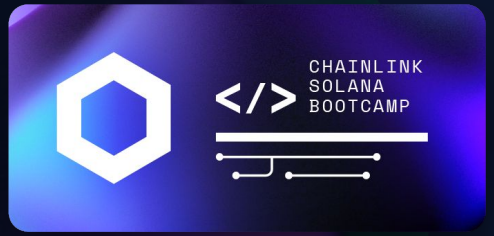

Chainlink
SOLANA
Blockchain Developer Bootcamp
Day 2



Chainlink

Solana Developer Bootcamp

Day 2

Introduction to Anchor

The Oracle Problem and Chainlink

Harry Papacharissiou

Developer Advocate, Chainlink Labs

harry@chainlinklabs.com

@pappas9999

harry-papacharissiou

Chainlink

Solana Blockchain Developer Bootcamp Outline

Day 1: Introduction

Learn about Solana, it's architecture and programming model. Build some example smart contracts such as a simple GM, and a token contract.

Day 2: Anchor and Chainlink

Learn how to use the Anchor framework, as well as how to use Chainlink Price Feeds to get pricing data in your Solana smart contracts

Chainlink

Housekeeping Rules

- 3 hour session, broken up into sections, with breaks
- Presentation, followed by questions, demo and exercises
- Questions can be asked at the end of each section
- Zoom: If you need help during a practical exercise, raise your hand and an instructor will eventually get to you and send you an invite to a private breakout session 
- Technical questions can be asked in the chat or in the #developer-bootcamp channel in the Chainlink Discord

Day 2 Exercises

- Exercise requirements:
  - NodeJS
  - Rust
  - Solana Tool Suite
  - Visual Studio Code
  - Anchor

Agenda: Day 2

1. **Introduction to Anchor**
2. Exercise: GM Anchor Program
3. Exercise: Solana Social
4. The Oracle Problem and Chainlink
5. Exercise: Price Feeds Consumer Program
6. Where You Can Learn More
7. Summary

Introduction to Anchor


RAILS

Chainlink

Chainlink

Anchor

- A framework for quickly building secure Solana programs
- Reduces coding efforts by generating various boilerplate code
- Handles security checks

Chainlink

Anchor Projects

- Serum
- Synthetify
- Mango IDO
- Metaplex Candy Machine
- SolFarm
- Saber
- Solend
- Jet Protocol
- Quarry
- PsyOptions
- Arrow Protocol
- Hubble Protocol
- + many more

Anchor Tools

- Rust smart contracts
- Interface Description Language
- Client Generators
- CLI

Anchor Workflow

Chainlink

Chainlink

Anchor High Level Overview

- Anchor program consists of three parts:
  - Program module
    - Where the program logic is written
  - Accounts structs
    - Where accounts are validated
  - Declare ID macro
    - Stores the address of the program

Chainlink

Anchor High Level Overview

```
use anchor_lang::prelude::*;

// declare an id for your program
declare_id!("Fg6PaFpoGXkYsidMpWTK6W2BeZ7FEfcYkg476zPFsLnS");

// write your business logic here
#[program]
mod hello_anchor {
    use super::*;
    pub fn initialize(_ctx: Context<Initialize>) -> ProgramResult {
        Ok(())
    }
}

// validate incoming accounts here
#[derive(Accounts)]
pub struct Initialize {}
```

Anchor: Accounts

- Where you define which accounts your instruction expects, and which constraints these accounts should follow
  - Types
    - Various account types
  - Constraints
    - Handles security and access control for accounts

Anchor: Account Type

- Used when an instruction is interested in the deserialized data of an account.

Chainlink

Anchor: Accounts

```
use anchor_lang::prelude::*;

declare_id!("Fg6PaFpoGXkYsidMpWTK6W2BeZ7FEfcYkg476zPFsLnS");

#[program]
mod hello_anchor {
    use super::*;
    pub fn set_data(ctx: Context<SetData>, data: u64) -> ProgramResult {
        ctx.accounts.my_account.data = data;
        Ok(())
    }
}

#[account]
#[derive(Default)]
pub struct MyAccount {
    data: u64
}

#[derive(Accounts)]
pub struct SetData<'info> {
    #[account(mut)]
    pub my_account: Account<'info, MyAccount>
}
```

Chainlink

Anchor: Constraints

- Accounts aren't dynamic enough to handle all the security checks required
- Constraints can be added to dynamically perform checks and validation

```
#[account(<constraints>)]
pub account: AccountType
```

```
#[derive(Accounts)]
pub struct SetData<'info> {
    #[account(mut)]
    pub my_account: Account<'info, MyAccount>,
    #[account(
        constraint = my_account.mint == token_account.mint,
        has_one = owner
    )]
    pub token_account: Account<'info, TokenAccount>,
    pub owner: Signer<'info>
}
```

Chainlink

Anchor: Program Module

- Where business logic is defined
- Write functions which can then be called by clients or other programs
- Each defined endpoint takes a ***Context*** type as an argument
    - Provides access to the accounts of the executing program and the Program ID

```
#[program]
mod hello_anchor {
    use super::*;
    pub fn set_data(ctx: Context<SetData>, data: u64) ->
ProgramResult {
        if ctx.accounts.token_account.amount > 0 {
            ctx.accounts.my_account.data = data;
        }
        Ok(())
    }
}
```

Anchor: Instruction Data

- Can be added by adding arguments to the function after the context argument
- Anchor will automatically deserialize the instruction data into the arguments

Chainlink

Anchor: Instruction Data

```
#[program]
mod hello_anchor {
    use super::*;
    pub fn set_data(ctx: Context<SetData>, val: u64) ->
ProgramResult {
        ctx.accounts.data.val = val;
        Ok(())
    }
}

#[account]
#[derive(Default)]
pub struct Data {
    pub val: u64
}
```

Chainlink

Anchor: IDL

- Generates an IDL specification for programs
  - Similar to an EVM-based ABI
- Used by clients to be able to interact with deployed Anchor-based programs
- Greatly simplifies calling functions, sending and receiving data
- Acts as the connecting glue between the on-chain program, and off chain clients

Anchor: IDL

```
use anchor_lang::prelude::*;
declare_id!("Fg6PaFpoGXkYsidMpWTK6W2BeZ7FEfcYkg476zPFsLnS");

#[program]
mod basic_1 {
   use super::*;

   pub fn initialize(ctx: Context<Initialize>, data: u64) -> ProgramResult {
       let my_account = &mut ctx.accounts.my_account;
       my_account.data = data;
       Ok(())
   }
}

#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(init, payer = user, space = 8 + 8)]
    pub my_account: Account<'info, MyAccount>,
    #[account(mut)]
    pub user: Signer<'info>,
    pub system_program: Program<'info, System>,
}

#[account]
pub struct MyAccount {
    pub data: u64,
}
```

Anchor: IDL

Chainlink

```
{
  "version": "0.1.0",
  "name": "basic_1",
  "instructions": [
    {
      "name": "initialize",
      "accounts": [
        {"name": "myAccount", "isMut": true,"isSigner": true},
        {"name": "user", "isMut": true,"isSigner": true}
        {"name": "systemProgram", "isMut": true,"isSigner": true}],
      "args": [
        {
          "name": "data",
          "type": "u64"
        }
      ]}],
  "accounts": [
    {
      "name": "MyAccount",
      "type": {
        "kind": "struct",
        "fields": [
          {"name": "data","type": "u64"}
        ]
      }
    }
  ]
}
```

Chainlink

Anchor: Generating a Client using IDL

```
// Read the generated IDL.
const idl = JSON.parse(
  require("fs").readFileSync("./target/idl/basic_1.json", "utf8")
);

// Address of the deployed program.
const programId = new anchor.web3.PublicKey("<YOUR-PROGRAM-ID>");

// Generate the program client from IDL.
const program = new anchor.Program(idl, programId);

// The Account to create.
const myAccount = anchor.web3.Keypair.generate();

// Create the new account and initialize it with the program.
await program.rpc.initialize(new anchor.BN(1234), {
  accounts: {
    myAccount: myAccount.publicKey,
    user: provider.wallet.publicKey,
    systemProgram: SystemProgram.programId,
  },
  signers: [myAccount],
});
```

Chainlink

Anchor: Building and Deploying Programs

```
anchor build
```

- Builds programs in the workspace targeting Solana's BPF runtime and emitting IDLs in the target/idl directory

Chainlink

Anchor: Building and Deploying Programs

```
anchor deploy [cluster]
```

- Deploys all programs in the workspace to the configured cluster.
- Optional cluster parameter ‘provider.cluster’
  - devnet, testnet, mainnet-beta, localnet

Chainlink

Anchor: Building and Deploying Programs

```
anchor new [program-name]
```

- Initializes a project workspace with the following structure.
  - *Anchor.toml*: Anchor configuration file.
  - *Cargo.toml*: Rust workspace configuration file.
  - *package.json*: JavaScript dependencies file.
  - *programs/*: Directory for Solana program crates.
  - *app/*: Directory for your application frontend.
  - *tests/*: Directory for JavaScript integration tests.
  - *migrations/deploy.js*: Deploy script.

