of History (PoH) and Proof of Stake (PoS) to achieve fast and secure transaction validation.

Proof of History

Solana's PoH is a unique feature that provides a verifiable and trustless record of time in the blockchain. It establishes a chronological order of events without relying on a centralized clock. PoH allows validators to timestamp transactions, ensuring the integrity of the blockchain accurately. PoH encodes the trustless passage of time into a ledger, which is essentially an append-only data structure. Unlike traditional blockchains that rely on local clocks, PoH provides verifiable timestamps for events.

How does PoH work?

A cryptographically secure function performs a sequence of computations on a single-core processor. Each output becomes the input for the next computation.
The resulting PoH ledger ensures that the order of events is accurately recorded, allowing for precise message ordering.

Key benefits of PoH

- **Fast Verification:** PoH enables sub-second finality times, reducing messaging overhead in a Byzantine Fault Tolerant (BFT) replicated state machine.
- **Trustless Time:** Participants can trust the elapsed time between events without relying on centralized clocks.
- **Secure Timestamps:** Timestamps are resistant to forgery.
- **Scalability:** PoH contributes to Solana's high throughput.

Proof of Stake (PoS)

In Solana's PoS mechanism, validators are chosen based on the number of tokens they hold and stake in the network. Validators are responsible for proposing and validating new blocks of transactions. PoS ensures decentralization and security by distributing block production among a diverse set of validators.

Breakdown of PoH and PoS Working Together

- **PoH Timestamping:** PoH provides validators with a verifiable time source, allowing them to order transactions accurately. Validators use PoH timestamps to create new blocks and validate transactions.
- **PoS Consensus:** Validators participate in a leader election process based on their stake in the network. Through a randomized selection algorithm, validators are chosen to

propose and validate blocks. This decentralized approach ensures that no single entity can control the network.

- **Synergy Between PoH and PoS:** By combining PoH's timestamping capabilities with PoS's decentralized consensus mechanism, Solana achieves fast and secure transaction validation. PoH ensures the accurate ordering of transactions, while PoS prevents double-spending and other malicious activities.

BFT

BFT stands for Byzantine Fault Tolerance, and it's a critical aspect of Solana's consensus mechanism. In Solana's consensus model, BFT ensures that the network can maintain agreement and consistency even in the presence of faulty or malicious nodes.

How does BFT work in the context of Solana?

- **Fault Tolerance:** BFT ensures that even if some nodes in the network fail or behave maliciously, the system as a whole can still reach a consensus on the state of the blockchain. This fault tolerance is essential for maintaining the integrity and security of the network.
- **Resilience to Attacks:** Byzantine Fault Tolerance enables Solana to resist various types of attacks, including attempts to manipulate transaction history, double-spend digital assets, or disrupt network operations.
- **Decentralization:** BFT contributes to the decentralization of the Solana network by allowing a diverse set of validators to participate in the consensus process. This distributed approach prevents any single point of failure or control, enhancing the network's resilience and censorship resistance.

Advantages of Solana's Consensus Mechanism

- **Scalability:** Solana's consensus mechanism enables high throughput, allowing the network to process thousands of transactions per second. PoH's fast verification times and PoS's efficient block production contribute to Solana's scalability.
- **Security:** The combination of PoH and PoS provides robust security against attacks and ensures the integrity of the blockchain. PoS ensures decentralization, while PoH's secure timestamps prevent manipulation of transaction order.
- **Decentralization:** Solana's PoS mechanism encourages broad participation in block production and validation, enhancing decentralization. This distributed approach prevents any single entity from controlling the network, promoting resilience and censorship resistance.

Solana's consensus mechanism enables high throughput, allowing the network to process thousands of transactions per second. PoH's fast verification times and PoS's efficient block production contribute to Solana's scalability.

Architecture of Solana

Solana's architecture is designed to support its high-performance blockchain platform, consisting of several layers that work together to facilitate fast and efficient transaction processing. Solana's architecture can be divided into three main layers.

Transaction Processing Layer

This layer is responsible for processing transactions submitted to the Solana network. Transactions are bundled into blocks and validated by network validators.

- Responsible for validating incoming transactions.
- Utilizes parallel processing to handle multiple transactions simultaneously.
- Includes features like parallel smart contract execution and parallel signature verification.

Consensus Layer

The consensus layer ensures agreement among network participants on the state of the blockchain. Solana uses a hybrid consensus mechanism, combining Proof of History (PoH) and Proof of Stake (PoS), to achieve consensus efficiently.

- Combines PoH and PoS.
- Validators use PoH timestamps to order transactions.
- Continuous block production, even if some slots have slow or unresponsive leaders.

Ledger Storage Layer

The ledger storage layer stores the blockchain's data, including transaction history and state information. Solana utilizes a distributed ledger storage system to maintain the integrity and accessibility of the blockchain data.

- The ledger storage layer stores the blockchain's data in a distributed manner across network nodes.
- Solana's ledger storage system utilizes a combination of sharding and replication to distribute data across multiple nodes while ensuring data consistency and availability.
- This layer plays a crucial role in maintaining the integrity and accessibility of the blockchain data, enabling efficient querying and retrieval of transaction information.

Key Components of Solana's Architecture

Solana's architecture consists of several key components that play crucial roles in maintaining the integrity, security, and performance of the network.

1. Nodes

Nodes are individual computers or servers that participate in the Solana network by running Solana software. There are different types of nodes in Solana.

- **Validator Nodes:** Validator nodes are responsible for validating transactions, proposing new blocks, and participating in the consensus process. They play a critical role in maintaining the integrity and security of the network.
- **Replicator Nodes:** Replicator nodes replicate and store copies of the blockchain ledger. They help distribute the load of storing blockchain data across the network, improving data redundancy and availability.
- **Archiver Nodes:** Archiver nodes store historical data of the blockchain, including past transactions and state changes. They provide access to historical data for auditing, analysis, and other purposes.

2. Validators

Validators are network participants responsible for validating transactions and maintaining the integrity of the blockchain. Validators play a key role in the consensus process by proposing and validating new blocks. They are selected based on the number of tokens they hold and stake in the network.
Validators ensure that transactions adhere to the network's rules and prevent double-spending and other malicious activities. They participate in a leader election process to propose blocks and reach a consensus on the state of the blockchain.

3. Cluster Architecture

Solana's cluster architecture consists of multiple nodes distributed across the network. The cluster architecture ensures decentralization and resilience by distributing the processing and storage of blockchain data across multiple nodes.
The cluster architecture includes different types of nodes, such as validator nodes, replicator nodes, and archiver nodes, working together to maintain the integrity and security of the network. By distributing tasks and responsibilities across the network, the cluster architecture improves scalability, fault tolerance, and performance.

Interactions Between Components

- Validator nodes validate transactions and propose new blocks based on the consensus mechanism.
- Replicator nodes replicate and store copies of the blockchain ledger, improving data redundancy and availability.
- Archiver nodes store historical data of the blockchain, providing access to past transactions and state changes.
- Validators collaborate to reach a consensus on the state of the blockchain, ensuring that all transactions are valid and consistent.

4

Accounts & Programs

Overview

This chapter explains how programs, accounts, and Program Derived Addresses (PDAs) form the foundation of the Solana blockchain, enabling the creation and operation of decentralized applications (DApps). Programs act as smart contracts, managing various functions and interactions, while accounts store data, and PDAs provide deterministic, secure ways to manage and sign transactions programmatically.

Introduction

Programs in Solana are the magic that brings decentralized applications to life. They provide the instructions for how the blockchain should operate, while accounts serve as the containers for storing data. Together, they form the backbone of the Solana ecosystem, enabling developers to build powerful, decentralized solutions for a wide range of applications. In comparison, PDA can be used for deterministic addresses.

Solana's innovative approach to building decentralized applications revolves around the concepts of Programs, Accounts, and PDAs. These foundational elements form the backbone of the Solana ecosystem, enabling developers to create high-performance DApps with ease.

Programs in Solana

In Solana, programs are like the building blocks of decentralized applications. Think of them as the contracts powering various functions and operations on the Solana blockchain. These programs are typically written in languages like Rust or C/C++, known for their reliability and efficiency.

In other words, Programs are nothing but "Smart Contracts". If you are from some other blockchain like Ethereum, you must be familiar with Smart contracts even if you are not, there is no need to worry as we will be covering all the aspects of programs.

Each program is an on-chain account that stores some executable code that is organized into specific functions known as instructions. It can be updated only by the upgrade authority, i.e. the account that deployed the program on the chain. If the upgrade authority is set to null then the program becomes immutable.

So, what do these programs do? They enable a wide range of functionalities, from token swaps to decentralized finance (DeFi) protocols and even non-fungible token (NFT) marketplaces. Essentially, anything you can imagine doing on a blockchain can be achieved through these programs.

Example: Let's say you want to create a decentralized voting application on Solana. You would write a program that includes instructions for:

- Creating a new voting session.
- Allowing users to cast their votes.
- Tallying the votes securely.

Each of these tasks would be defined in the program's code. For instance, the program would specify how to record a user's vote, ensure that each user can only vote once, and count the votes accurately.

Accounts in Solana

In Solana, accounts serve as the data containers that hold information about various entities on the blockchain. These entities could be tokens, programs, or even user balances. In the account, all data is stored in a key-value pair, where the key will be the address and the value will be the account information. An account can store up to 10MB of data. Some rent is deposited based on the amount of data stored in the account and can be refunded once the account is closed.

Accounts come in different types, such as program accounts, user accounts, and system accounts. Each type of account plays a specific role in the Solana ecosystem. For example, program accounts store the code and state of programs, while user accounts hold the balances and transaction history of individual users.

Now, let's talk about how programs interact with accounts on Solana. Think of accounts as the storage units where data is kept on the blockchain. In our voting example, each voter's information, including their choices, would be stored in individual accounts.

The program would interact with these accounts by reading data from them (to check if a user has already voted, for example) and writing data to them (to record a user's vote). This interaction ensures that the voting process is transparent, secure, and tamper-proof.

When a program is deployed, three types of accounts are created. They are-

- **Buffer Account:** A buffer account is a temporary account that stores byte code while the program is being deployed on-chain or upgraded. Once this process is complete, all data is transferred to the Program Executable Data Account, and the buffer account is closed.
- **Program Executable Data Account:** It is an account that contains the executable byte code of the program.
- **Program Account:** It is the main account representing the on-chain program. It stores the address of the executable data account, and the address authorized to make changes to the program.

Program Derived Addresses (PDAs)

Program Derived Addresses, or PDAs for short, are a clever mechanism used to facilitate interactions between different programs and accounts. PDAs are derived from a program's address and a set of seeds, which are essentially parameters that determine the properties of the PDA.

