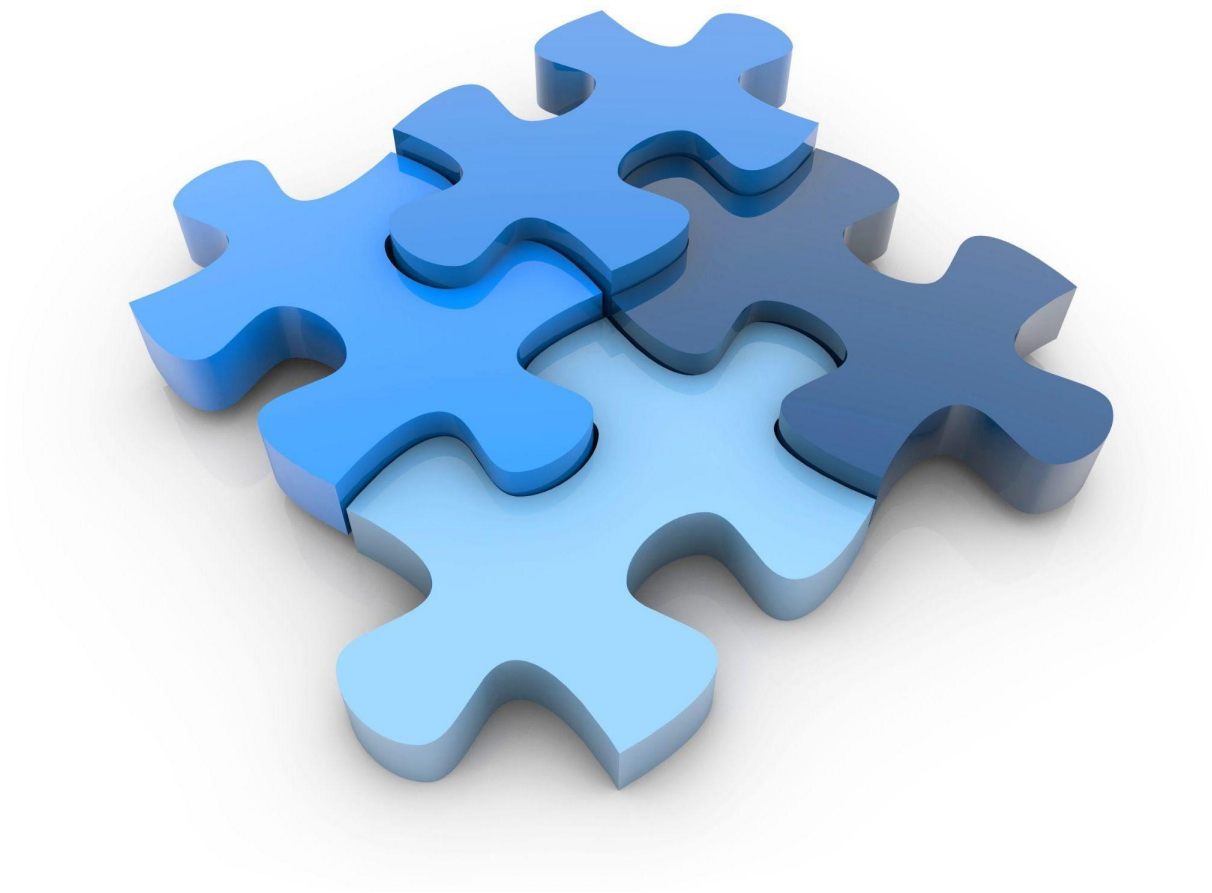


Cardano 스마트 컨트랙트



with Helios

Lawrence Ley, B.Sc

This book would not have been possible without Christian Schmitz and the open source project called Helios.

Preface

2008년 비트코인의 발명으로, 사토시 나카모토의 [백서](#) "비트코인: P2P 전자 현금 시스템"이 신뢰 없는 탈중앙화된 디지털 화폐 시대를 시작했습니다. 2015년에 출시된 [이더리움](#)은 탈중앙화 원장 기술에 프로그래밍 가능성 개념을 도입했습니다. 이로 인해 애플리케이션 개발자들은 블록체인을 이용해 데이터와 프로그램을 탈중앙화되고 허가 없는 방식으로 저장할 수 있게 되었습니다. 2017년에 출시된 **Cardano**는 비트코인의 안전한 UTXO 아키텍처를 활용하고 새로운 결정론적이고 확장 가능하며 안전한 프로그래밍 가능 블록체인을 구축했습니다. **Cardano**는 비트코인과 이더리움을 기반으로 한 3세대 블록체인으로 간주되며 세계 금융 운영 체제가 되고자 합니다.