Chainlink

Anchor: Building and Deploying Programs

```
anchor init
```

- Creates a new program in the workspace's programs/ directory initialized with boilerplate code

Anchor: Building and Deploying Programs

```
anchor verify [program-id]
```

- Verifies the on-chain bytecode matches the locally compiled artifact.

Chainlink

Testing With Anchor

```
anchor test
```

- Run an integration test suite against the configured cluster, deploying new versions of all workspace programs before running them.
  - Looks for a [test] script definition in the Anchor.toml file
- If the configured network is a localnet, then automatically starts the local network and runs the test.

Chainlink

Exercise 1: GM Anchor Program

- Program that accepts a name parameter, says GM to it and stores the name in an account
- 30 minutes

Chainlink

Exercise 2: Solana Social Program

- Social Media posts on Solana
  - Create post
  - Update post
- 60 minutes

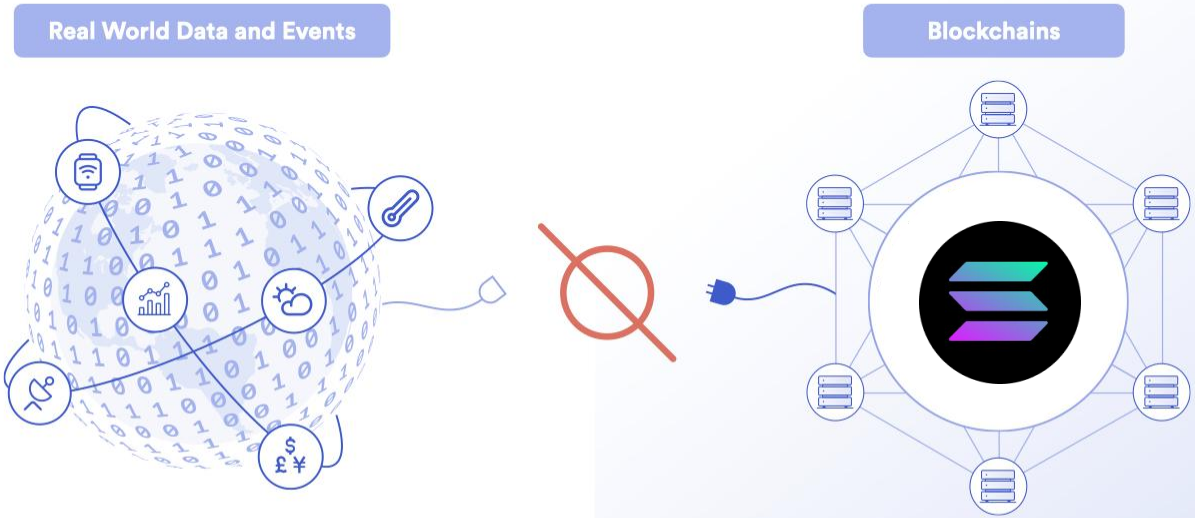
Word Cloud

https://www.menti.com/hj9942efsc

The Oracle Problem and Chainlink

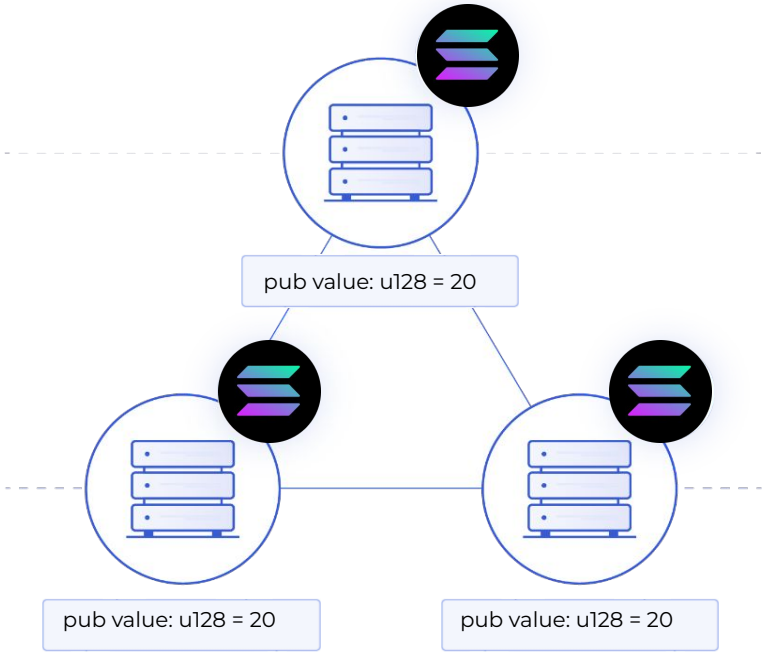
The Oracle Problem

Solana Programs are unable to connect with external systems, data feeds, APIs, existing payment systems or any other off-chain resources on their own.