Solana keypairs are points on the Ed25519 curve on which, for each public key there exists a private key, but PDA fall off this curve and, hence, have no private key and cannot be used for signing transactions the way normal keypairs are used. But PDA can sign transactions by using the program that was used to create PDA.

The beauty of PDAs lies in their versatility. They can be used to create new accounts, perform transactions, and execute program functions, all while ensuring security and efficiency on the Solana blockchain.

Use case of Program Derived Addresses (PDAs)

- Deterministic account address as derived from seeds and program ID.
- Enables the program to sign transactions programmatically.

Example : Continuing with our voting application, suppose we need a PDA to handle the distribution of tokens as incentives for voting. We can derive a PDA using the voting program's address and specific needs related to token distribution. This PDA can then create new accounts to store tokens and execute transactions, all while ensuring security and efficiency on the Solana blockchain.

How to derive a Program Derived Addresses (PDA) in Solana?

Now, let's see how we can derive a Program Derived Addresses (PDA) in Solana using web3.js.

```
import { PublicKey } from "@solana/web3.js";
const programId = new PublicKey("11111111111111111111111111111111");
const string = "hello";
const [PDA, bump] = PublicKey.findProgramAddressSync(
  [Buffer.from(string)],
  programId,
);
```

```
console.log(`PDA: ${PDA}`);
console.log(`Bump: ${bump}`);
```

In the above code, the first line imports the PublicKey class from the @solana/web3.js library. The PublicKey class is used to represent public keys on the Solana blockchain. In the next line, we create a new PublicKey object representing the program ID "11111111111111111111111111111111". In Solana, programs are identified by their program ID, and this line sets the program ID for the program being interacted with. In the next line, we define a string "hello" that will be used in the next steps. In the next line, we use, the findProgramAddressSync method called on the PublicKey class to synchronously find the Program Derived Address (PDA) and the bump seed. The first argument is an array containing the buffer representation of the string "hello". The Buffer.from() method converts the string to a buffer. The second argument is the program id. The method returns an array containing the PDA and the bump seeds. At last, we log the PDA and the bump seed to the console.

And our PDA is derived.

5

Solana Explorer

Overview

Explorers play a vital role in the blockchain ecosystem, offering real-time insights into transactions, addresses, and blocks. This chapter focuses on top explorers in the Solana ecosystem, helping users select reliable tools and understand Solana's network dynamics and future trends.

Introduction

Explorers are essential tools in the blockchain ecosystem, offering real-time visibility into transactions, addresses, and blocks. They empower users, developers, and investors by providing insights into network activity, debugging smart contracts, and tracking token movements. This chapter aims to spotlight top explorers in the Solana ecosystem, aiding users in selecting reliable tools and providing insights into Solana's dynamics and future trends.

Criteria for Selecting Top Explorers

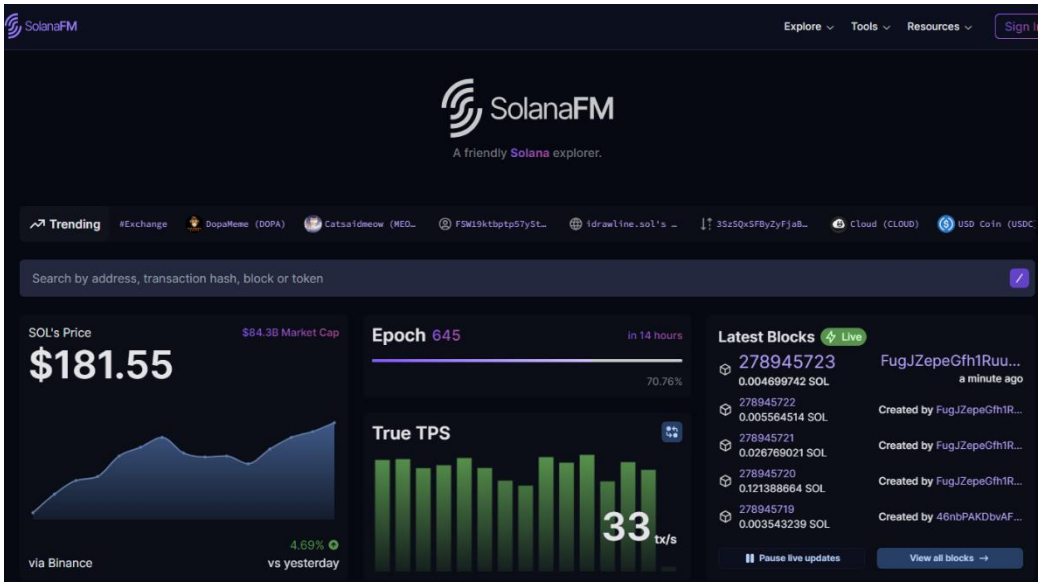
- **Reliability and Performance:** Users rely on explorers to provide accurate and timely information about blockchain transactions and network activity. A top explorer should have robust infrastructure, minimal downtime, and fast response times to ensure users can access data when needed.
- **User Experience and Interface:** User experience (UX) and interface design play a significant role in the usability and adoption of an explorer. A top explorer should offer an intuitive and user-friendly interface that makes it easy for users to navigate, search for transactions, and access relevant data. Clear visualization of transaction details, block information, and account balances enhances the overall user experience.
- **Features and Tools Offered:** Beyond basic transaction tracking, a top explorer should provide advanced analytics, insights, and tools that cater to the diverse needs of users, developers, and investors. Features such as transaction history, address monitoring, smart contract debugging, and API access are essential for developers building on the blockchain. Additionally, tools for token analysis, portfolio tracking, and market data integration add value for investors and traders.
- **Community Engagement and Support:** A top explorer should actively engage with its community through channels such as forums, social media, and developer documentation. Regular updates, user feedback mechanisms, and responsive customer support demonstrate a commitment to addressing user needs and maintaining transparency. Furthermore, community-driven initiatives, educational resources, and developer outreach programs contribute to building a strong and vibrant explorer ecosystem.

Top Explorers

Explorer	Strengths	Weaknesses	Unique Offerings and Value Propositions
SolanaFM	Real-time transaction tracking	Limited community engagement.	Advanced analytics and insights. Developer tools for smart contract debugging. Customizable dashboard for personalized user experience.
Solscan	Reliable performance	Limited features compared to others.	API access for developers.
Solana Beach	Feature-rich explorer	Some users report occasional downtime.	Support for token analysis and portfolio tracking. Modern and intuitive interface.
Solana Explorer	User-friendly interface	Limited community engagement.	Developer tools for smart contract debugging.
OKLink	Versatility across multiple blockchains	It may lack specialized features for the Solana ecosystem.	Support for multi-chain integration. Real-time updates. Advanced analytics and visualizations.

SolanaFM

SolanaFM is a comprehensive explorer of the Solana blockchain, offering a wide range of features and tools for users, developers, and investors. It provides real-time visibility into transactions, blocks, and accounts on the Solana network.



(source: https://solana.fm/)

Key Features

- Real-time transaction tracking
- Block explorer with detailed information
- Account monitoring and balance tracking
- Advanced analytics and insights
- Developer tools for smart contract debugging
- Customizable dashboard for personalized user experience

User Experience

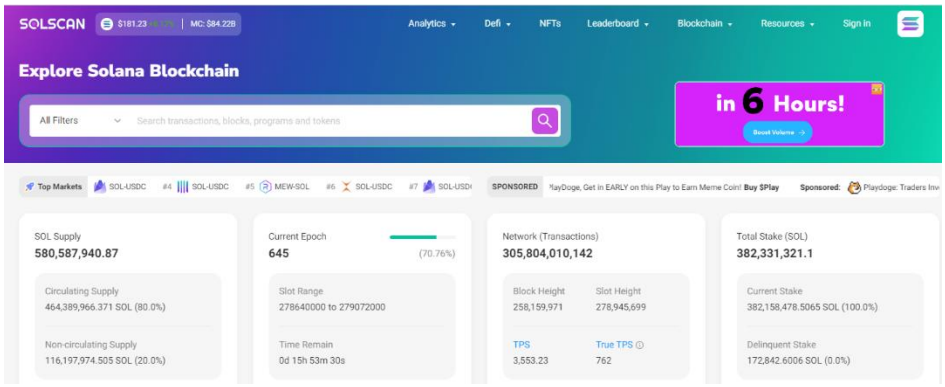
SolanaFM offers a user-friendly interface with intuitive navigation and clear visualization of data. Users can easily search for transactions, explore block details, and monitor their accounts. The platform's responsive design ensures a seamless experience across devices.

Community Engagement

SolanaFM actively engages with its community through social media channels, forums, and developer documentation. The platform regularly updates its features based on user feedback and provides responsive customer support. Community-driven initiatives and educational resources contribute to a vibrant explorer ecosystem.

Solscan

Solscan is a popular explorer for the Solana blockchain, known for its reliability and performance. It offers a range of features for tracking transactions, exploring blocks, and monitoring accounts on the Solana network.



(source: https://solscan.io/)

Key Features

- Transaction explorer with real-time updates
- Block details and confirmation times
- Account monitoring and balance tracking
- Rich analytics and visualizations
- API access for developers

User Experience

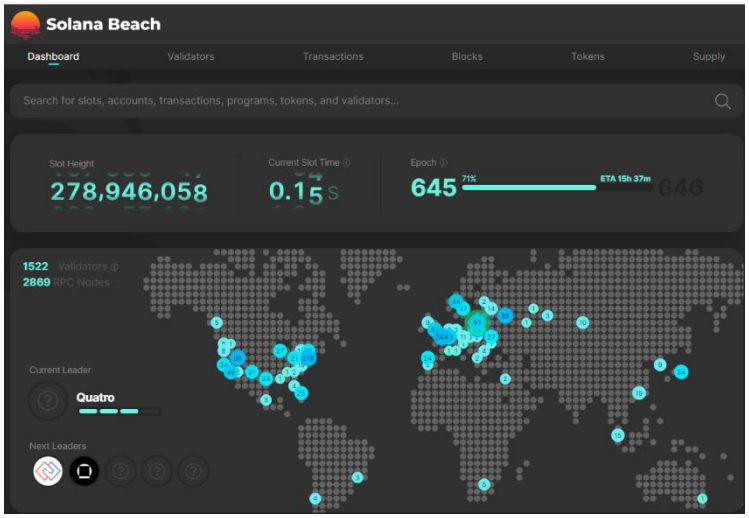
Solscan provides a clean and user-friendly interface, making it easy for users to navigate and find the information they need. The platform offers fast response times and reliable performance, ensuring a smooth user experience.