Helios를 활용한 **Cardano** 스마트 계약이라는 책은 **Cardano**에 대한 새로운 개발자의 온보딩 경험을 가능한 한 쉽고 간단하게 만드는데 초점을 맞추고 있습니다. 다룰 내용이 많으므로 기본부터 시작하여 온체인 및 오프체인 코드를 포함한 보다 정교한 실제 사례에 이르기까지 다룹니다. 이 책은 블록체인 모니터링, 문제 해결 기법 및 생산 준비와 함께 마무리됩니다.

목차

Preface	3
Table of Contents	4
Introduction	7
Demeter Run	8
Setup	9
Query Tip	10
Workspaces	10
Addresses	11
Base Address	12
Enterprise Address	14
Transactions	15
Datums	17
Redeemers	18
Metadata	18
Cardano Blockchain Architecture	19
UTXO Model	19
eUTXO Model	19
Plutus Scripts	20
Signing Transactions	21
Cardano-cli	22
Bash Shell Scripts	23
Test Drive	24
Transfer Ada	26
Wallet	27
Testnet Faucet	28
Expose Port	29
Running Next.js Application	30
Test Drive	31
Under The Hood	34
Minting	36
NFT	37
Minting Smart Contract	38
Compiling The Script	40
Test Drive	45

Submit Tx	46
Cexplorer	48
Multisig NFT	49
Code Changes	50
New Functions	53
Test Drive	54
Validators	59
Vesting	60
Next.js Setup	64
Next.js Startup	65
Smart Contract Viewer	66
Lock Ada	67
Vesting Key Minting Policy	68
Claim Funds or Cancel Vesting	69
Test Drive	72
Donation Traceability (Lock Ada)	75
System Components	76
Smart Contract Code	78
Pay With Ada	80
Locking Ada At The Smart Contract	80
Compile & Deploy	81
Next.js Setup	83
Shopify Settings	84
Test Drive	84
Donation Traceability (Unlock Ada)	89
Unlocking Ada At The Smart Contract	90
Bash Shell Scripts	91
Reference Scripts	93
Spend Script	94
Test Drive	99
Before tx-spend.sh is executed	99
After tx-spend.sh is executed	99
Blockchain Monitoring	101
Polling	102
Cron	103
Valid UTXOs	103
Monitoring	104
Events	105
Oura	105
Event Flow	106

WebHook	107
Smart Contract & Next.js Setup	109
Oura Setup	110
Test Drive	111
Troubleshooting	114
Trace()	115
Show()	116
Print()	117
Common Sources Of Errors	118
Cardano-cli	119
Helios Tx Builder	120
Network Parameters	121
Baseline Comparison	122
CBOR.me	122
Production	125
Legal Notice	126
Audit	126
Test, Test, Test	128
Smart Contract Lifecycle	129
Sources and Acknowledgements	130
Open Source	131
References	132
Appendix	133
Address Key Derivation	134
Installing Next.js	136
Startup Next.js	137
IPFS (NFT Images)	139
Shopify Ada Payments	140
Shopify Configuration	141
About Helios	143
About The Author	144

소개

Cardano의 스마트 계약 개발을 위해서는 학습이 쉽지 않은 **Haskell PlutusTx**를 배워야 했습니다. 낮은 언어, **CPU** 및 메모리 실행 제약 조건, 설정에 소요되는 시간 등으로 인해 새로운 개발자들에게 걸림돌이 발생했습니다.

다행히도, 이러한 걸림돌을 제거하고 개발자가 빠르게 시작할 수 있게 해주는 새로운 대안 **Plutus** 스마트 계약 언어가 등장했습니다. 이를 통해 일반적이지 않은 언어 구문, 시간이 많이 소요되는 **nix-shell/cabal** 구성 및 컴파일로 인한 속도 저하를 피하고, 애플리케이션 및 비즈니스 로직에 더 많은 집중이 가능합니다.

