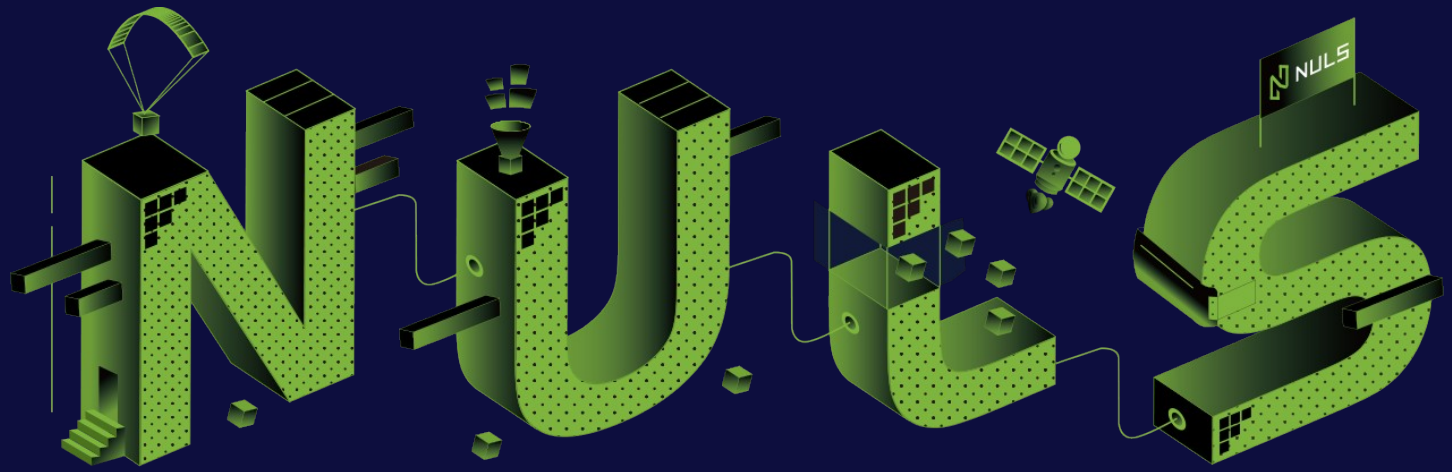
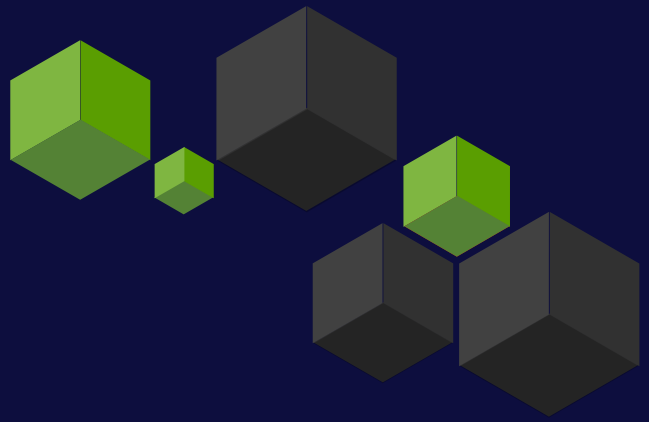
NULS

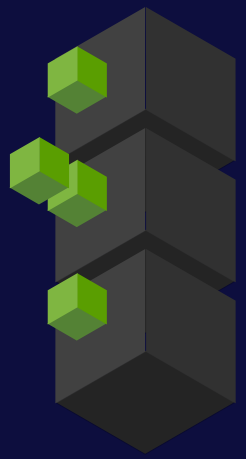


NULS

Blockchain ?



- Blocks
- Transactions
- UTXO
- Open-Source



Smart Contracts

NULS

Smart Contracts

- What is a Smart Contract?
  - Distributed
  - Public

- In which environment do they live?
  - Agent Nodes
  - NVM (Nuls Virtual Machine)
  - Deterministic (validations)
  - Mainnet / Testnet

NULS

Smart Contracts

- How can I build one ?
  - Java (JDK 8) (with some restrinctions...)
  - NULS - SC SDK
  - IntelliJ IDEA - NULS Plugin

- How can I interact with it ?
  - NULS wallet (deployment, calls)
  - nuls-js / Dapps
  - Gas!