Community Engagement

Solscan maintains an active presence in the Solana community, regularly interacting with users through social media and forums. The platform solicits feedback from its users and incorporates community suggestions into its development roadmap. Developer resources and documentation contribute to a supportive explorer ecosystem.

Solana Beach

Solana Beach is a feature-rich explorer for the Solana blockchain, offering a comprehensive suite of tools for users, developers, and investors. It provides detailed insights into transactions, blocks, and accounts on the Solana network.



(source: https://solanabeach.io/)

Key Features

- Transaction explorer with detailed transaction history
- Block explorer with real-time updates
- Account monitoring and balance tracking
- Advanced analytics and visualizations
- Support for token analysis and portfolio tracking

User Experience

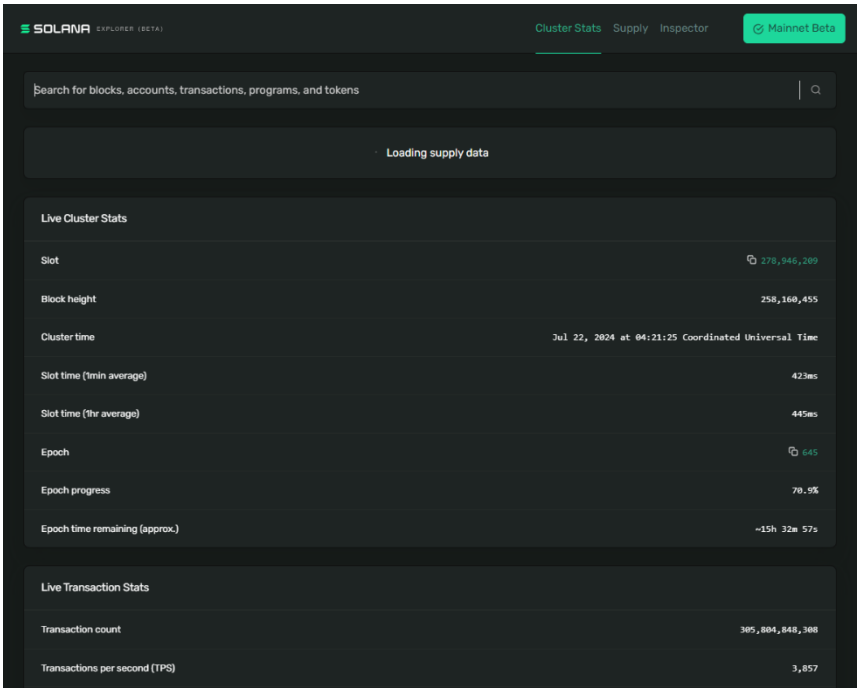
Solana Beach offers a modern and intuitive interface, making it easy for users to navigate and explore the blockchain. The platform's responsive design ensures a seamless experience across devices, while its rich visualizations provide valuable insights into network activity.

Community Engagement

Solana Beach actively engages with its community through social media channels, forums, and developer documentation. The platform welcomes user feedback and suggestions for improvement, demonstrating a commitment to serving its users. Community-driven initiatives and educational resources contribute to a thriving explorer ecosystem

Solana Explorer

Solana Explorer is a reliable and user-friendly explorer designed for the Solana blockchain. It offers a range of features for tracking transactions, exploring blocks, and monitoring accounts, catering to the needs of users, developers, and investors.



(source: https://explorer.solana.com/)

Key Features

- Real-time transaction tracking with detailed information
- Block explorer with comprehensive block details
- Account monitoring and balance tracking
- Advanced analytics and visualizations

- Developer tools for smart contract debugging

User Experience

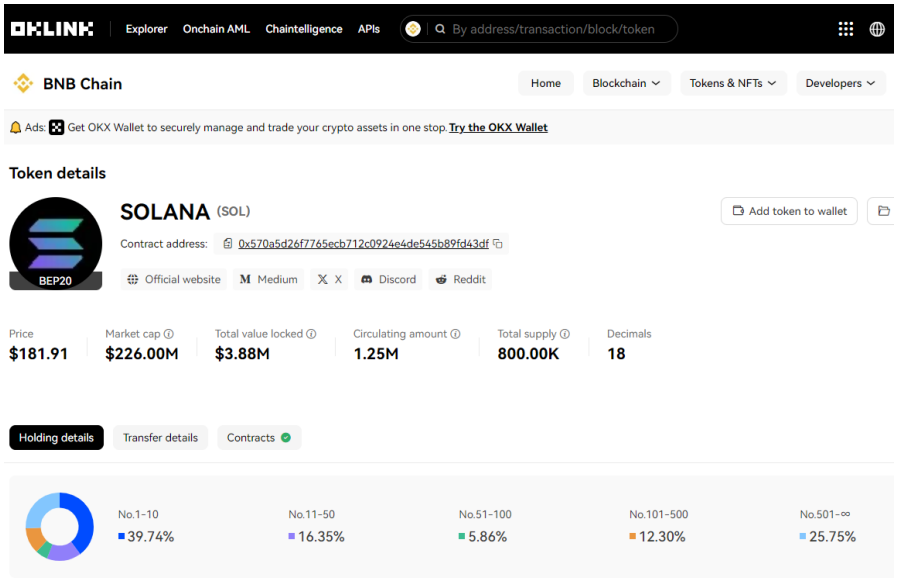
Solana Explorer provides a straightforward and intuitive interface, making it easy for users to access and analyze blockchain data. The platform offers fast response times and reliable performance, ensuring a seamless user experience.

Community Engagement

Solana Explorer actively engages with its community through social media, forums, and developer documentation. The platform welcomes user feedback and suggestions for improvement, demonstrating a commitment to serving its users. Community-driven initiatives and educational resources contribute to a vibrant explorer ecosystem.

OKLink

OKLink is a versatile explorer offering support for multiple blockchains, including Solana. It provides a range of features and tools for tracking transactions, exploring blocks, and monitoring accounts on the Solana network.



(source: https://www.oklink.com/)

Key Features

- Transaction explorer with real-time updates
- Block explorer with detailed block information
- Account monitoring and balance tracking
- Advanced analytics and visualizations
- Support for multi-chain integration

User Experience

OKLink offers a user-friendly interface with intuitive navigation and clear visualization of data. The platform provides fast response times and reliable performance, ensuring a seamless user experience.

Community Engagement

OKLink maintains an active presence in the blockchain community, regularly interacting with users through social media and forums. The platform welcomes user feedback and suggestions for improvement, demonstrating a commitment to serving its users. Community-driven initiatives and educational resources contribute to a thriving explorer ecosystem.

6

Transactions and Instructions

Overview

In this chapter, readers will explore the core elements of Solana's digital ecosystem, including transactions, instructions, and transaction fees. It offers an in-depth understanding of how these components drive the platform's efficiency and economic activities, preparing readers to effectively participate and leverage the opportunities within Solana's vibrant community.

Introduction

In this chapter, we will read about transactions, instructions, and transaction fees that are the bedrock of Solana's vibrant digital economy. By understanding these concepts, you'll be well-equipped to navigate the bustling seas of Solana and seize the endless opportunities it offers.

Transactions: The Building Blocks of Solana's Digital Economy

At its core, a transaction on Solana is a digital record of an action—a movement of tokens, data, or instructions from one digital wallet to another within the Solana blockchain. Picture it like sending a digital parcel from your digital backpack to a friend's digital mailbox. These parcels can contain SOL tokens, Solana's native cryptocurrency, or commands for decentralized applications (dApps) to execute specific tasks.

But what sets Solana apart is its speed. While other blockchains may leave you tapping your foot impatiently, waiting for transactions to confirm, Solana processes transactions in a flash, thanks to its innovative architecture and high-performance consensus mechanism.

Transaction

A transaction comprises one or more instructions, each representing a specific operation to be processed. It is atomic in nature, either fully completes or fully fails. The maximum size of a transaction is 1232 bytes.

A transaction contains two components: signature and message.

- **Signature:** Array of signatures of private key.
- **Message:** Payload of information that we send to the program. In other words, it is a list of instructions to be processed.

Message

This message comprises four main parts-

- **Header:** It specifies the number of signer and read-only accounts with or without signatures.
- **Account Addresses:** It is an array of addresses of the account that we are going to interact with.
- **Recent Blockhash:** It is the hash of the last observed blockchain ledger.
- **Instruction:** It is the array of instructions to be executed.

Example. Let's say A wants to send 1 SOL to B. Here's what the transaction would look like:

- **Signature:** A sign the transaction with her private key.
- **Message:** The header indicates A as the signer. The account addresses are A's and B's wallet addresses. Recent Blockhash is the hash from the latest block, and the instruction is the command to transfer 1 SOL from A's account to B's.

Instructions: The Blueprint for Action

Now, let's zoom in on the instructions. In the world of Solana, an instruction is like a carefully crafted blueprint, detailing the precise actions to be taken within a transaction. Whether you're transferring SOL tokens, minting NFTs (non-fungible tokens), or swapping assets on a decentralized exchange (DEX), each action within a transaction is executed according to a specific set of instructions. When you submit a transaction on Solana, you're essentially sending a package of instructions into the digital ether, where they're executed with precision and transparency.

Instruction is a request to process a specific action on-chain and is the smallest contiguous unit of execution logic in a program. An instruction contains the following information-

- **Program ID:** It specifies the program being invoked by the instruction.
- **Accounts:** It is the array of accounts the instructions read from or write to.
- **Instruction Data:** It is a byte array of data that the program will use to handle each transaction.

Example. Imagine you want to mint a new NFT. Here's what the instruction might include:

Program ID: The ID of the NFT minting program.

Accounts: Your wallet address, the destination account for the NFT, and the mint authority account.

Instruction Data: Data specifying the metadata of the NFT (e.g., name, description, image URL).

Transaction Fees: The Fuel for Solana's Engine

Now, let's talk about everyone's favorite topic: transaction fees. In traditional finance, making transactions can feel like bleeding money, with fees eating away at your hard-earned cash. But on Solana, transaction fees are refreshingly low, making it affordable to send even the tiniest fractions of SOL or interact with dApps to your heart's content.
It is the fees paid to process instructions on the Solana blockchain. It is paid to the network to help support the economic design of the blockchain.

Why pay transaction fees?

- Compensation to the validator.
- Reduce network spam.
- Provide long-term economic stability.

So, how are transaction fees determined on Solana? A fixed amount from the fees is burnt and the rest goes to the validator to keep the network going. Transaction fees is equal to the sum of base fees per signature and compute units.

In Solana, you can pay prioritization fees, an optional fee paid to prioritize the transaction. There are some of the best practices that you should follow to minimize fees while paying prioritization fees.

- Request minimum compute units.
- Get recent prioritization fees as this helps you know what others are paying.