이 책에서는 자바스크립트로 작성된 컴파일러를 가진 강타입의 함수형 프로그래밍 언어인 **Helios**를 다룰 것입니다. **Helios**의 주요 장점 중 하나는 자바스크립트를 지원하는 거의 모든 대상 아키텍처에서 쉽게 컴파일할 수 있다는 것입니다. 또한, **Helios**를 사용하면 **Haskell PlutusTx V2**와 비교하여 약 **50%**의 성능 오버헤드 개선이 가능합니다. **Helios**는 또한 오프체인 코드에서 거래를 구축, 서명 및 제출하는 데 사용할 수 있는 거래 빌더입니다.

Demeter Run

Demeter Run은 빠르게 Cardano 환경을 구축하는 데 탁월한 플랫폼입니다. Cardano 개발 환경에서 가장 중요한 측면 중 하나는 [Cardano 노드](#)에 액세스하는 것입니다. 일반적으로는 소스 또는 도커 이미지에서 Cardano 노드를 다운로드한 다음 올바른 구성으로 설정하는 과정을 거칩니다. 프로덕션 환경에서는 Cardano 노드가 별도의 서버에서 16GB 메모리, 100GB 이상의 SSD 디스크, 2개의 CPU 코어와 함께 실행되어야 하며, Cardano 노드 DB [스냅샷](#)을 사용하더라도 동기화하는 데 시간이 소요됩니다.

다행히도, 이때 Demeter Run이 등장합니다. 이를 통해 미리 구성되어 있는 작업 공간을 생성해 줄 것입니다!

Setup

1. Go to <https://demeter.run/> and create an account
2. Select a cluster (US or Europe)
3. Select Discover Plan (free)
4. Select Network (Preview/Preprod/Mainnet)
5. Enter Project name and select Create Project
6. After project has been built select Open
7. Select Setup Dev Workspace
8. Select Clone an existing github repository
 - a. <https://github.com/lley154/helios-examples.git>
9. Select your coding stack as Typescript
10. Select workspace as small
11. Select network to connect to as Preprod
12. Scroll down and Select Create Workspace
13. After the workspace has been completed, select OPEN VSCODE top right on the screen
14. A popup window will appear and select the checkbox to trust the authors of the parent folder workspace. Then choose the button "Yes, I trust the authors"
15. Customize Web VS code as needed
16. Select the hamburger menu (top left) -> Terminal -> New

Terminal Now you have access to your workspace.

Query Tip

다음 명령어를 실행해 보세요.

```
$ cardano-cli query tip --testnet-magic 1
{
  "block": 526708,
  "epoch": 45,
  "era": "Babbage",
  "hash": "258c7fc2d52fce305720f9d8b3fb4376100f0914a63d77efb73c5098833a299f",
  "slot": 18213433,
  "syncProgress": "100.00"
}
```

syncProgress가 100%이므로, 이는 **Cardano** 노드가 **Preprod** 테스트넷의 블록체인과 완전히 동기화되어 있다는 것을 의미합니다.

Workspaces

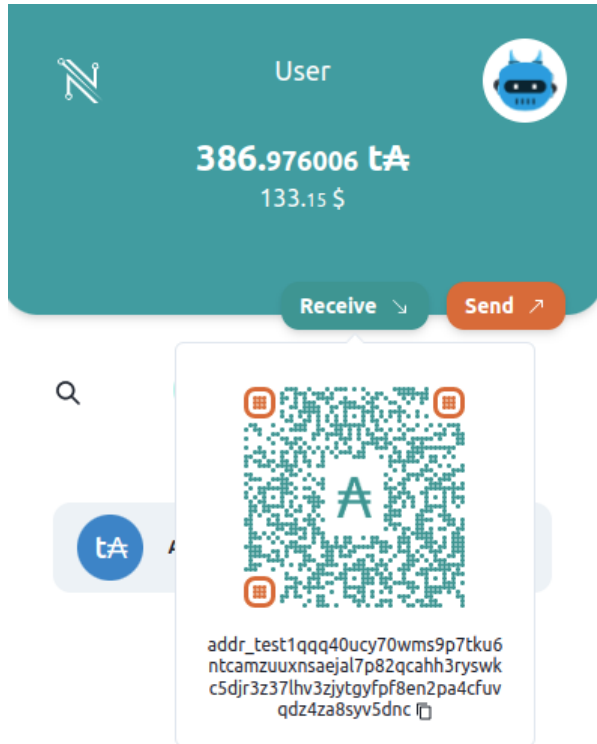
작업 공간은 임시 공간이므로, 작업이 끝나면 꼭 작업 내용을 커밋하고 온라인 **GitHub** 저장소에 푸시하세요.