NULS
Contract development
- Introduction to NULS Smart Contract SDK
@NalexNuls

Smart Contract - SDK

```java
package contracts.examples;

import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;

public class SimpleStorage implements Contract {

    private String storedData;

    @View
    public String getStoredData() {
        return storedData;
    }

    @Payable
    public void setStoredData(@Required String storedData) {
        this.storedData = storedData;
    }

}
```

package io.nuls.contract.sdk

- Contract interface
- Block
- BlockHeader
- Address
- Msg (call context)
- Utils
- Annotations
- Events

Smart Contract - SDK

```java
package contracts.examples;

import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;

public class SimpleStorage implements Contract {

    private String storedData;

    @View
    public String getStoredData() {
        return storedData;
    }

    @Payable
    public void setStoredData(@Required String storedData) {
        this.storedData = storedData;
    }

}
```

Contract interface

- The main class of our contract must implement the Contract interface
- It only can be one class that implements this interface

Smart Contract - SDK

Contract interface

- *_payable* method can be optionally implemented by our contract.
- It will be called each time that our contract receive a transfer

```
/**
 *  Contract interface, implemente
 */
public interface Contract {

    /**
     * Directly transfer to the co
     */
    default void _payable() {
    }

}
```

Smart Contract - SDK

```java
package contracts.examples;

import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;

public class SimpleStorage implements Contract {

    private String storedData;

    @View
    public String getStoredData() {
        return storedData;
    }

    @Payable
    public void setStoredData(@Required String storedData) {
        this.storedData = storedData;
    }

}
```

Contract state

- We store data, just by modifying class member properties
- Contract state is composed by all property values
- The last contract state will be the initial state in a new call
- Supported data types:
  - Primitive types: byte, bool, int, short, long, float, double…
  - Basic types: String, Integer, BigInteger, Foat, Double, BigDecimal…
  - Basic Data Structures: ArrayList, HashMap, HashSet...

Smart Contract - SDK

```java
package contracts.examples;

import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;

public class SimpleStorage implements Contract {

    private String storedData;

    @View
    public String getStoredData() {
        return storedData;
    }

    @Payable
    public void setStoredData(@Required String storedData) {
        this.storedData = storedData;
    }

}
```

View annotation

- Makes method callable
- Can not modify the contract state
- Do not spend “gas”
- Must return some value
- Returned values are serialized using *toString* method
- Called through API (not reflected as transaction)

Smart Contract - SDK

```java
package contracts.examples;

import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;

public class SimpleStorage implements Contract {

    private String storedData;

    @View
    public String getStoredData() {
        return storedData;
    }

    @Payable
    public void setStoredData(@Required String storedData) {
        this.storedData = storedData;
    }

}
```

Payable annotation

- Makes method callable
- Can modify the contract state (not mandatory)
- Spent “gas” for each line of code executed
- Not necessarily returns some value
- An special blockchain transaction is made under the hood

Smart Contract - SDK

```java
package contracts.examples;

import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;

public class SimpleStorage implements Contract {

    private String storedData;

    @View
    public String getStoredData() {
        return storedData;
    }

    @Payable
    public void setStoredData(@Required String storedData) {
        this.storedData = storedData;
    }

}
```

Method arguments

- Our contract methods can accept some arguments
- All types that we saw before are allowed to be passed as argument
- Can be passed serialized as String

Required annotation

- Declare some method argument as mandatory

Smart Contract - SDK

```
public class Faucet extends Owner implements Contract {

    private BigInteger amount;
    private Set<Address> blackList = new HashSet<>();

    public Faucet(@Required BigInteger amount) {

        this.amount = amount;

    }
```

Contract constructor

- Optional implementation
- Will be called during contract deployment
- We can provide some arguments to the constructor that will be requested on deployment

Smart Contract - SDK

```
@Payable
public void getFunds() {

    Msg.sender().transfer(Msg.address().balance());

}
```

```
@Payable
public void registerIntoSpace(@Required Address spaceAddress) {

    String[][] args = new String[][] {{Msg.address().toString()}};
    spaceAddress.call("registerProvider", "", args, BigInteger.ZERO);

    advertisementSpaces.add(spaceAddress);

}
```