Example. If you're sending a transaction with 1 signature and 2 instructions, and the base fee per signature is 0.000005 SOL, while each instruction consumes 500 compute units (with a base fee per compute unit of 0.000001 SOL), the total fee might be:

- Signature fee: 0.000005 SOL
- Instruction fees: 2 * 500 * 0.000001 SOL = 0.001 SOL
- Total fee: 0.001005 SOL

Monitoring Transaction Fees: Tools of the Trade

If you're curious about how to monitor transaction fees on Solana, fear not! There's a treasure trove of tools and resources at your disposal. The Solana Explorer, for example, is a web-based tool that allows you to delve into real-time transaction data on the Solana blockchain, including the fees paid for each transaction.

Furthermore, many Solana wallets and dApps come equipped with built-in fee estimators that can help you gauge the appropriate fee to include with your transactions. These estimators factor in current network conditions and transaction demand, providing you with optimal fee recommendations to ensure your transactions sail smoothly through the digital seas.

Tips for Developers: Common Pitfalls When Constructing Transactions on Solana

Now that we have discussed transactions, instructions, and transaction fees, let's see some tips that would help developers avoid the common pitfalls while constructing transactions on Solana.

- **Mind the Size Limit:** Keep transactions under 1232 bytes to avoid failures.
- **Order Instructions Properly:** Sequence instructions correctly to prevent logical errors.
- **Correct Account Addresses:** Use accurate and authorized account addresses.
- **Ensure Proper Signatures:** Include the right signatures for all involved accounts.
- **Use Recent Blockhash:** Always fetch the latest blockhash before submitting a transaction.
- **Optimize Compute Units:** Minimize compute unit usage to reduce fees and ensure smooth execution.
- **Include Sufficient Fees:** Calculate and include adequate fees to cover base and compute costs.
- **Monitor Network Conditions:** Adjust fees based on current network congestion.
- **Test on Testnet/Devnet:** Use Solana’s testnet or devnet to test transactions before mainnet deployment.
- **Leverage Tools and Libraries:** Use Solana-Web3.js and other tools for efficient transaction handling.
- **Implement Error Handling:** Handle transaction failures gracefully to enhance user experience.
- **Stay Updated:** Keep up with Solana's updates to avoid compatibility issues and leverage new features.

By following these tips, you can avoid common pitfalls and ensure smooth transaction construction on Solana.

7

Clusters in Solana

Overview

This chapter provides an in-depth exploration of Solana clusters, detailing their structure, operation, and significance in the Solana blockchain. It highlights key components such as validators, RPC nodes, and clients, and explains how clusters enhance scalability, performance, and fault tolerance. Additionally, it discusses various applications and challenges faced by Solana clusters, setting the stage for future advancements.

Introduction

Blockchain technology has revolutionized various industries by providing decentralized, secure, and transparent systems for transactions and data management. Among the many blockchain platforms, Solana stands out for its high-performance capabilities, promising unparalleled speed and efficiency. A critical component that underpins Solana's functionality is its clustering system. Solana clusters form the backbone of the network, ensuring scalability, reliability, and performance. This article delves into the intricacies of Solana clusters, exploring their structure, operation, and significance.

What is a Solana Cluster?

A Solana cluster is essentially a network of computers working together to maintain the Solana blockchain. Each cluster operates independently, maintaining a ledger and processing transactions according to the Solana protocol. There are different types of clusters, they are as follows-

- **Mainnet Beta:** The Mainnet Beta is the primary network where real transactions occur.
- **Testnet:** Testnet is used for testing new features and updates
- **Devnet:** Devnet is used by developers for experimentation and development.

Components of a Solana Cluster

A Solana cluster consists of several key components, each playing an important role in the network's functionality. These are as follows-

Validators

Validators are nodes responsible for processing transactions and adding new blocks to the blockchain. They verify the correctness of transactions and ensure that the ledger remains consistent. Validators play a crucial role in maintaining the network's security and performance.

RPC Nodes

Remote Procedure Call (RPC) nodes provide an interface for clients to interact with the blockchain. They handle requests such as querying account balances, submitting transactions, and fetching blockchain data. RPC nodes are essential for the seamless operation of decentralized applications (dApps) and end-user interactions.

Clients

Clients are the end-users and applications that interact with the Solana network. They submit transactions and query the blockchain via RPC nodes. Clients depend on the network's performance and reliability to execute their operations effectively.

Consensus Mechanism

Solana employs a unique consensus mechanism called Proof of History (PoH) combined with Tower Byzantine Fault Tolerance (BFT). PoH provides a cryptographic clock that allows nodes to agree on the time order of events, significantly improving throughput and efficiency. Tower BFT ensures the finality and security of transactions.

How do Solana clusters operate?

The operation of a Solana cluster involves several steps, from initialization to transaction processing. These are as follows-

Initialization and Bootstrapping

When a new Solana cluster is initialized, the process begins with nodes coming online and connecting. This bootstrapping process involves nodes synchronizing their local copies of the blockchain ledger, ensuring that they have the latest state of the network. Nodes also establish connections with other nodes in the cluster to form a cohesive network. During this phase, nodes may exchange cryptographic keys and other necessary information to authenticate and communicate securely.

Data Propagation and Transaction Processing

Once the cluster is initialized, it becomes operational, and ready to process transactions submitted by clients. Transactions are propagated across the network using a gossip protocol, where nodes share transaction data with their neighbors. Validators responsible for processing transactions and adding them to the blockchain, receive these transactions and execute them according to the rules encoded in programs. As transactions are executed, the state of the blockchain is updated, and successful transactions are recorded in blocks. These blocks are then successively added to the blockchain, ensuring the integrity and chronological order of transactions.

Coordination Between Nodes

To achieve consensus and maintain a consistent view of the blockchain, nodes within the Solana cluster coordinate using the Proof of History (PoH) mechanism. PoH provides a cryptographic clock that timestamps transactions, enabling nodes to agree on the time order of events without needing a traditional clock. This coordination ensures that transactions are processed and added to the blockchain in a deterministic and ordered manner, facilitating high throughput and low latency.

Maintaining Ledger Integrity and Consistency

Validators play a critical role in maintaining the integrity and consistency of the blockchain ledger. They continuously validate incoming transactions, ensuring that they adhere to the rules of the blockchain protocol and do not violate any security constraints. Validators also prevent double-spending by verifying that each transaction is valid and has not already been included in a previous block. In cases where conflicts or discrepancies arise, the consensus mechanism, which may include mechanisms such as Tower Byzantine Fault Tolerance (BFT), helps nodes reach an agreement on the state of the blockchain, resolving conflicts and ensuring consensus among all participants.

What is the role of clusters in scaling and performance?

Solana's clusters play an important role in achieving scalability and high performance by implementing a distributed architecture that optimizes resource utilization and removes potential bottlenecks.

Horizontal Scaling: Clusters allow the network to handle a growing number of transactions without sacrificing speed or efficiency. By distributing the workload across multiple validators and RPC nodes within each cluster, Solana effectively parallelizes transaction processing, maximizing throughput and reducing latency.

Fault Tolerance: Clusters facilitate fault tolerance and resilience by ensuring that no single point of failure can disrupt the entire network. In the event of node failure or network partitioning, other nodes within the same cluster can seamlessly pick up the slack, maintaining the network's integrity and availability. This redundancy and decentralization contribute to the robustness of Solana's architecture, making it more resistant to attacks and failures.

Efficient resource utilization: Solana's clusters enable efficient resource utilization by dynamically allocating computational resources based on demand. As transaction volume fluctuates, clusters can scale up or down accordingly, optimizing performance and cost-effectiveness. This elasticity ensures that Solana remains responsive and reliable even under varying workloads, making it well-suited for both small-scale applications and large-scale enterprise solutions.

Use Cases and Applications

Solana clusters support a wide range of real-world applications, from decentralized finance (DeFi) to gaming and NFTs.

DeFi (Decentralized Finance) Projects

The distributed architecture of Solana clusters ensures high throughput and resilience, enabling platforms like Serum to handle large transaction volumes efficiently. By distributing the workload across multiple validators and RPC nodes within each cluster, Solana ensures that DeFi transactions can be processed quickly and reliably, contributing to the overall scalability and performance of the network.

Gaming and NFTs

Solana's distributed infrastructure ensures that in-game transactions and NFT transfers are processed with minimal latency, providing gamers and collectors with a smooth and immersive experience. By leveraging Solana's clusters, gaming platforms like Star Atlas can offer real-time interactions and asset transactions, showcasing the platform's scalability and responsiveness in the gaming and NFT space.

Web3 Applications

Solana's distributed architecture enables platforms like Audius to maintain a resilient infrastructure that empowers content creators and users alike. By leveraging Solana's clusters, Audius can provide a censorship-resistant platform for sharing and discovering music content, demonstrating the platform's suitability for Web3 applications that prioritize decentralization and user empowerment.

Enterprise Solutions

Platforms like Mango Markets rely on Solana's clusters to process transactions quickly and securely, ensuring efficient price discovery and liquidity provision in the cryptocurrency markets. Solana's distributed architecture enables platforms like Mango Markets to offer institutional-grade trading infrastructure to users, highlighting the platform's reliability and scalability in enterprise settings.

Challenges and Limitations of Solana Clusters

Despite its advantages, Solana faces challenges in cluster management and security.

- **Cluster Management:** Managing a large number of nodes and ensuring their coordination can be complex. Effective tools and strategies are needed to maintain the network’s robustness.
- **Security Concerns:** As with any blockchain, Solana must address security threats such as Sybil attacks and network partitioning. Ongoing research and development are crucial to mitigate these risks.
- **Future Developments:** Future enhancements, such as improved tooling for developers and better node management solutions, are expected to address current limitations and further strengthen Solana’s position in the blockchain space.

8

Tokens in Solana

Overview

Tokens are central to Solana, facilitating value exchange and powering decentralized applications (dApps). They represent diverse assets like digital currencies, real-world items, and financial instruments. This chapter explores their role as digital assets denoting value, ownership, or access rights within Solana's blockchain ecosystem, including fungible cryptocurrencies and non-fungible tokens (NFTs).

Introduction

Tokens are fundamental building blocks in the Solana ecosystem, serving as the primary medium of value exchange and a vital component of decentralized applications (dApps). Tokens on Solana can represent a variety of assets, from digital currencies to real-world assets and complex financial instruments.
In blockchain technology, tokens are digital assets representing value, ownership, or access rights within a particular blockchain ecosystem. These tokens can be fungible, meaning they are identical and interchangeable with one another, like cryptocurrencies, or non-fungible, meaning they are unique and distinct, like digital collectibles or NFTs (non-fungible tokens).