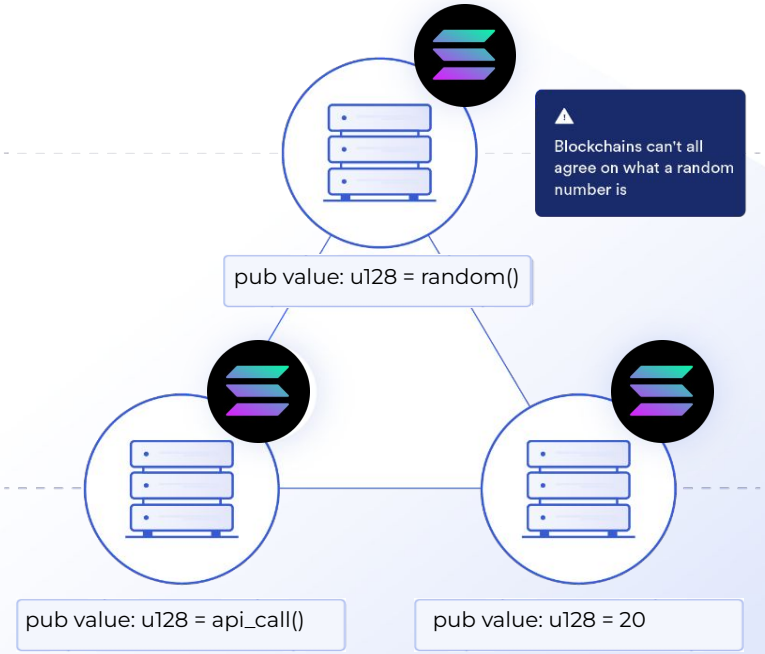
Chainlink

Deterministic



Consensus reached

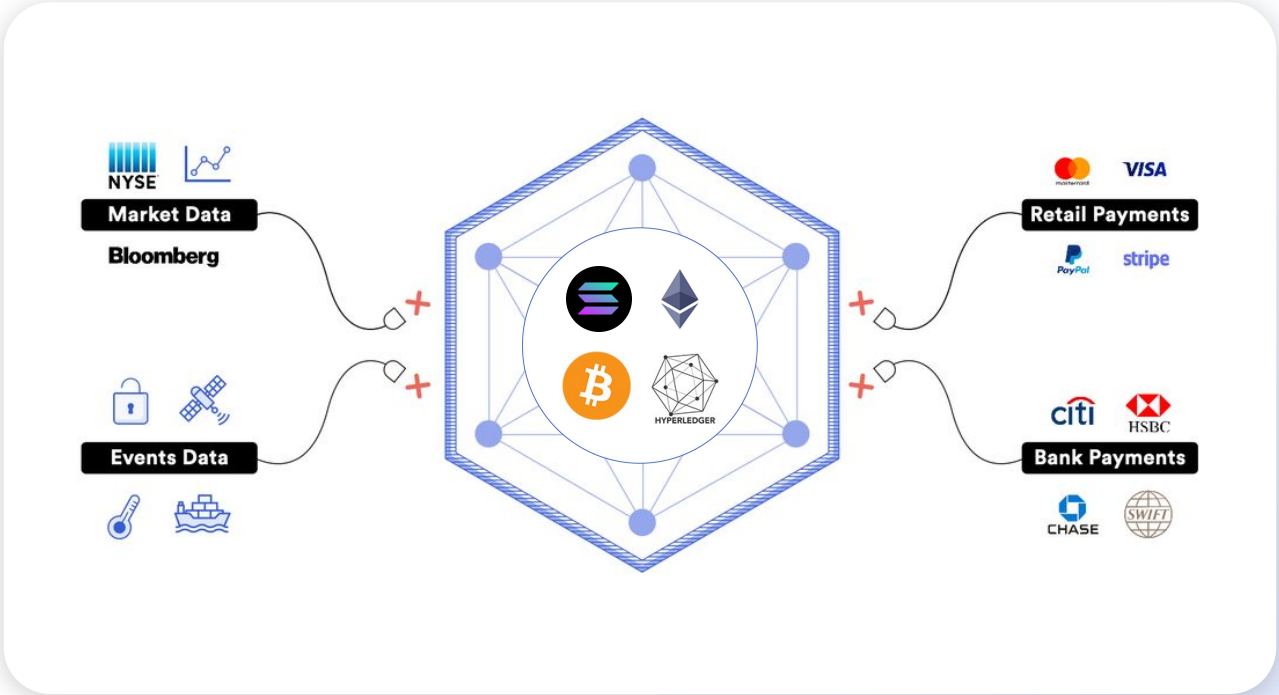
Non-deterministic



Blockchains can't all agree on API calls

Consensus not reached

The Smart Contract Connectivity Problem



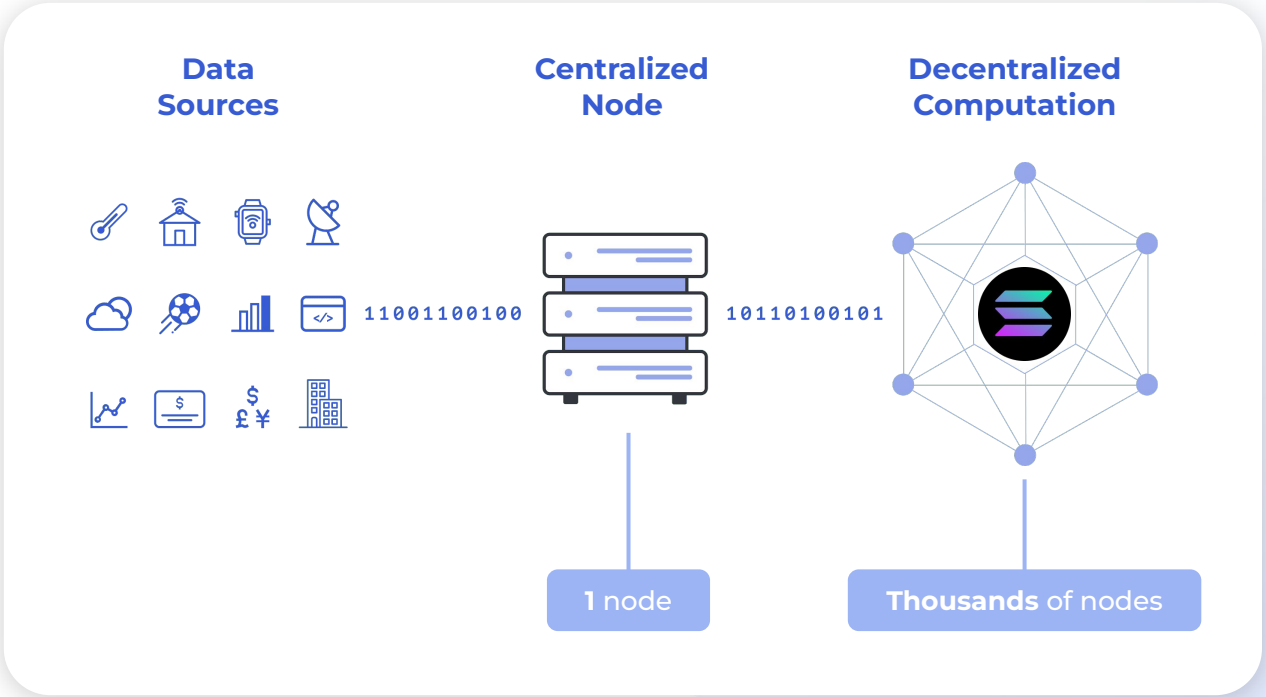
Chainlink

The Smart Contract Connectivity Problem

Blockchain Oracle:

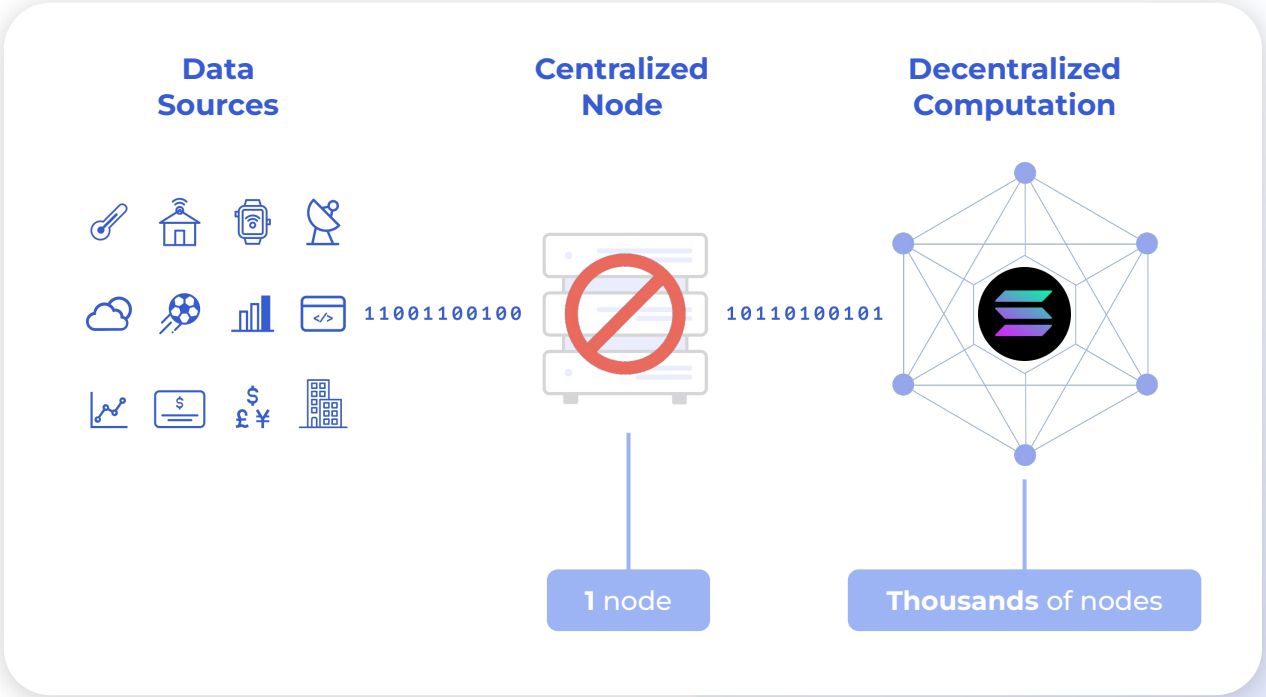
Any device that interacts with the off-chain world to provide external data or computation to smart contracts.