Addresses

Cardano 주소는 **Cardano** 블록체인의 핵심입니다. **Cardano**에서의 모든 자산은 주소에 위치해야 합니다. 일반적으로 다루게 될 주소에는 두 가지 유형이 있습니다. 기본 주소와 엔터프라이즈 주소입니다.

기본 주소

기본 주소는 브라우저 지갑(예: [Nami](#), [Eternl](#), [Flint](#) 등)에서 에이다를 받을 때 볼 수 있는 주소입니다



① Preprod

이 경우, 다음과 같은 주소를 가지고 있습니다:

Addr_test1qqq40ucy70wms9p7tku6ntcamzuuxnsaejal7p82qcahh3ryswkc5djr3z37lhv3zjytgyfpf8en2pa4cfuvqdz4za8syv5dnc

다음 명령어를 사용하여 주소를 검사할 수 있습니다:

```
$ echo  
addr_test1qqq40ucy70wms9p7tku6ntcamzuuxnsaeja17p82qcahh3ryswkc5djr3z37lhv3zjyt  
gyfpf8en2pa4cfuvqdz4za8syv5dnc | ./utils/cardano-address address inspect  
{  
  "address_style": "Shelley",  
  "address_type": 0,  
  "network_tag": 0,  
  "spending_key_hash":  
  "0157f304f3ddb8143e5db9a9af1dd8b9c34e1dccbbff04ea063b7bc4",  
  "spending_key_hash_bech32":  
  "addr_vkh1q9t1xp8nmkupg0jahx5678wch8p5u8wvh0lsf6sx8daug8cyak6",  
  "stake_key_hash":  
  "6483ad8a364388a3efdd911488b4112149f33507b5c278c03455174f",  
  "stake_key_hash_bech32":  
  "stake_vkh1vjp6mz3kgwy28m7ajy2g3dq3y9ylxdg8khp83sp525t57tzwkxk",  
  "stake_reference": "by value"  
}
```

이 주소는 **address_type**이 0이기 때문에 테스트 주소입니다. 또한 "addr_test" 접두사도 확인할 수 있는데, 이는 테스트넷(예: **preprod**) 주소임을 나타냅니다. **address_type**이 1이면 프로덕션 주소이며 접두사로 "addr"이 붙습니다.

주소는 **spending_key_hash**와 **stake_key_hash** 모두에서 파생됩니다.
spending_key_hash는 공개 키 해시(PKH)라고 더 일반적으로 알려져 있습니다.
staking_key_hash는 지갑에서 에이다를 [스테이크 풀](#) 운영자에게 스테이크할 때 사용됩니다

엔터프라이즈 주소

일반적으로 사용되는 또 다른 주소 유형은 엔터프라이즈 주소라고 불립니다.