How Tokens Function Within the Solana Ecosystem?

Tokens in the Solana ecosystem are essential instruments for various purposes, ranging from value transfer to accessing specific services within decentralized applications (dApps). Here's an overview of how they operate.

1. Creation and Management

- Tokens on Solana are often created using the Solana Program Library (SPL), which provides standardized token contracts known as SPL Tokens. These contracts define the rules and functionalities of the tokens, including minting, burning, transferring, and managing token supply.
- Developers use SPL to create new project tokens, ensuring compatibility and interoperability across different applications within the Solana ecosystem.

2. Value Transfer

- One of the primary functions of tokens in Solana is to facilitate the transfer of value between users and applications. This includes transferring tokens representing cryptocurrencies, such as SOL (Solana's native token) or any other SPL Token.
- The efficiency of Solana's network allows for fast and low-cost transactions, making it feasible for a wide range of financial applications.

3. Utility in dApps

- Tokens serve various utility purposes within decentralized applications. For instance, they can be used to pay for services, access premium features, or participate in governance decisions within a dApp.
- Many DeFi (Decentralized Finance) applications on Solana utilize tokens for lending, borrowing, staking, and yield farming, where users can earn rewards by providing liquidity or participating in specific protocols.

4. Incentivization

- Tokens are often used to incentivize user behavior and participation within the Solana ecosystem. For example, users might earn tokens as rewards for staking SOL to secure the network or for providing liquidity to a decentralized exchange.
- These incentives help to drive engagement and growth within the network, encouraging users to contribute to its security and functionality.

5. Asset Representation

- Tokens on Solana can represent ownership of various digital and real-world assets. This includes digital collectibles (NFTs), real estate, commodities, and other tangible or intangible assets.
- By tokenizing these assets, Solana enables fractional ownership, increased liquidity, and broader access to investment opportunities.

Token Standards in Solana

Token standards are essential protocols that define the rules and functionalities for creating and managing tokens within a blockchain ecosystem. In Solana, token standards ensure interoperability, security, and efficiency for tokens created and used within its network. These standards facilitate the development of decentralized applications (dApps) and services that rely on tokenized assets.

SPL (Solana Program Library) Token Standard

The Solana Program Library (SPL) is a collection of on-chain programs that Solana developers can use to build and interact with decentralized applications. One of the most prominent components of SPL is the SPL Token standard, which defines how tokens should be created, transferred, and managed within the Solana ecosystem. Here are the key features and functionalities of SPL tokens.

1. Token Creation

SPL Tokens are created using the 'spl-token' program, which provides a straightforward way to mint new tokens. Developers can specify the total supply, decimals (for fractional tokens), and initial distribution of tokens.
The creation process ensures that the tokens adhere to a standardized format, enabling compatibility across various dApps and wallets within the Solana network.

2. Token Management

The SPL token standard includes functions for minting new tokens, burning existing tokens, and freezing or thawing accounts. This gives developers control over the token lifecycle and helps manage the token economy.
Token accounts in SPL are represented by the 'spl-token' program, which maintains a ledger of all token balances and transactions.

3. Security Features

SPL Tokens include robust security features such as multi-signature (multisig) support, which requires multiple private keys to authorize transactions. This enhances security for high-value tokens and important operations.
The standard also supports various access control mechanisms, allowing token issuers to implement custom rules for token transfers and usage.

4. Interoperability

The SPL token standard ensures that tokens can be seamlessly integrated with other Solana-based services and applications. This includes decentralized exchanges (DEXs), wallets, and other financial protocols.
By adhering to a common standard, SPL tokens can easily be listed and traded on multiple platforms without requiring custom integration work.

Comparison with Other Blockchain Token Standards (e.g., ERC-20 on Ethereum)

Comparison Criteria	SPL Tokens	ERC-20 Tokens
Flexibility and Functionality	Core functionalities include token creation, transfer, and balance management. Benefit from Solana's high throughput and low transaction costs. Rapidly gaining adoption within the Solana ecosystem.	Core functionalities include token creation, transfer, and balance management. Widely used in the Ethereum ecosystem with a broad range of existing dApps and tools.
Transaction Speed and Costs	Leverage Solana's high transaction speed and low fees for fast and cost-effective transactions. Particularly suitable for applications requiring high scalability.	Ethereum often faces congestion and high gas fees, especially during peak network usage.
Development and Ecosystem	Integrated development environment with tools like Solana CLI and Anchor framework for building and deploying SPL tokens. Simplifies the process of development within the Solana ecosystem.	Supported by a vast array of development tools and resources in the Ethereum ecosystem. The abundance of documentation, tutorials, and community support.
Interoperability and Adoption	Designed for interoperability within the Solana ecosystem. Quickly integrated into a growing number of Solana-based projects and services.	Extensive adoption and interoperability within the Ethereum ecosystem. Compatible with numerous wallets, exchanges, and DeFi protocols.

Types of Tokens in Solana

Fungible Tokens

Fungible tokens are interchangeable and indistinguishable from one another. Each token has the same value and can be divided into smaller units. These tokens are primarily used for value transfer and can be easily traded or exchanged.

Examples and Use Cases

- **Cryptocurrencies:** Solana's native token, SOL, is a fungible token used for transaction fees, staking, and governance within the Solana network.
- **Stablecoins:** USDC and USDT issued on Solana are fungible tokens used for trading and as a store of value with minimal price volatility.
- **Utility Tokens:** Fungible tokens can also represent utility within a specific platform or ecosystem, such as access to services or products.

Non-Fungible Tokens (NFTs)

Non-fungible tokens (NFTs) are unique and cannot be exchanged on a one-to-one basis with other tokens. Each NFT has distinct properties and cannot be divided into smaller units. NFTs are commonly used to represent ownership of digital or physical assets.

Examples and Use Cases

- **Digital Art:** Artists create and sell unique digital artworks as NFTs, ensuring authenticity and ownership.
- **Collectibles:** Digital collectibles like trading cards, in-game items, and virtual real estate are popular NFT use cases.
- **Certification:** NFTs can be used to certify ownership or authenticity of physical assets like real estate, rare items, and more.

Utility Tokens

Utility tokens are designed to provide access to a specific product or service within a blockchain-based platform. They are not primarily used as a medium of exchange but rather as a functional tool within an ecosystem.

Examples and Use Cases

- **Access Tokens:** Tokens that grant access to a platform's features, such as storage space, computing power, or specific functionalities.
- **In-Platform Currency:** Utility tokens used within decentralized applications (dApps) for transactions, paying for services, or gaining benefits.

Governance Tokens

Governance tokens enable holders to participate in the decision-making processes of a decentralized organization or platform. They often grant voting rights on proposals, upgrades, and changes to the protocol or ecosystem.

Examples and Use Cases

- **Decentralized Autonomous Organizations (DAOs):** Governance tokens are used in DAOs to vote on governance proposals and influence the direction of the organization.
- **Protocol Governance:** Tokens that allow users to vote on changes to the protocol, such as adjustments to fees, features, or policies.

Security Tokens

Security tokens represent ownership in an asset, such as equity in a company, real estate, or other financial instruments. They are subject to securities regulations and provide investors with rights such as profit sharing, dividends, or voting power.

Examples and Use Cases

- **Equity Tokens:** Represent shares in a company, giving holders ownership rights and potential dividends.
- **Asset-Backed Tokens:** Tokens backed by physical assets like real estate, commodities, or other tangible properties.
- **Revenue Sharing Tokens:** Tokens that entitle holders to a share of the profits or revenue generated by a project or company.

Use Cases and Applications of Tokens in Solana

Applications of Fungible Tokens

DeFi (Decentralized Finance)

- **Stablecoins:** Fungible tokens on Solana are often used as stablecoins, which are pegged to fiat currencies like USD. For example, USDC and USDT are popular stablecoins used in various DeFi applications on Solana.
- **Lending and Borrowing:** Platforms like Solend and Larix allow users to lend and borrow assets. Fungible tokens serve as collateral for loans and as the assets being lent or borrowed.
- **Decentralized Exchanges (DEXs):** DEXs like Serum and Raydium utilize fungible tokens for trading. Users can swap between different tokens, providing liquidity and earning fees in the process.
- **Yield Farming:** Yield farming protocols incentivize users to provide liquidity by rewarding them with fungible tokens. These tokens often represent a share of the profits generated by the protocol.

Gaming

- **In-Game Currency:** Fungible tokens can be used as in-game currencies, enabling players to purchase items, upgrades, and other digital assets. Games like Star Atlas leverage fungible tokens for their in-game economy.
- **Reward Systems:** Players can earn fungible tokens as rewards for achieving milestones or completing tasks, which can then be traded or used within the game's ecosystem.

Applications of NFTs

Digital Art

- **Art Marketplaces:** Platforms like Solanart and DigitalEyes allow artists to mint, showcase, and sell their digital artworks as NFTs. Collectors can purchase unique pieces, ensuring ownership and authenticity through the blockchain.
- **Royalties:** Smart contracts on Solana enable artists to receive royalties from secondary sales, ensuring they benefit from the increasing value of their work.

Collectibles

- **Virtual Collectibles:** NFTs represent unique virtual items, such as trading cards, virtual pets, and memorabilia. Projects like SolPunks and Thugbirdz offer unique digital collectibles that users can trade or showcase.
- **Limited Editions:** Brands and celebrities release limited edition NFTs, creating a digital scarcity that drives value and demand among fans and collectors.

Emerging Trends and Innovative Projects on Solana Tokens

Metaverse

- **Virtual Real Estate:** Projects like Portals and Solice are building virtual worlds where users can own, develop, and trade virtual real estate using NFTs and fungible tokens.

- **Interoperable Assets:** In the metaverse, assets like avatars, accessories, and virtual goods can be tokenized and transferred across different platforms, providing a seamless user experience.

Social Tokens

- **Creator Economy:** Social tokens empower creators and influencers to tokenize their brands. Fans can purchase these tokens to access exclusive content, participate in decision-making, and receive rewards. Platforms like Rally are exploring this model on Solana.

Tokenized Real-World Assets

- **Real Estate:** Companies are beginning to tokenize physical real estate, allowing fractional ownership and easier trading of property shares. This opens up real estate investment to a broader audience.
- **Commodities and Securities:** Tokenizing commodities like gold or securities enables easier trading and ownership transfer, with improved transparency and reduced barriers to entry.

Decentralized Identity

- **Self-Sovereign Identity:** Using tokens, users can maintain control over their digital identities. Projects like Civic are developing solutions that leverage blockchain technology for secure, decentralized identity management.