Centralized Oracles are a Point of Failure



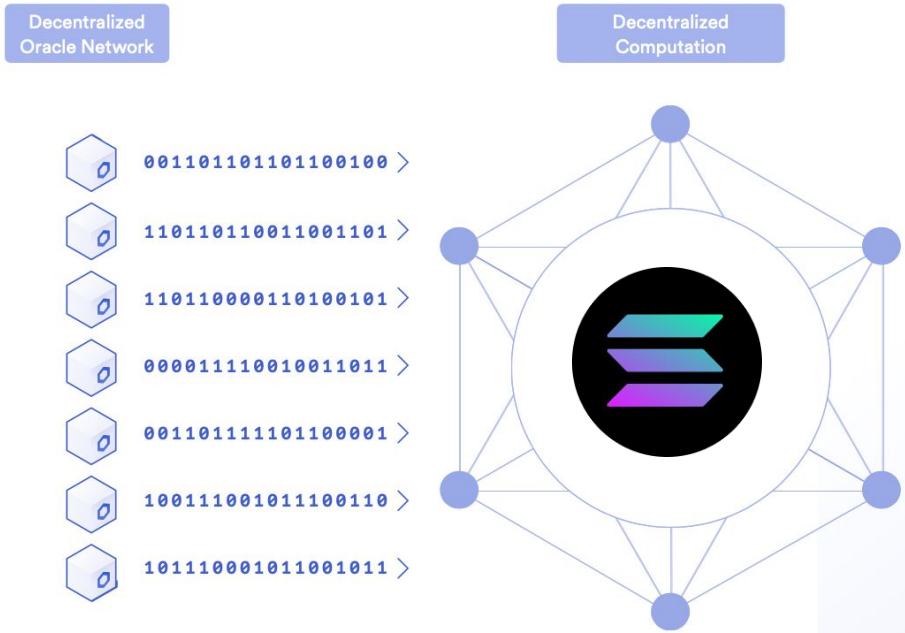
Chainlink

Centralized Oracles are a Point of Failure



Chainlink

A Decentralized Oracle Network

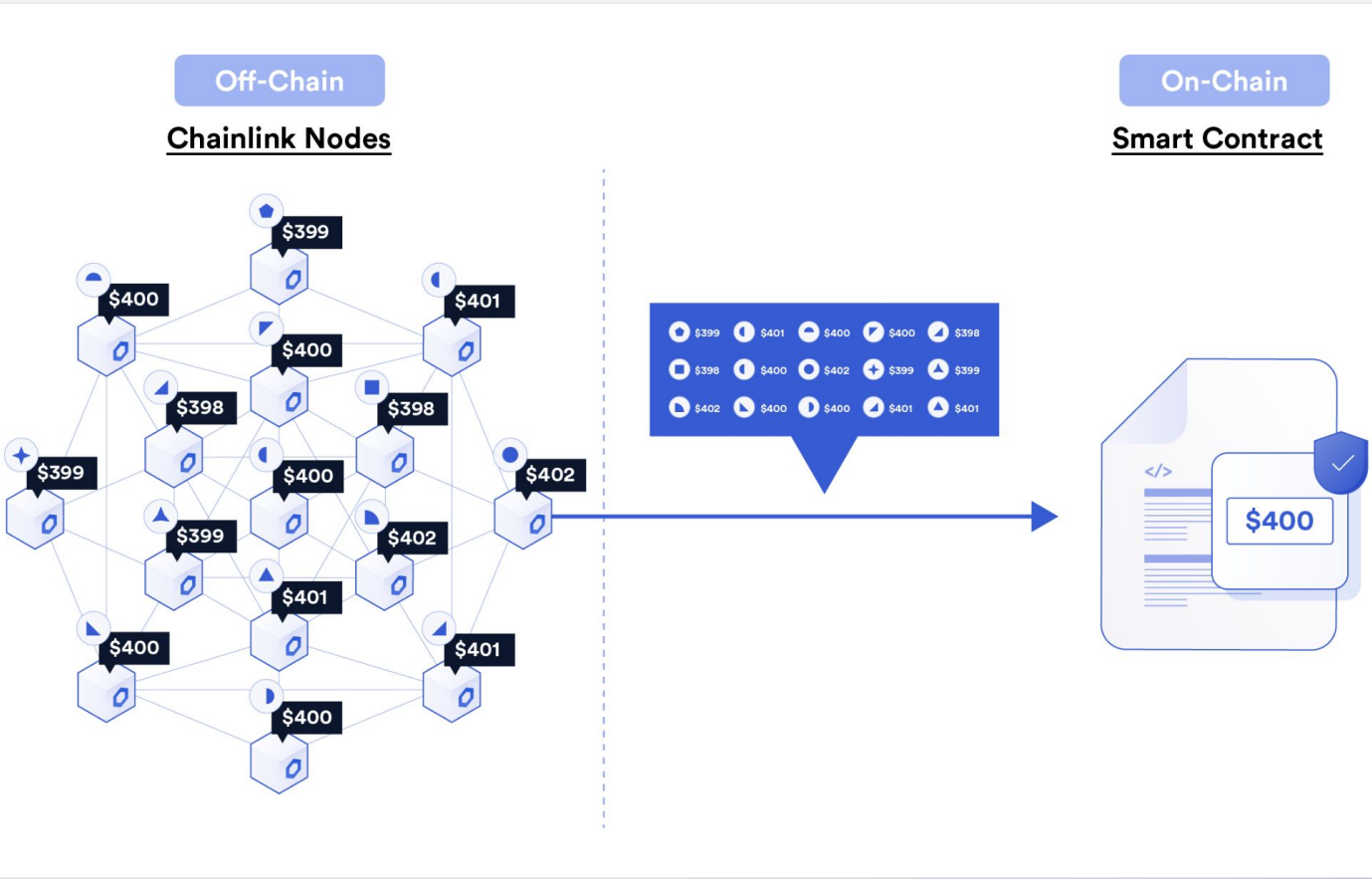


Decentralization

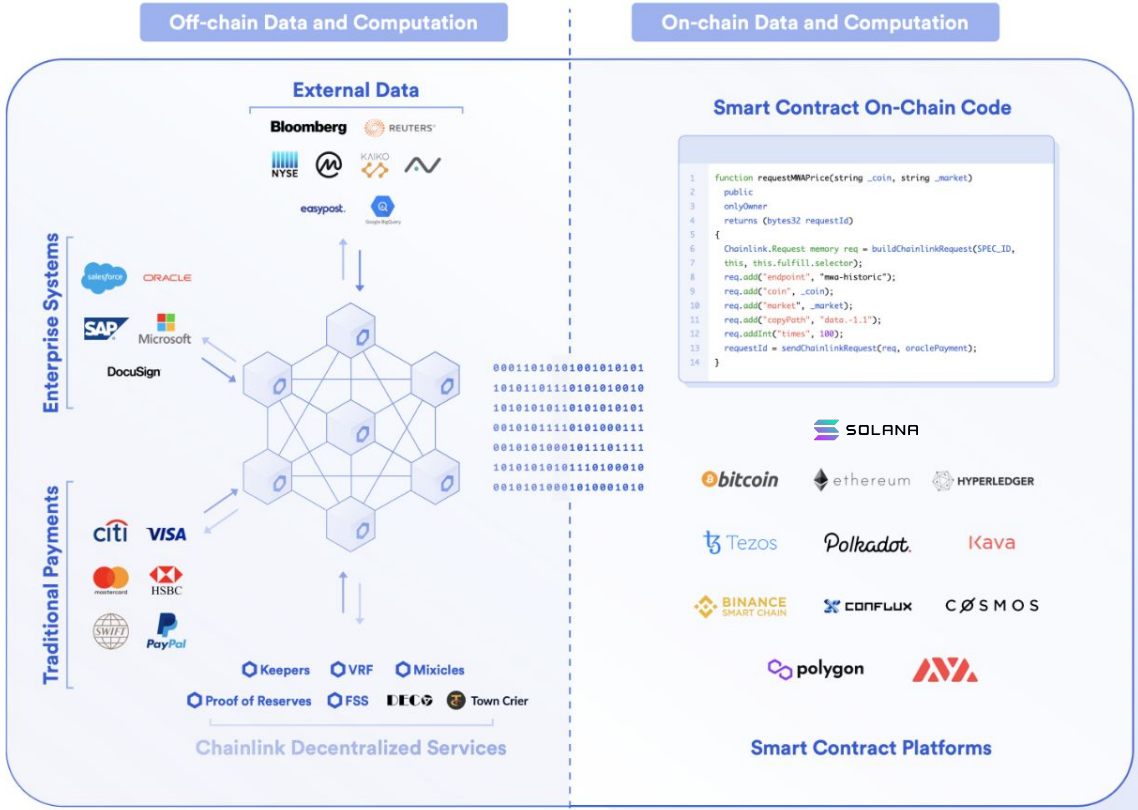
Full replicas being run by independent and sybil resistant node operators, coming to consensus about a computation.

Focused on data validation and consensus about individual off-chain values to make them reliable enough to trigger contracts.

Node Operators are security reviewed, can provide a proven performance history and are high quality and highly sybil resistant.