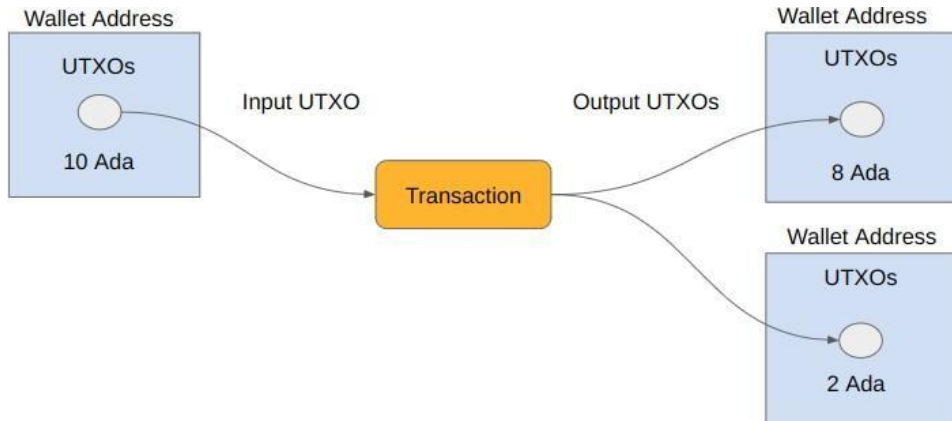
```
$ echo addr_test1vq5s7k4kwqz4rrfe8mm9jz9tpm7c5u93yfwwsaw708yxs5sm70qjg |  
./utils/cardano-address address inspect  
{  
  "address_style": "Shelley",  
  "address_type": 6,  
  "network_tag": 0,  
  "spending_key_hash":  
"290f5ab67005518d393ef65908ab0efd8a70b1225ce875de79c86852",  
  "spending_key_hash_bech32":  
"addr_vkh19y844dnsq4gc6wf77evs32cwlk98pvfztn58thneep59yhwpvvm",  
  "stake_reference": "none"  
}
```

엔터프라이즈 주소는 **stake_key_hash**가 없고 오직 **spending_key_hash**만 가지고 있습니다.

cardano-address를 설치하고 암호문구에서 주소를 파생하는 방법에 대한 자세한 정보는 부록 섹션의 주소 키 파생을 참조해 주세요.

Transactions

여러 트랜잭션은 한 주소에서 다른 주소로 자산을 전송하는 데 사용됩니다. 모든 트랜잭션에는 하나 이상의 입력과 하나 이상의 출력이 포함됩니다. 다음은 **Cardano** 트랜잭션을 사용하여 에이다를 전송하는 간단한 다이어그램입니다.



UTXO는 사용되지 않은 트랜잭션 출력입니다. 트랜잭션은 하나 이상의 **UTXO**를 소비한 다음 주소에서 잠긴 각 **UTXO**에 하나 이상의 **UTXO**를 생성합니다. 주소에서 자산의 양을 확인하려고 할 때, 실제로는 주소에서 잠긴 각 **UTXO**의 자산 금액을 세고 있습니다.

다음은 주소에서 2개의 UTXO의 예시입니다. *

```
$ cardano-cli query utxo --address addr_test1vzu...dxn7 --cardano-mode
--testnet-magic 1
TxHash TxIx Amount
-----
158...925 0 3551912116 lovelace + TxOutDatumNone
b4e...0d2 0 5000000 lovelace + TxOutDatumNone
```

* 주소와 TxHash가 이 페이지에 적절하게 들어갈 수 있도록 줄여진 것에 유의해 주세요.

여러 관찰 결과:

- 이 예시에서 2개의 UTXO가 있습니다.
- 인덱스 0의 트랜잭션 해시 158...925는 3,551,912,116 러브레이스를 가지고 있습니다.
- 인덱스 0의 트랜잭션 해시 b4e...0d2는 5,000,000 러브레이스를 가지고 있습니다.
- 1 에이다에는 1,000,000 러브레이스가 있습니다.
- 주소에서 잠긴 에이다의 양은 다음과 같습니다.

$$\begin{array}{r} 3,551.912116 \\ + 5.000000 \\ \hline 3,556.912116 \end{array}$$

Datums

이전 예제에서 **UTXO** 출력의 끝에 **TxOutDatumNone**이 있는 것을 볼 수 있습니다. 이 필드는 **UTXO**에 지속적인 정보를 저장하는 데 사용되며 데이텀이라고 합니다. 데이텀은 트랜잭션을 빌드하고 제출할 때 생성됩니다. **TxOutDatumNone**은 이 **UTXO**와 연관된 데이텀 정보가 없음을 나타냅니다. 다음은 **UTXO**에 인라인 데이텀이 포함된 예시입니다.

```
cardano-cli$ cardano-cli query utxo --address addr_test1wrm...hm3f
--cardano-mode --testnet-magic 1
TxHash TxIx Amount
```

```
-----
b07...91da 0 88270000 lovelace + TxOutDatumInline ... (ScriptDataConstructor 0
[ScriptDataNumber 87770000,ScriptDataBytes "4666620215361",ScriptDataBytes
"0.34180"])
b87...107e 1 200000000 lovelace + TxOutDatumNone
```

TxOutDatumInline은 인라인 데이텀이 있음을 나타냅니다. 데이텀은 실제 데이텀의 해시값일 수도 있습니다. 데이텀 크기가 크지 않은 경우, 인라인 데이텀이 일반적으로 선호되는 방식입니다. 데이텀 크기가 큰 경우, **Vasil** 하드포크 전까지는 데이텀 해시를 사용하는 것이 유일한 옵션이었습니다.