9

Creating Tokens Using CLI

Overview

In this chapter, you will learn how to create and manage a custom token on the Solana blockchain using the command-line interface (CLI). This includes setting up a Solana wallet, installing the necessary tools, creating and minting tokens, and managing various token functions such as transfers, limiting supply, and setting permissions.

Introduction

One of the essential features of any blockchain ecosystem is the ability to create and manage tokens, which can represent various assets, currencies, or utilities within the network.
In this chapter, we will learn how to create a token on the Solana blockchain using the command-line interface (CLI).

Prerequisites

Before moving ahead, make sure you have the following tools installed on your system.

- Solana CLI
- Rust
- Solana Wallet

How do you create a token using CLI?

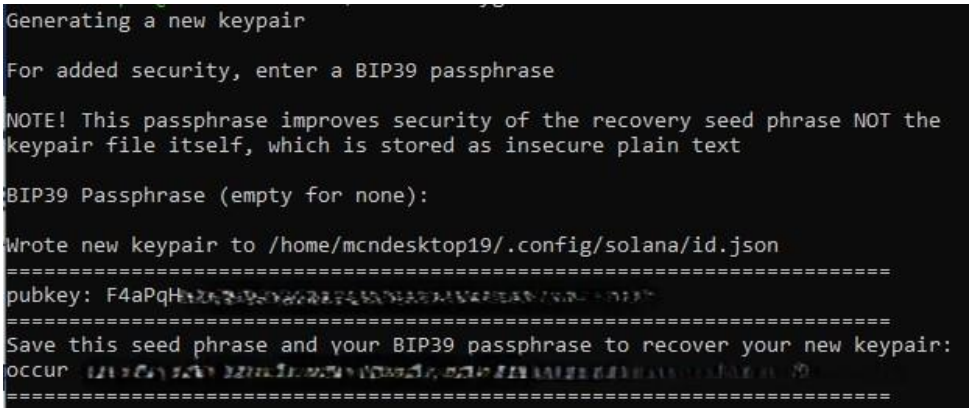
Following are the steps that you need to follow to create a token using CLI.

Step 1. Setting Up a Solana Wallet

If you have already created a wallet, you can skip this step. In your terminal, enter the following command.

```
Solana-keygen new
```

This will create a key pair. Keep your passphrase in a secure place.



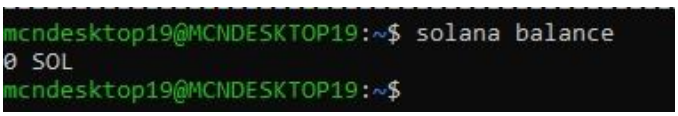
If you already have a wallet and want to create another, you can forcefully do this by following the command.

```
Solana-keygen new --force
```

This will create a new key pair forcefully.

You can check the balance of your wallet by following command.

```
Solana balance
```



You can also airdrop some tokens by following the command.

```
solana airdrop 2 F4aPqHpcrZXDyb6MBdBLG**************** --url devnet
```

This will airdrop 2 tokens in your devnet account having the above public key. Sometimes the airdrop may fail due to the limit, you can try again, and it will work.

If you want to get information about your account, use this command.

```
solana account ************XDyb6MBdBLGr********
```

This will give you the information associated with the above account.

Step 2. Installing the Token Program CLI

Now that you have your wallet ready. Let's install the SPL token CLI tool, which is necessary for creating and managing tokens using CLI.

```
cargo install spl-token-cli
```

If you are getting any pkg-config error, you should install pkg-config by following the command.

```
sudo apt install pkg-config
```

This will resolve this error.

To verify whether the spl token program is installed properly, you can check the version by following the command.

```
spl-token --version
```

This ensures we have properly installed the spl token program.

Step 3. Creating the Token

Now we have a wallet and token program, it's time to create our own token. To do so follow the below command.

```
spl-token create-token --url devnet
```

This will create a token using the existing spl token program. You will get a token address in the terminal, which represents your new token. Store this in a safe place.

Step 4. Creating a Token Account

Now that our token is created. we need to create an associated token account to hold our tokens. To create an account for our token, use the following command.

```
spl-token create-account <CREATED_TOKEN> --url devnet
```

This will create an account for the above token. Make sure to use your correct token address in the above command.

Now you can check the balance of the created token.

```
spl-token balance <CREATED_TOKEN> --url devnet
```

Step 5. Minting Tokens

Now you can mint new tokens to the created account by the following command.

```
spl-token mint <TOKEN_ADDRESS> <amount> --url devnet
```

This will mint the given number of tokens in your token account.

To check the tokens in the supply, you can use the following command.

```
spl-token supply <CREATED_TOKEN> --url devnet
```

This command will show you the tokens that are currently present in the supply.

How to manage the created token in Solana?

Now that we have created our token, we can perform various actions on the created token.

Limit the supply of the token

Although we are currently creating tokens for our development and testing purposes, if in the future we create our token, it will be important to limit the supply of the token as an unlimited supply is not good for your token value and reputation and there are many more things to it.

So, to limit the supply of your token, you can use the following command.

```
spl-token authorize <CREATED_TOKEN> mint --disable --url devnet
```

This command will disable the further minting of tokens and limit the supply. Currently, our token is on devnet so, I am using devnet in the command, you can change it accordingly.

Transferring Tokens

We can transfer our created token to others by following the command.

```
spl-token transfer <TOKEN_ADDRESS> <AMOUNT> <RECIPIENT_WALLET_ADDRESS>
--url devnet
```

If you are having issues like unfunded recipient, you can use the following command.

```
spl-token transfer <TOKEN_ADDRESS> <AMOUNT> <RECIPIENT_WALLET_ADDRESS>
--url devnet --allow-unfunded-recipient --fund-recipient
```

This will solve the unfunded recipient issue.

Make sure to verify the transaction to ensure tokens are transferred correctly.

Managing the Token by setting permissions

You can set various permissions on your token, such as giving mint authority, and freezing or thawing accounts. Before setting these permissions make sure you were to do this and give permissions to only the most trusted person.

Freeze an Account: Freezing an account prevents any further transactions involving the token from that account. To freeze use the following command.

```
spl-token authorize <TOKEN_ACCOUNT> freeze <AUTHORITY_ADDRESS> --url
devnet
```

In the above command, replace <TOKEN_ACCOUNT> with the account to be frozen and <AUTHORITY_ADDRESS> with the authority's public key.

Thaw an Account: Thawing reverses freezing action, allowing transactions to resume. To thaw use the following command.

```
spl-token authorize <TOKEN_ACCOUNT> thaw <AUTHORITY_ADDRESS> --url
devnet
```

In the above command, replace <TOKEN_ACCOUNT> with the account to be thawed and <AUTHORITY_ADDRESS> with the authority's public key.

Mint Authority of an Account: If you want to give mint authority of your token to someone else, you can use the following command.

```
spl-token authorize <TOKEN_ADDRESS> mint <NEW_AUTHORITY>
```

In the above command, replace <NEW_AUTHORITY> with the new authority's public key.

Burning Tokens: Burning is useful for managing token supply and maintaining value. To burn tokens and reduce the supply, you can use the following command.

```
spl-token burn <TOKEN_ACCOUNT> <AMOUNT> --url devnet
```

This will burn the given number of tokens from the above token account. Note that, the tokens will be burned from the remaining tokens of your token account. If you previously sent some tokens to others, their token will remain unaffected by this action.

10

Languages Supported by Solana

Overview

This chapter explores how developers use languages like C, C++, and Rust to build efficient and secure applications on Solana. It also covers the Anchor framework, which simplifies creating decentralized apps (dApps) on the platform.

Introduction

Solana supports C, C++, and Rust for blockchain development. These languages offer different advantages and are popular choices for building efficient and secure applications on the Solana blockchain. Additionally, Solana provides the Anchor framework to simplify smart contract development, making it easier for developers to create decentralized applications (dApps) with Solana.

Language Supported by Solana

Solana blockchain supports three languages: C, C++, and Rust. Let's understand what the benefits and limitations of these languages are.

Language	Benefits	Limitations
C	Efficiency: Ideal for resource-intensive tasks Scalability: Complements Solana's architecture Compatibility: Widespread support simplifies integration	Learning Curve: Steep for developers new to C Security: Risks like buffer overflows must be addressed
C++	Performance: High-performance capabilities Rich Ecosystem: Vast libraries and frameworks Flexibility: Custom solutions and optimization	Learning Curve: Steeper compared to some other languages Memory Management: Requires manual management Interoperability: Consideration for seamless integration
Rust	Performance: Zero-cost abstractions for high-performance Safety: The ownership model enhances security Concurrency: Built-in support for efficient code	Learning Curve: Some complexity, especially for newcomers Tooling: Specialized tools may be less mature

Projects or Libraries Using Rust in Solana Development

- **Serum DEX (Decentralized Exchange):** Serum, a decentralized exchange built on Solana, leverages Rust extensively for its smart contracts and backend infrastructure. Rust's performance and safety features are crucial for handling the high-volume trading activity on Serum, ensuring reliability and security for users.
- **Anchor:** Anchor is a framework for building Solana smart contracts in Rust. It provides developers with high-level abstractions and tooling to streamline the development process. Using Rust, Anchor enables developers to write expressive and efficient smart contracts while leveraging Solana's unique capabilities.
- **Solana Runtime:** The Solana runtime is written in Rust, underscoring the platform's commitment to the language. By utilizing Rust, Solana benefits from its performance optimizations and safety guarantees, laying a strong foundation for building decentralized applications and scaling the network.

We can say that Rust will be the first choice for the Solana development as Rust's integration with Solana opens up a world of possibilities for blockchain developers. With its focus on performance, safety, and concurrency, Rust empowers developers to build fast, secure, and scalable decentralized applications on the Solana blockchain. So we move forward with the Anchor framework. Let's install the Anchor.

Anchor Setup

The Anchor framework for Solana is a powerful tool designed to simplify and accelerate the development of decentralized applications (dApps) on the Solana blockchain. Built with developers in mind, Anchor provides a robust and developer-friendly environment for building, deploying, and managing smart contracts on Solana. At its core, Anchor leverages Rust programming language, a language known for its performance, safety, and concurrency features. By using Rust, Anchor offers developers a familiar and powerful ecosystem to build secure and efficient smart contracts for the Solana blockchain.