Address class

- Represents an account of the network
- It can be an smart contract address too
- Allow us to handle account operations:
  - *transfer* - tokens to the account
  - *call* - external contracts methods
  - *balance* - checks account balance

Smart Contract - SDK

```
class Owner {

    protected Address owner;

    Owner() {
        owner = Msg.sender();
    }

    protected boolean isOwner() {
        return Msg.sender().equals(owner);
    }

    protected void requireOwner() {
        require(isOwner(), "Sender is not owner");
    }

}
```

```
@Payable
public void topUp() {

    require(Msg.value().compareTo(BigInteger.ZERO) > 0);
    require(Msg.address().balance().compareTo(BigInteger.ZERO) == 0);

}
```

Msg class

- Contains information about the call context:
  - *Sender* - The address of the account or contract that is calling the method
  - *Address* - The address of the current contract being called
  - *Value* - The amount of NULS sent to the contract with the call
  - *GasLeft* - The amount of gas spent till this line
  - *GasPrice* - The gas price set for this call

Smart Contract - SDK

```
class Owner {

    protected Address owner;

    Owner() {
        owner = Msg.sender();
    }

    protected boolean isOwner() {
        return Msg.sender().equals(owner);
    }

    protected void requireOwner() {
        require(isOwner(), "Sender is not owner");
    }

}
```

```
long numTickets = ticketMap.size();

BigInteger seed = getRandomSeed(Block.newestBlockHeader().getHeight(), 20);

long winnerIndex = (long) (pseudoRandom(seed.longValue()) * numTickets) + 1;
```

Utils class

- Static helper class for different purposes:
  - *require / revert* - To throw an exception and abort the call under some conditions
  - *emit* - To trigger an event
  - *sha3* - Used for hashing
  - *verifySignatureData* – To verify ECDSA signatures
  - getRandomSeed / pseudoRandom - To produce random numbers using NULS random number solution

Smart Contract - SDK

```
private void updateStatus(Lottery lottery) {
    if (lottery.getStartTime() <= Block.timestamp()) {
        lottery.setStatus(LotteryStatus.OPEN);
    }
    if (lottery.getEndTime() <= Block.timestamp()) {
        lottery.setStatus(LotteryStatus.CLOSED);
    }
}
```

```
long numTickets = ticketMap.size();
BigInteger seed = getRandomSeed(Block.newestBlockHeader().getHeight(), 20);
long winnerIndex = (long) (pseudoRandom(seed.longValue()) * numTickets) + 1;
```

Block & BlockHeader classes

- Each payable transactions are stored in NULS blockchain as a transaction
- This classes give us information about the block where the “call contract” transaction will be included:
  - *timestamp*
  - *hash / number*
  - *coinbase* - The block miner reward address
  - *PackerAddress* - The block miner address
  - *txCount*

Smart Contract - SDK

```java
@Payable
public void registerIntoSpace(@Required Address spaceAddress) {

    String[][] args = new String[][] {{Msg.address().toString()}};
    spaceAddress.call("registerProvider", "", args, BigInteger.ZERO);

    advertisementSpaces.add(spaceAddress);

}
```

```java
for (Address provider : advertisementProviders) {

    String returnedValue = provider.callWithReturnValue("viewAds", "", null, BigInteger.ZERO);

    try {

        List<Advertisement> adsFromProvider = Advertisement.listFromString(returnedValue, provider);
        ads.addAll(adsFromProvider);

    } catch (Exception ignored) { }

}
```

Calling external contracts

- We will use ***call*** or ***callWithReturnValue*** methods of *Address* class to achieve this.
- Returned values are casted to **String**, so we will need a parsing helper
- We must provide as arguments:
  - Method name to be called
  - Description of the method (optional)
  - A *String[][]* of serialized args to be passed in the call
  - A *BigInteger* with an amount of NULS to be transferred with the call