Off-Chain
Chainlink Nodes
On-Chain
Smart Contract
$399
$400
$401
$400
$398
$398
$399
$400
$402
$399
$402
$401
$400
$401
$400
$399 $401 $400 $400 $398
$398 $400 $402 $399 $399
$402 $400 $400 $401 $401
</>
$400

Hybrid Smart Contracts Combine On-chain & Off-chain Systems



Chainlink

DeFi Provides a Unique “Cryptographic Truth” Guarantee

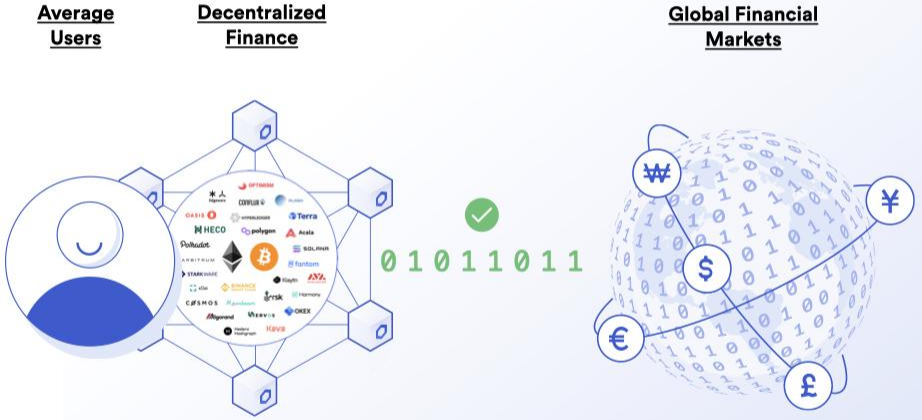
Nasdaq

DeFi Trend: The Solution – and Huge Opportunity – to the Robinhood Fiasco

CONTRIBUTOR
Matt McCall — InvestorPlace

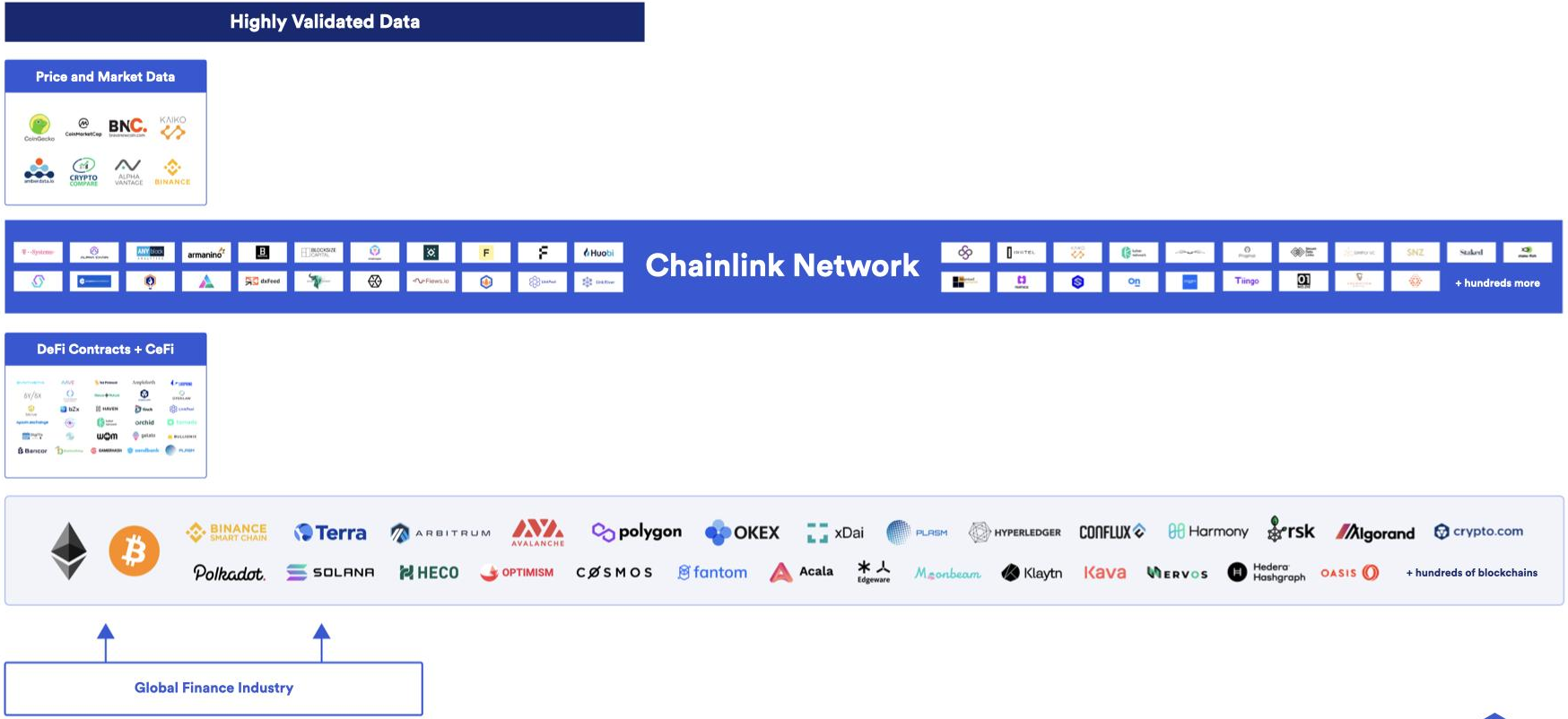
PUBLISHED
FEB 6, 2021 5:30PM EST

Source: Shutterstock



Chainlink

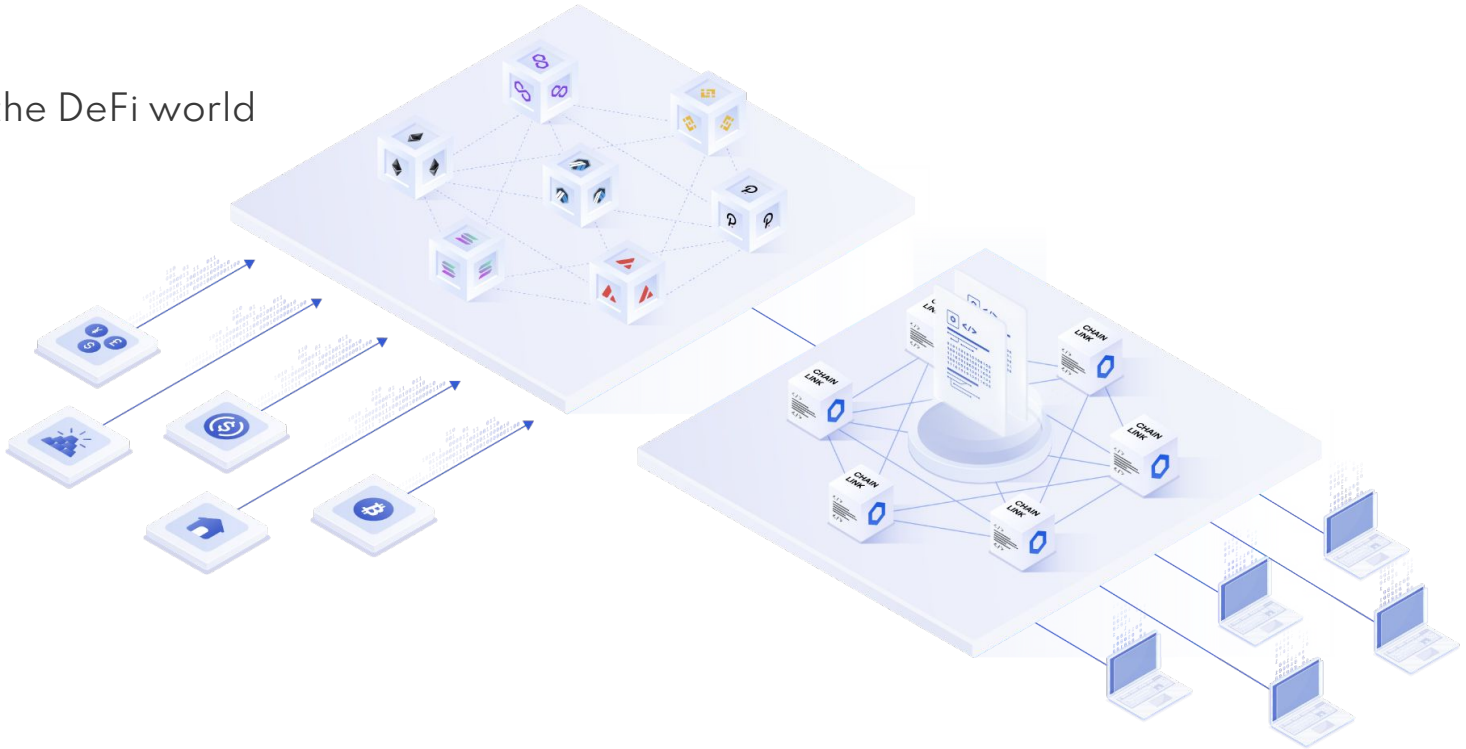
Decentralized Price Data Enabled DeFi to Rapidly Grow