Installation and Setup

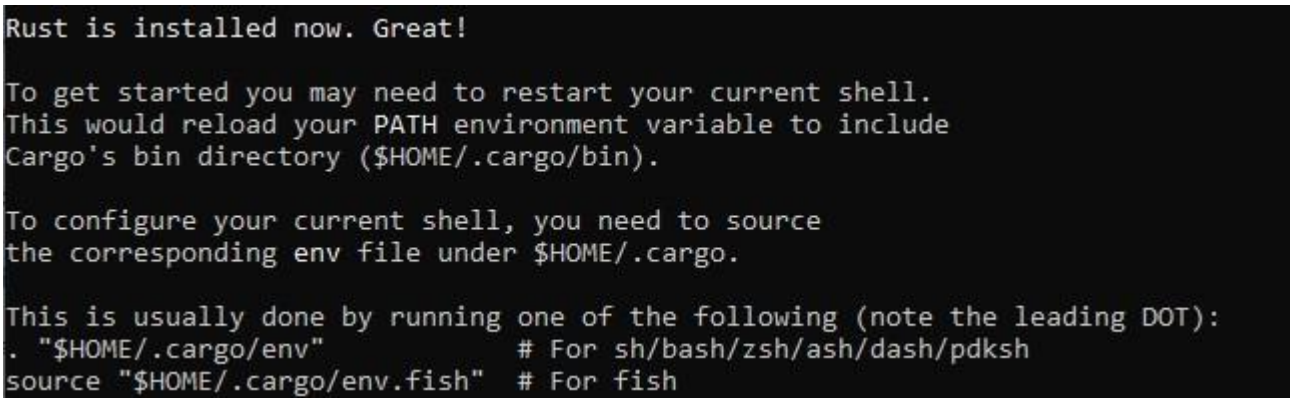
Getting started with Anchor is straightforward. To install Anchor, follow these simple steps:

Install Rust

Before installing Anchor, ensure that Rust is installed on your system. Rust can be installed using the Rustup tool, which is available for Windows, macOS, and Linux. For WSL, macOS, Linux, and Unix-like OS, run the following command to install Rust.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
```

Output



Now restart your current shell to reload the PATH environment variable to include Cargo's bin directory or run the following command to achieve the same.

```
. "$HOME/.cargo/env"
```

2. Install Anchor CLI

Once Rust is installed, the next step is to install the Anchor Command Line Interface (CLI). For this first, we install AVM(Anchor Version Manager) by running the following command.

cargo install --git https://github.com/coral-xyz/anchor avm --locked --force

Once installation is done run the following command for the confirmation. It will display the current installed version.

```
avm --version
```

After this, run the following command to install the anchor.

```
avm install latest
avm use latest
```

You can replace the 'latest' with a specific version like '0.29.0' also as your requirement. Once it is done, run the following command to verify the installation.

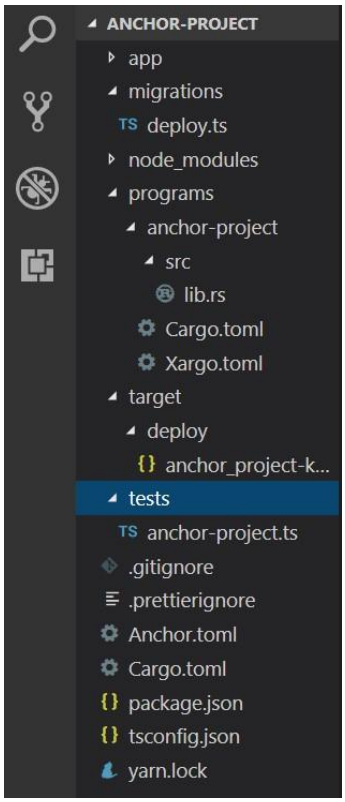
```
anchor --version
```

Create a new anchor project

Run the following command to create a new anchor workspace.

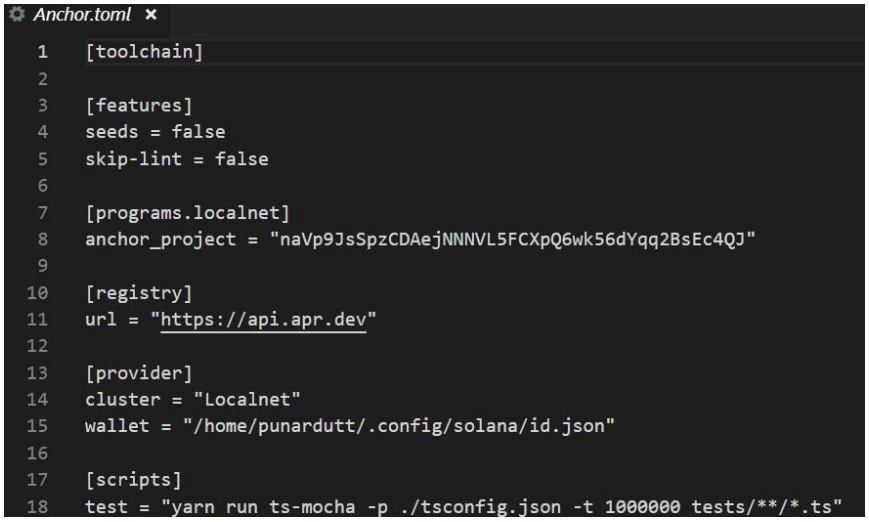
```
anchor init <YOUR_PROJECT_NAME>
```

it will create a new workspace that has the following structure.



Let's understand the workings of these directories one by one.

- **The anchor directory:** It holds the most recent program logs, and a local ledger used for testing.
- **The app directory:** An empty directory designed to accommodate your front end if you choose a monorepo configuration.
- **The programs directory:** This folder houses your programs. At first, it might have only one program named <YOUR_PROJECT_NAME>. This program comes with a pre-existing lib.rs file containing sample code.
- **The tests directory:** The location of your end-to-end (E2E) tests is already provided, along with a file that tests the sample code found in the programs/<YOUR_PROJECT_NAME> directory.
- **The migrations directory:** You have the option to keep your deployment and migration scripts for your programs here.
- **The Anchor.toml file:** This document establishes settings that apply to your programs across the entire workspace. Initially, it sets up



- **[programs.localnet]:** The addresses of your programs within the localnet.
- **[registry]:** A registry where your program will be pushed.
- **[provider]:** A provider to be used in your tests.
- **[scripts]:** Anchor automatically runs scripts for you, including the test script, during "anchor test" execution. You can execute your scripts using "anchor run <script_name>".

Anchor Commands

- **account:** Fetch and deserialize an account using the IDL provided
- **build:** Builds the workspace
- **cluster:** Cluster commands
- **deploy:** Deploys each program in the workspace
- **expand:** Expands the macros of a program or the workspace
- **help:** Prints this message or the help of the given subcommand(s)
- **idl:** Commands for interacting with interface definitions
- **init:** Initializes a workspace
- **migrate:** Runs the deploy migration script
- **new:** Creates a new program
- **shell:** Starts a node shell with an Anchor client setup according to the local config
- **test:** Runs integration tests against a local network
- **upgrade:** Upgrades a single program. The configured wallet must be the upgrade authority
- **verify:** Verifies the on-chain bytecode matches the locally compiled artifact.

We have done with the setup of the anchor framework, in the next article we will learn to write a program(smart contract) and test using the anchor framework.

11

Developing and Deploying with Solana Playground

Overview

In this chapter, you will learn how to leverage the Solana Playground to develop, deploy, and test blockchain programs without needing traditional Integrated Development Environments (IDEs). It provides a step-by-step guide on creating and managing Solana programs, emphasizing the ease and flexibility of the playground tool for blockchain development.

Introduction

Solana Playground is a web-based integrated development environment (IDE) designed to simplify the process of building, testing, and deploying smart contracts on the Solana blockchain. It has an easy-to-use interface and a variety of tools for both new and experienced developers, making it a versatile platform for blockchain development.

Key Features of Solana Playground

The following are the main features of Solana playground:-

- **Interactive Interface:** Solana Playground offers an intuitive interface that allows developers to write, compile, and deploy smart contracts directly from their web browser. This eliminates the need for lengthy local setup and configuration, freeing developers to concentrate on building and testing code.
- **Preconfigured Environment:** The playground provides a preconfigured development environment, removing the hassle of setting up the necessary tools and dependencies. This feature is particularly beneficial for newcomers who might find the initial setup process daunting.
- **Built-in Templates:** Solana Playground includes several templates and examples to help developers get started quickly. These templates cover common use cases and standard contract structures, providing a solid foundation for new projects.
- **Real-time Collaboration:** One of the standout features of Solana Playground is its support for real-time collaboration. Multiple users can work on the same project simultaneously, making it an excellent tool for team-based development. This feature enhances productivity and enables collective problem-solving.
- **Integration with Solana CLI:** Solana Playground integrates with the Solana Command Line Interface (CLI) to allow sophisticated deployment and management functions. This enables developers to carry out difficult tasks and manage their projects more efficiently.

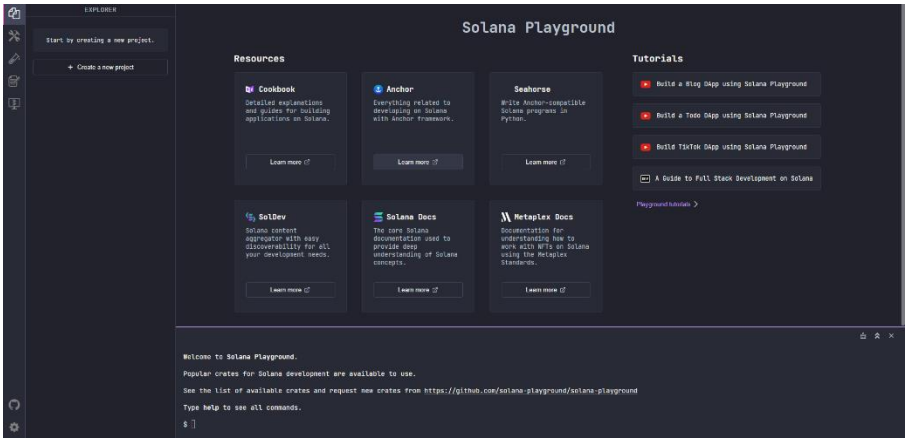
Advantages of Using Solana Playground

The following are the main advantages of Solana playground:-

- **Ease of Use:** The playground's user-friendly interface makes it accessible for developers of all skill levels. Whether you're a seasoned developer or a newcomer to blockchain technology, Solana Playground offers an intuitive platform to work on.
- **Speed:** Quick setup and immediate access to development tools speed up the development process. With everything ready to go from the start, you can focus on coding and testing rather than spending time on configuration.
- **Collaboration:** Real-time collaboration features enhance teamwork and collective problem-solving. Working with others in real-time enables rapid feedback and faster project iteration.
- **Learning Resource:** For beginners, Solana Playground is a wonderful educational tool. It provides a hands-on environment to understand Solana’s ecosystem and smart contract development, making the learning curve less steep.

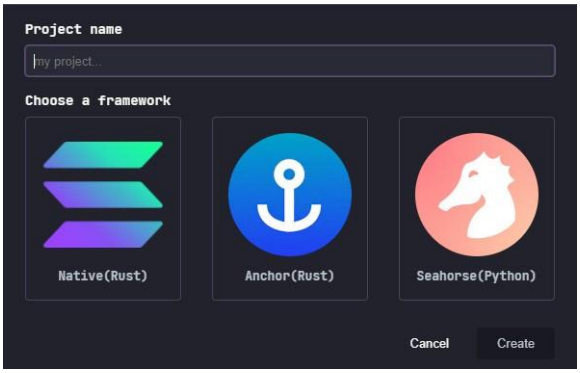
Getting Started with Solana Playground

To begin using Solana Playground, visit the official Solana Playground website - beta.solpg.io/. The website looks like the following.



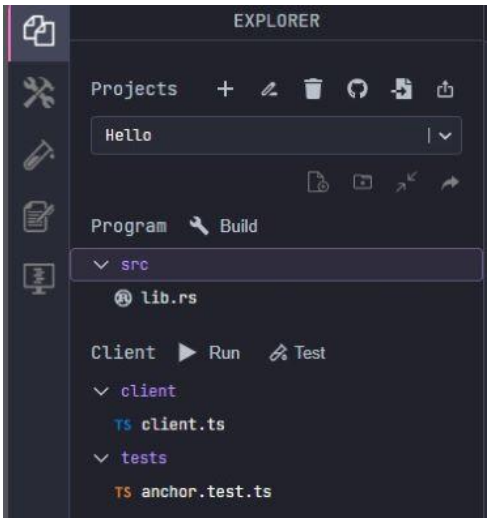
Creating a New Project

Start a new project by selecting from available templates that are Native (Rust), Anchor (Rust), or Seahorse (Python), name your project, and click Create. Here, I am selecting Anchor(Rust), you can choose according to your needs.



Writing Smart Contracts

After creating a project, a program structure gets created which looks like the below-



Here, we have three main folders 'src', 'client', and 'tests'. In the 'src' folder, we have a file named 'lib.rs'. In the 'client' folder, we have a file named 'client.ts' and in the 'tests' folder, we have a file named 'anchor.test.ts'. Initially, we have a default program written in these files. We can modify these files according to our requirements. In this article, we will not be creating our program but rather using the default program to explore how the playground works.

In lib.rs-