Smart Contract - SDK

```
public class LotteryWinnerEvent implements Event {

    private Long lotteryId;
    private Long ticketId;
    private Ticket ticket;

    public LotteryWinnerEvent(Long lotteryId, Long ticketId, Ticket ticket) {
        this.lotteryId = lotteryId;
        this.ticketId = ticketId;
        this.ticket = ticket;
    }
```

```
private void setWinnerTicket(Lottery lottery, Ticket ticket, int prize, BigInteger amount) {

    ticket.getOwner().transfer(amount);
    this.decreasePot(lottery, amount);
    ticket.setPrize(prize);
    ticket.setPrizeAmount(amount);

    emit(new LotteryWinnerEvent(lottery.getId(), ticket.getId(), ticket));

}
```

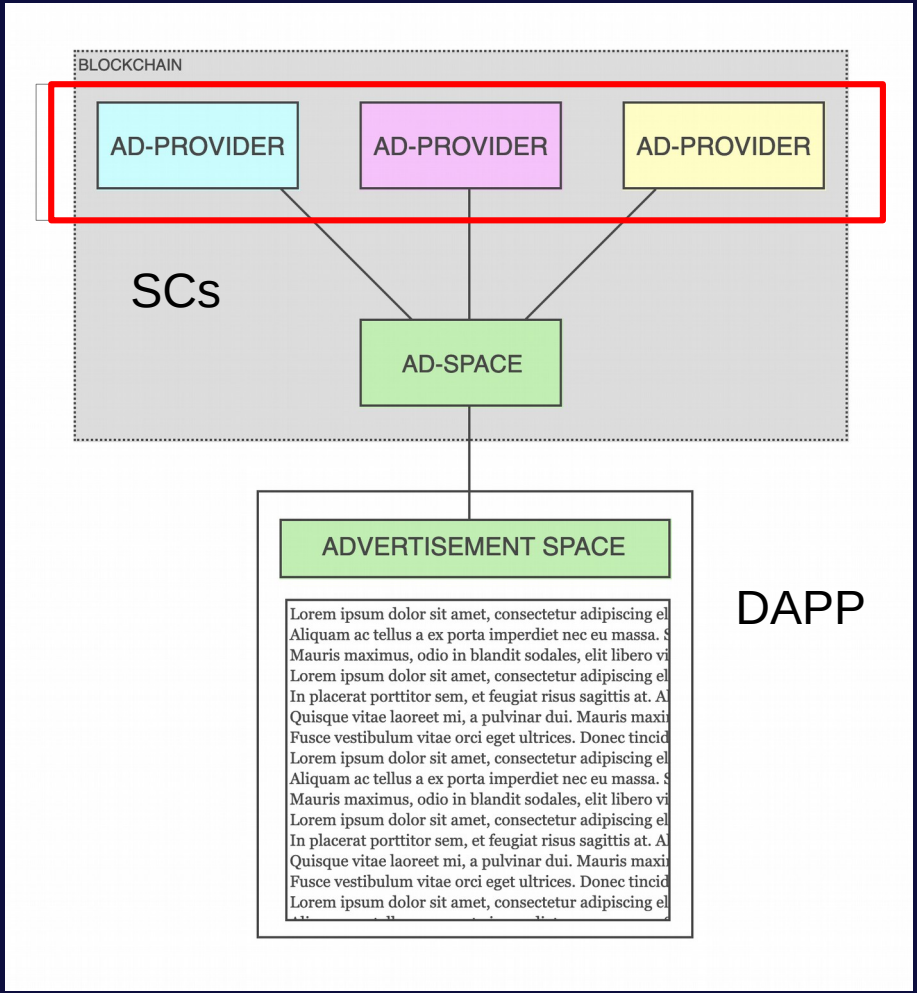
Contract Events

- Should implement Event interface
- Used to emit data off-chain in an async way
- Triggered by calling Utils.emit()
- Common use case: Oracles

Time to build our contract

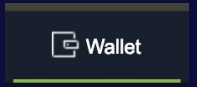
- Advertising provider example

Smart Contract - SDK

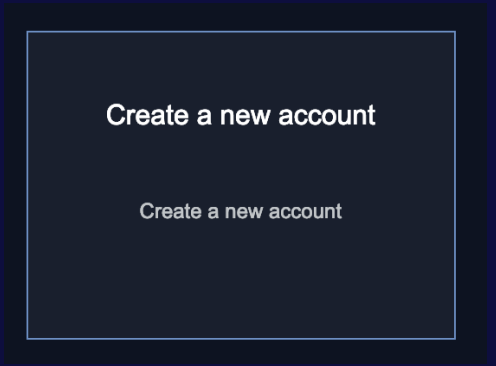


Smart Advertising

Before start

- Download advertisement example repository from github:
  - git clone https://github.com/amalcaraz/nuls-advertising-dapp-example.git
- Open project "*advertising-provider-smartcontract*" on IntelliJ
- Install IDEA NULS plugin:
  - Preferences > Plugins > Install plugin from disk > NULS_IDEA_plugin.zip
- Create a new account in NULS Wallet
  - Go to http://wallet.nuls.services > 