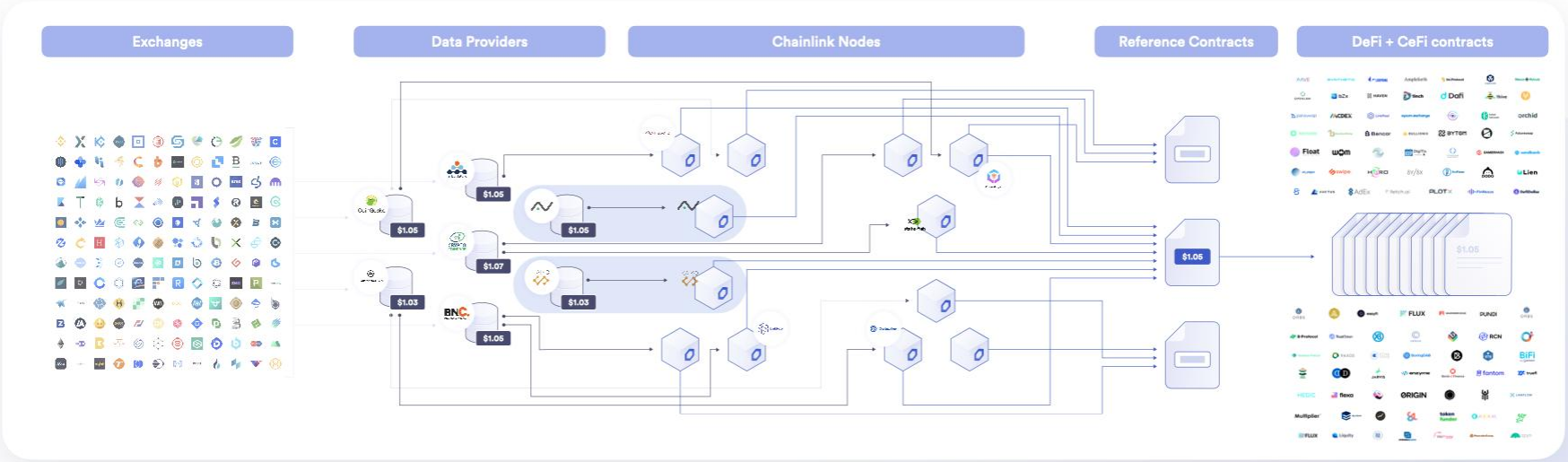
Chainlink

Chainlink Data Feeds

Powering the DeFi world



Chainlink's Decentralized Data Feeds Secure Most of DeFi



"With partnerships across DeFi and traditional markets, the ground the Chainlink team has covered in a short time is nothing short of breathtaking."

Forbes

"Chainlink has taken a commanding lead among DeFi oracles, a perch that's been strengthened with frequent announcements of new partnerships and node operators."

coindesk

"Chainlink's price oracles are quickly emerging as the industry-standard data source for the DeFi sector"

COINTELEGRAPH

Chainlink

Solana Devnet Chainlink Price Feeds

Solana Feeds

☆☆☆☆☆
Suggest Edits

To learn how to implement these feeds, see the Solana Examples for Consuming Data Feeds.

To learn how to obtain SOL tokens on the Solana Devnet, see the Solana Documentation.

Note, off-chain equity and ETF assets are only traded during standard market hours (9:30 am - 4 pm ET M-F). Using these feeds outside of those windows is not recommended.

Solana Devnet

☐ Show Heartbeat and Deviation Details

Pair	Dec	Account
BTC / USD	8	F4QZEMQjNRvwHd9tLgp74fVD99Bg3cXbHuEQ7EGymBq6
ETH / USD	8	5zxs8888az8dgB5KauGEFoPuMANtrKtkpFiFRmo3cSa9
LINK / USD	8	CFRkaCg9PcuMaCZZdcePkaa8d8ugtH221HL7tXQHNVia
SOL / USD	8	EdWr4ww1Dq82vPe8GFjjcVPo2Qno3Nhn6baCgM3dCy28
USDC / USD	8	H7z12eWzwmjgBwy7bxHzSY5EmFdviriVE1Y9RxFQvK1L
USDT / USD	8	A7Ev4pEqutTxHofM9AvWFKgVAkA27u3LU2Xf1KzuN9RS

Please be careful with the feeds used by your smart contracts. The feeds listed in our official documentation have been reviewed; feeds built for custom deployments by other community members might have additional risks. Please do close diligence on your feeds before implementing them in your contracts. Learn more about making responsible data quality decisions.

Chainlink

Exercise 3: Price Feeds Consumer Contract

- Program that looks up SOL/USD price using Chainlink Data Feeds, and stores the result
- 60 minutes

Where Can You Learn More

Solana Cookbook

Chainlink

Cooking with Solana

The *Solana Cookbook* is a developer resource that provides the essential concepts and references for building applications on Solana. Each concept and reference will focus on specific aspects of Solana development while providing additional details and usage examples.

Contributing

The Cookbook is designed in a way that makes it easy for new Solana developers to contribute. Even if you don't know how to do something, contributing to the cookbook is a great way to learn!

Check out all open issues here. Contribution guidelines here. If you find the cookbook is missing a suggestion, please add an issue.

How to Read the Cookbook

The Solana Cookbook is split into different sections, each aimed at a different goal.

Section	Description
Core Concepts	Building blocks of Solana that are good to know for development
Guides	Snack-sized guides about different tools for development
References	References to commonly needed code snippets