```
use anchor_lang::prelude::*;
// This is your program's public key and it will update
// automatically when you build the project.
declare_id!("11111111111111111111111111111111");
#[program]
mod hello_anchor {
    use super::*;
    pub fn initialize(ctx: Context<Initialize>, data: u64) ->
Result<()> {
        ctx.accounts.new_account.data = data;
        msg!("Changed data to: {}!", data); // Message will show up in
the tx logs
        Ok(())
    }
}
#[derive(Accounts)]
pub struct Initialize<'info> {
    // We must specify the space in order to initialize an account.
    // First 8 bytes are default account discriminator,
    // next 8 bytes come from NewAccount.data being type u64.
    // (u64 = 64 bits unsigned integer = 8 bytes)
    #[account(init, payer = signer, space = 8 + 8)]
    pub new_account: Account<'info, NewAccount>,
    #[account(mut)]
    pub signer: Signer<'info>,
    pub system_program: Program<'info, System>,
}
#[account]
pub struct NewAccount {
    data: u64
}
```

This program demonstrates how to use the Anchor framework to create a new account and initialize it with a value on the Solana blockchain. It shows the basic structure of an Anchor program, including how to define instructions, specify account constraints, and log messages for transaction tracing. This example is a good starting point for understanding how to build more complex Solana programs using Anchor.

In Client.ts,

```
console.log("My address:", pg.wallet.publicKey.toString());
const balance = await pg.connection.getBalance(pg.wallet.publicKey);
console.log(`My balance: ${balance / web3.LAMPORTS_PER_SOL} SOL`);
```

In the above file, we have a typescript file that logs my address and my balance.

In anchor.tests.rs

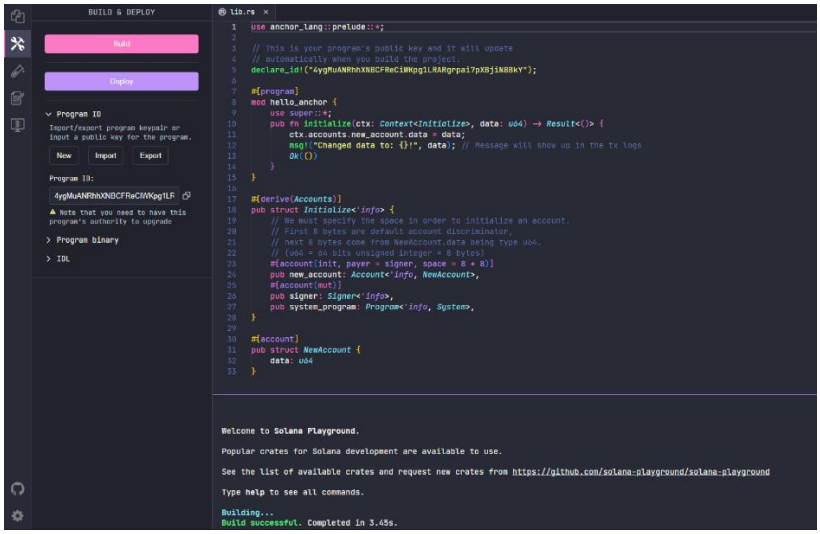
```
// No imports needed: web3, anchor, pg and more are globally available
describe("Test", () => {
    it("initialize", async () => {
```

```
    // Generate keypair for the new account
    const newAccountKp = new web3.Keypair();
    // Send transaction
    const data = new BN(42);
    const txHash = await pg.program.methods
      .initialize(data)
      .accounts({
        newAccount: newAccountKp.publicKey,
        signer: pg.wallet.publicKey,
        systemProgram: web3.SystemProgram.programId,
      })
      .signers([newAccountKp])
      .rpc();
    console.log(`Use 'solana confirm -v ${txHash}' to see the logs`);
    // Confirm transaction
    await pg.connection.confirmTransaction(txHash);
    // Fetch the created account
    const newAccount = await pg.program.account.newAccount.fetch(
      newAccountKp.publicKey
    );
    console.log("On-chain data is:", newAccount.data.toString());
    // Check whether the data on-chain is equal to local 'data'
    assert(data.eq(newAccount.data));
  });
});
```

In the above code, we have a test case for a Solana program using the Anchor framework. It initializes a new account with a value of `42`, sends the transaction, and confirms it. After fetching the account data from the blockchain, it checks if the on-chain data matches the expected value. Finally, it asserts that the stored data is correct, ensuring the smart contract functions as intended.

Building the program in the playground

To build this program, click the second option from the left navbar. Here, we have the Build button in the Explorer. Click on build to build your program, if there are no errors in your program, your program will build successfully.



Here, we have successfully built our program. Here, if you notice, our program public key is updated as we build the program.

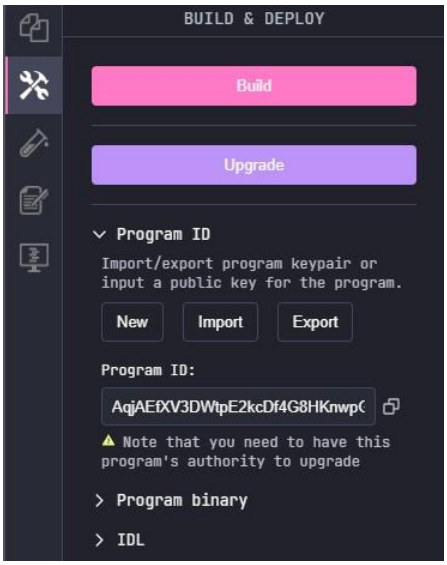
Deploying to the Solana Network using the playground

Once our program is successfully built, we can deploy it to the Solana devnet, testnet, or mainnet.

To deploy the program, click on the Deploy button. After a while, you'll notice that your program has been deployed, and a few SOLs have deducted your balance. This deduction is the cost of program deployment.

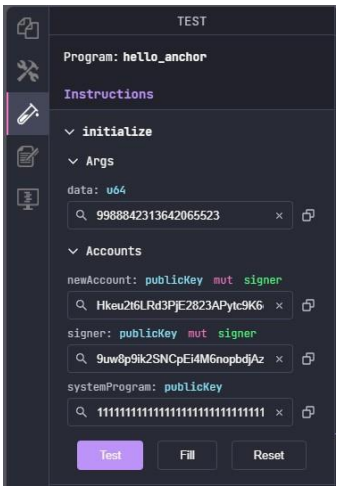
```
Deployment successful. Completed in 41s.
```

Once the program is deployed on the chain, you will have a button to upgrade. This is used when you want to update or upgrade the program after the deployment. You can make changes and click on Upgrade to update the on-chain program.

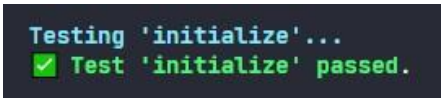


Testing the program using the playground

Now, we can test this on-chain program, by clicking on the third option from the left navbar. Here, we have the necessary data for the testing. Before testing, we will need the values for Args, Accounts, etc. we can either fill it up by ourselves or we can click on the Fill button. the values will be automatically filled, and we are ready to test our program.



Now, click the Test button, and test cases written in the anchor.test.ts file will be executed. If the test cases pass successfully, the playground terminal will look like the below.



If any test case fails during this, the playground terminal will show the error related to the failed test case.

If you click on Accounts and then click the Fetch All button, you will have the account generated with the same data value we provided in our test.

Now, we have successfully developed, deployed, and tested the program using the Solana playground. Similarly, you can use the playground to create programs according to your requirements without using the IDEs.

C#Corner

OUR MISSION

Free Education is Our Basic Need! Our mission is to empower millions of developers worldwide by providing the latest unbiased news, advice, and tools for learning, sharing, and career growth. We're passionate about nurturing the next young generation and help them not only to become great programmers, but also exceptional human beings.

ABOUT US

CSharp Inc, headquartered in Philadelphia, PA, is an online global community of software developers. C# Corner served 29.4 million visitors in year 2022. We publish the latest news and articles on cutting-edge software development topics. Developers share their knowledge and connect via content, forums, and chapters. Thousands of members benefit from our monthly events, webinars, and conferences. All conferences are managed under Global Tech Conferences, a CSharp Inc sister company. We also provide tools for career growth such as career advice, resume writing, training, certifications, books and white-papers, and videos. We also connect developers with their potential employers via our Job board. Visit C# Corner

MORE BOOKS