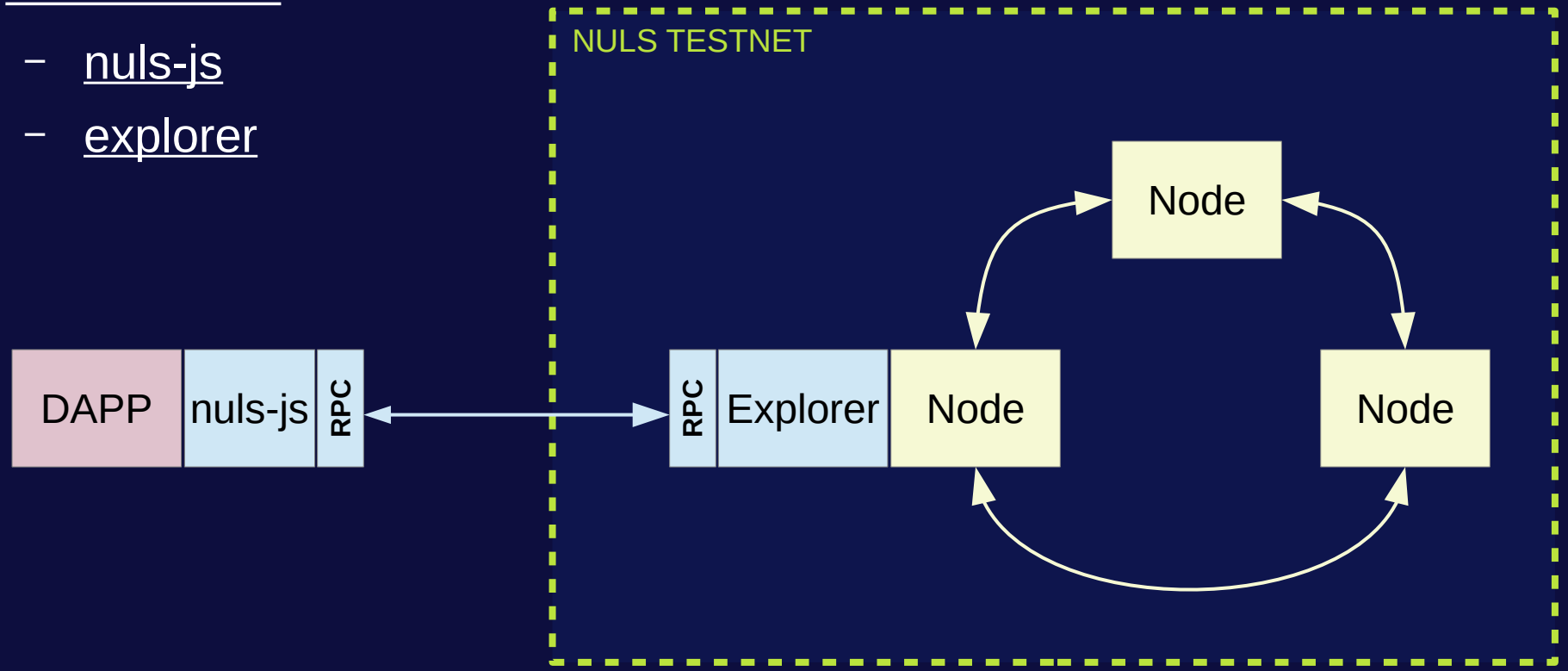
>  >  >
  - Ask some testnet tokens in http://testnet.wallet.nuls.io/#/testNetNULS/testNetNULS (ip error? --> try to do it from your 4G connection)



NULS

What else?

- nuls-js
- explorer




Thank you!