solanacookbook.com

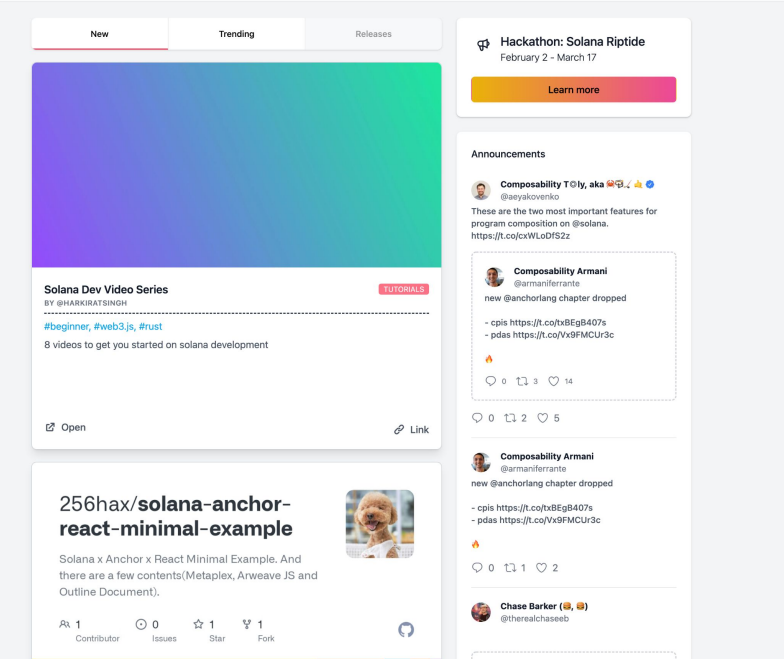
Contributors: Frank V. Castellucci

Last Updated: 13/02/2022, 00:20:49

Chainlink

SolDev

Chainlink

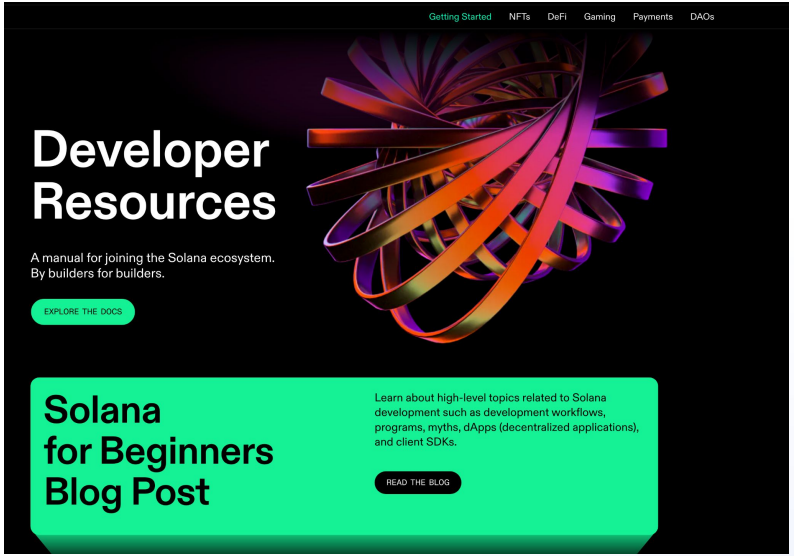


https://soldev.app/

Chainlink

Solana Developers

Chainlink

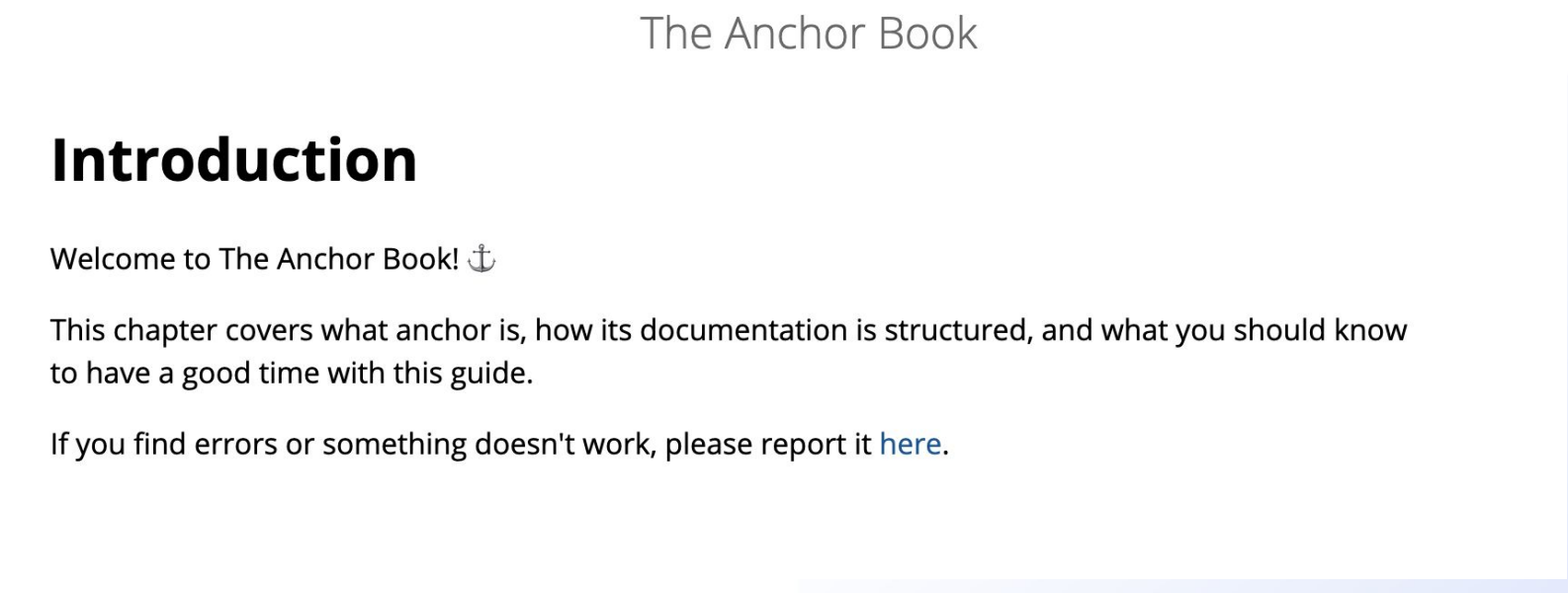


https://solana.com/developers

Chainlink

The Anchor Book

Chainlink



https://book.anchor-lang.com/

Chainlink

Solana Starter Kit

Chainlink Solana Starter Kit

The Chainlink Solana Starter Kit is an Anchor based program and client that shows developers how to use and interact with Chainlink Price Feeds on Solana. The demo is configured to run on the Devnet cluster, and is comprised of an on-chain program written in Rust, and an off-chain client written in JavaScript. The program takes paramters and account information from the off-chain client, retrieves the latest price data from the specified Chainlink Price Feed on Devnet, then writes the data out to the specified account, which can then be read by the off-chain client.

Running the example on Devnet

Requirements

- NodeJS 12 or higher
- Rust
- Solana CLI
- A C compiler such as the one included in GCC.

Building and Deploying the Consumer Program

First, ensure that you're in the `solana-starter-kit` directory in this repository

```
cd ./solana-starter-kit
```

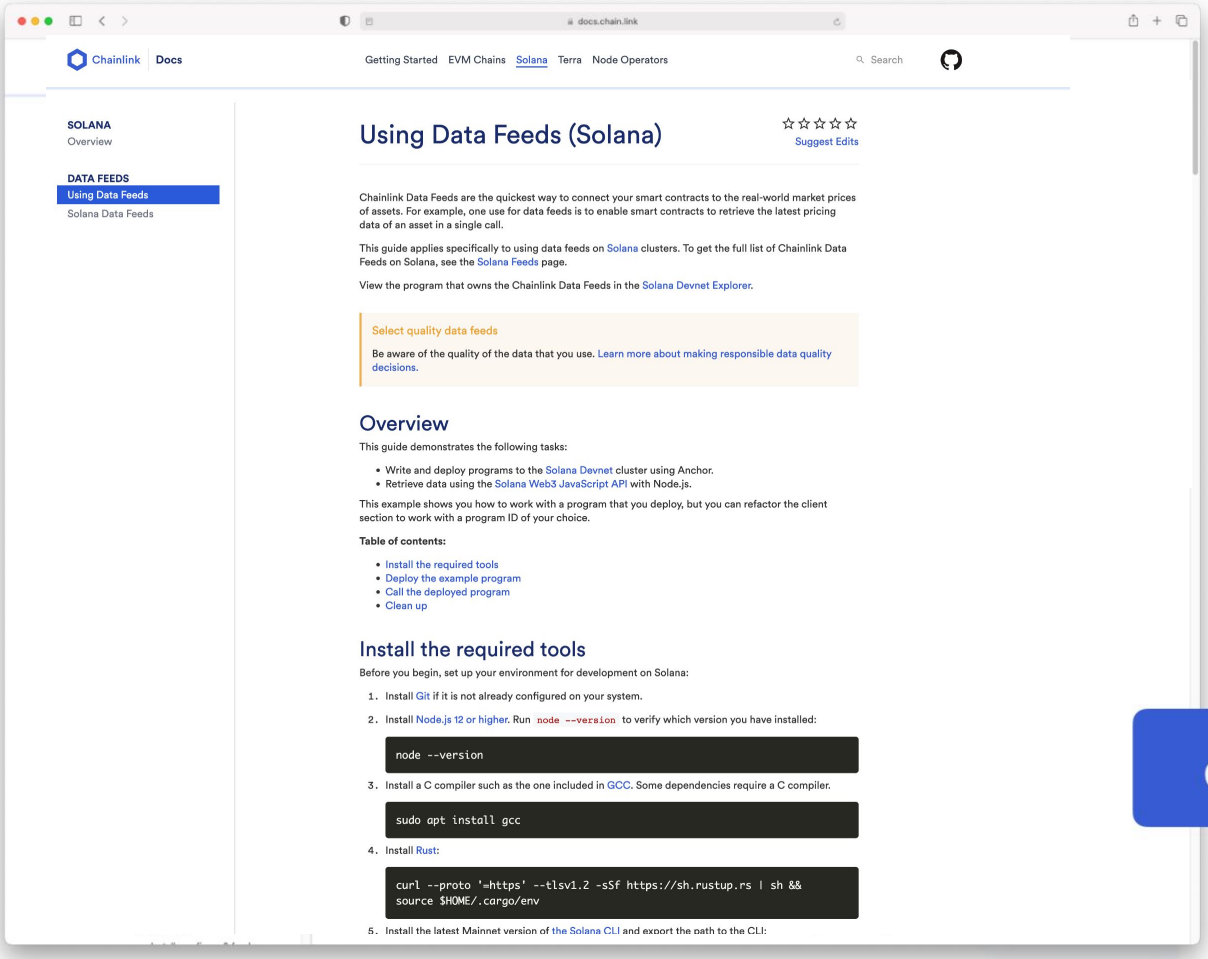
Next step is to install all of the required dependencies:

```
npm install
```

Note for Apple M1 chipsets: You will need to perform an extra step to get the Anchor framework installed manully from source, as the NPM package only support x86_64 chipsets currently, please run the following command to install it manually:

```
cargo install --git https://github.com/project-serum/anchor --tag v0.20.1 anchor-cli --locked
```

Chainlink



LINK Token Contracts

Ethereum

Mainnet

Kovan

Rinkeby

Goerli

docs.chain.link

blog.chain.link

blog.chain.link

Chainlink VISION ANNOUNCEMENTS EVENTS DEVELOPERS EDUCATION GRANTS CHINESE KOREAN

Developers

DEVELOPERS

Fetch IPFS Data in Smart Contracts Using a Chainlink External Adapter

Learn how to use a Chainlink External Adapter to fetch IPFS data and reduce Ethereum gas fees for large token distributions and airdrops on Audius and other platforms.

DEVELOPERS

Off-Chain Computation: Statistical Analysis With Chainlink

Learn how to use Chainlink's External Adapters to enable off-chain computation of statistical weather analysis.

DEVELOPERS

Chainlink Developers Starter Kit Showcase

Announcing the Starter Kit Showcase, a month-long initiative that encourages developers to use our Starter Kits to create smart contracts that make use of Chainlink tools.

DEVELOPERS

How to Connect a Two-Factor Authentication (2FA) API to a Smart Contract With Chainlink

Chainlink hackathon winner Digital Bridge walks through how they built a two-factor authentication service to trigger on-chain events using Chainlink oracles.

DEVELOPERS

Build Your Own Dynamic NFT With Hardhat

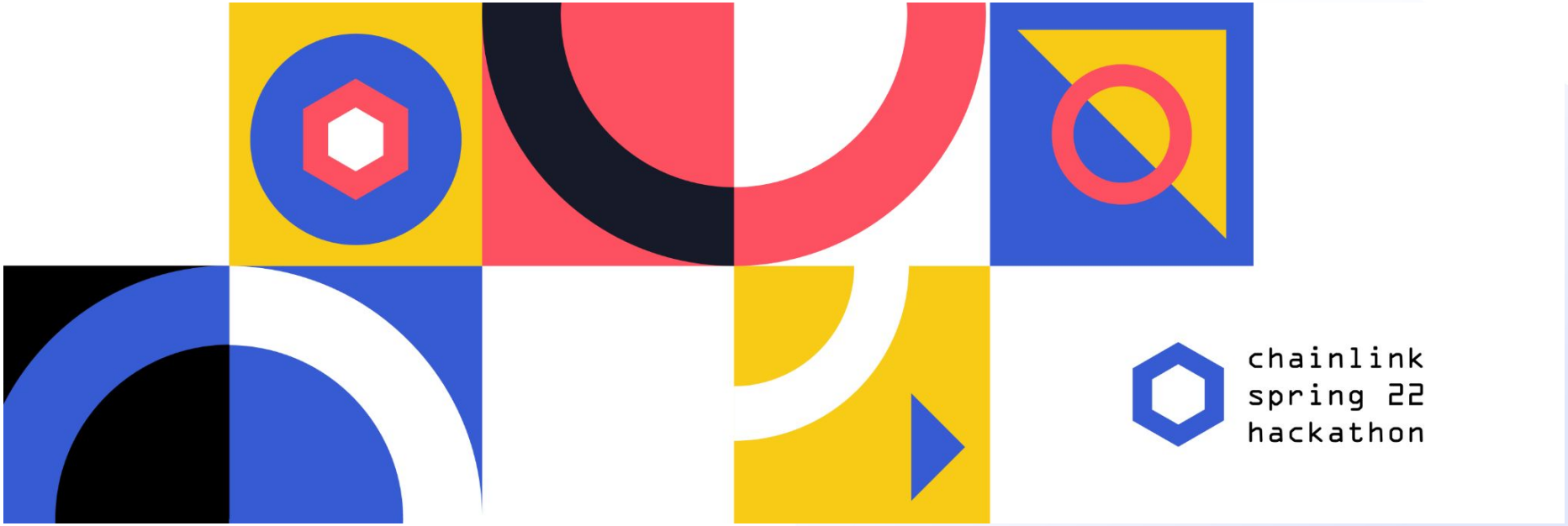
In this tutorial, learn how to use Chainlink VRF in Hardhat's popular development environment to build a next-generation NFT that changes every time it's transferred.

DEVELOPERS

Craft Whiskey Crypto Payments With Chainlink Oracles

Learn how Chainlink bounty winners implemented a Chainlink Price Feed to provide whiskey buyers and sellers accurate and tamper-proof on-chain prices of craft whiskey bottles.

http://hack.chain.link



Fundamentally, contracts define the terms and obligations for an exchange of value between two or more independent parties. Historically, a centralized arbitrator is usually required to verify if those terms and conditions are met. However, thanks to the advent of blockchain technology and smart contract applications, we can now replace centralized arbitrators with decentralized infrastructure, reducing counterparty risk and improving operational efficiency.

- **A Verifiable Random Function (VRF)** to access a provably fair and secure Random Number Generator (RNG) directly on-chain.
- **Modular External Adapters** to connect to any off-chain resource like premium data providers, web APIs, IoT sensors, bank payments, enterprise backends, other blockchains, and more.
- **Various other oracle services** such as Fair Sequencing Services for fair transaction ordering, DECO for privacy-preserving attestations of TLS web session data, Arbitrum Rollups for scalable off-chain Solidity computation, and more.

Decentralized Finance

Money is the common medium used today to value and exchange assets. Financial products provide different vehicles in which people can maximize the value of their money via different strategies like hedging, speculating, earning interest, collateralizing loans, and more. However, traditional finance is often gated, wherein well-capitalized entities have disproportionate control over the issuance of money and ial products. The result is a lack of ducts and the introduction of more influence on whether the financial agreed upon terms.

inistic execution to financial products, tion, and provide tamper-proof oracles play a critical role in creating ial products and monetary instruments, et data like FX rates, interest rates, asset

d 1:1 to fiat currency, commonly the US old a non-volatile cryptocurrency. While off-chain bank account, decentralized on-chain cryptocurrencies and require g, a user's collateral is worth over 150%

eta-stablecoin (a stablecoin backed by Feeds to track the price of the , and USDC. In the event that one or USD peg, thus causing DUSD to also he four reserves in order to preserve the

DeFiDollar uses Chainlink Price Feeds for valuation data on the stablecoins backing DUSD

Money Markets

Blockchain-based money markets are crucial financial infrastructure that use smart contracts to connect lenders, who wish to earn yield on their assets, with borrowers, who wish to gain access to working capital. They allow users to increase the utility of their crypto holdings and participate in both the supply and demand side. However, to ensure the solvency of the platform, price feeds are required to track the valuation of assets used on the platform as a means of ensuring loans are issued at fair market prices and liquidations automatically occur on undercollateralized loans.

Aave is an example of an on-chain money market protocol that uses Chainlink Price

Tokensets is one such example of a protocol that uses Chainlink Price Feeds to generate various "Sets," tokenized positions that execute trades on the behalf These Sets are based on Technical Analysis (TA) metrics such as the RSI or mo averages, designed to catch key price action trends. Additionally, users can use Set tokens as collateral within other protocols, such as the Aave money market additional capital efficiency.

Real-World Assets

As discussed in our recent education piece, tokenized real-world assets are am most promising use cases for blockchain and smart contract technology. They real-world assets and represent them on the blockchain as a token. Compared traditional assets, tokenized assets benefit from global accessibility, permissio liquidity, on-chain transparency, and reduced transactional friction.

Proof of On-Chain Reserve

Wrapped cross-chain assets—cryptocurrencies/tokens native to one blockchai are locked into a contract and then "unlocked" on another blockchain—are bec increasingly popular due to their ability to increase the collateral types availab within the DeFi ecosystem. However, in order to ensure the integrity of DeFi applications supporting wrapped asset deposits, Proof of Reserve reference co can be used to supply data regarding the true collateralization of these on-chai

Two protocols using Chainlink to power Proof of Reserve reference feeds inclu BitGo's WBTC and Ren Protocol's renBTC, representing over 90% of the wrap Bitcoin on Ethereum and representing billions in USD value. These Proof of R reference feeds provide DeFi protocols with the data they need to autonomous collateral reserves and swiftly protect user funds during an undercollateraliza event. Proof of Reserve reference feeds can also be used to track the collateral assets beyond cross-chain tokens includes stablecoins and real-world commod further increasing the collateral available within DeFi.

Chainlink Proof of Reserve provides smart contracts proof of the Bitcoin collateral backing BitGo's Wrap

Proof of Off-Chain Reserve

Bringing real-world assets onto the blockchain provides a large potential to ex economic activity of DeFi, as seen with the adoption of fiat-backed stablecoins. However, this requires the underlying collateral to be held by a central custodi disconnecting the on-chain tokenized representation from the actual off-chain itself. Through Chainlink Proof of Reserve, smart contacts are able to autonom audit the collateralization of real-world asset backed tokens, protecting users black swan events.

An example of this is the TUSD Proof of Reserve reference feed that provides applications with data regarding the true amount of US dollars backing the sta TUSD held by TrustToken's off-chain escrowed bank accounts as reviewed by Armanino, an independent top 25 auditing firm in the United States. This collateralization data can be checked against the total amount of circulating TU tokens on various blockchains, as reported by the complementary TUSD Proof Supply feed, to determine the collateralization of TrustToken's tokenized USD

Get Started Now!

Solana Discord Chat
https://discord.gg/solana

Discord Chat
https://discord.gg/chainlink

Documentation:
http://docs.chain.link

Hackathon Resources
https://stackoverflow.com/questions/ask?tags=chainlink

Chainlink

Solana Blockchain Developer Bootcamp Day 2 Summary

- Intro Anchor
- Writing and interacting with programs using Anchor
- The oracle problem and Chainlink
- Chainlink Data Feeds

Chainlink


Chainlink
SOLANA
Congratulations for completing the Solana Blockchain Developer Bootcamp
Milestone
Milestone
Milestone