Redeemers

여러 리디머는 스마트 계약 트랜잭션에 포함되어 있으며, 리디머의 값에 따라 다른 조건을 실행하는 데 사용할 수 있습니다. 또한, 여러 리디머는 트랜잭션 생성 중에 런타임에서 스마트 계약에 데이터를 전달하는 데 사용될 수 있습니다.

메타데이터

트랜잭션은 체인상에 저장되는 메타데이터를 생성할 수도 있습니다. 이를 통해 트랜잭션에 대한 인간이 읽을 수 있는 정보를 제공하고 응용 프로그램에서 활용할 수 있습니다. (여러 데이터 및/또는 메타데이터) 기술은 모두 체인상에 지속적인 데이터를 저장합니다. 어떤 기술을 사용할지 결정하는 것은 데이터에 액세스하는 방식에 따라 달라집니다. 데이터가 스마트 계약 내에서 유효성을 검증해야 하는 경우 데이터를 사용하고, 오프체인 응용 프로그램, 토큰 발행 또는 인간 확인에 대한 트랜잭션 세부 정보를 제공해야 하는 경우 메타데이터를 사용하세요.

Cardano 블록체인 아키텍처

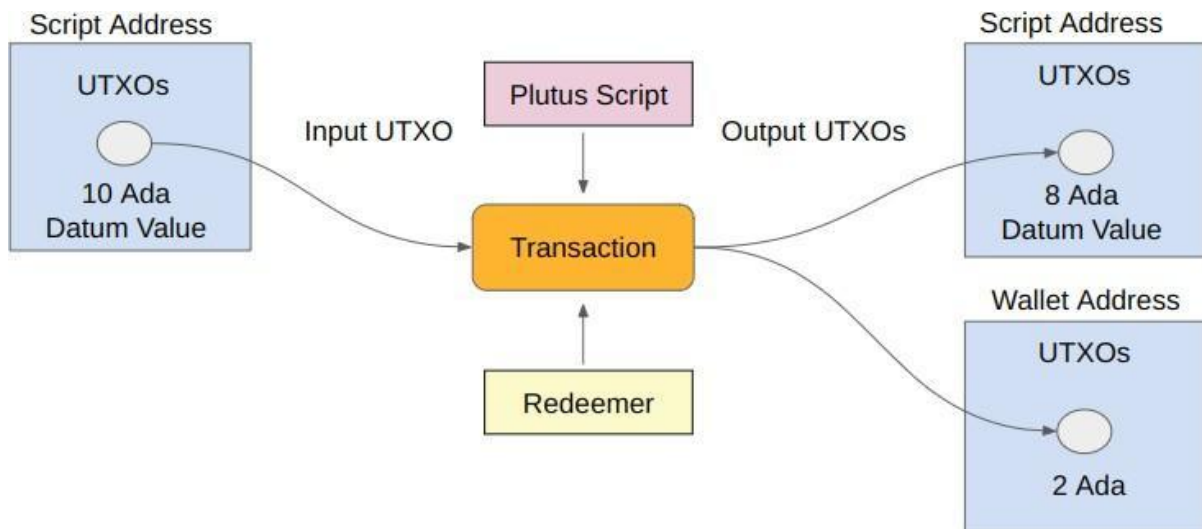
Cardano는 확장(*extended*) UTXO 블록체인 아키텍처를 사용합니다. 이것은 자산에 대한 UTXO를 사용하는 것 외에도, UTXO에 데이터 값을 추가할 수 있는 기능을 제공합니다.

UTXO 모델

UTXO 모델 (예: 비트코인)에서는, 주소에 대한 개인 키의 소유자만 해당 주소에 잠긴 UTXO를 사용할 수 있습니다.

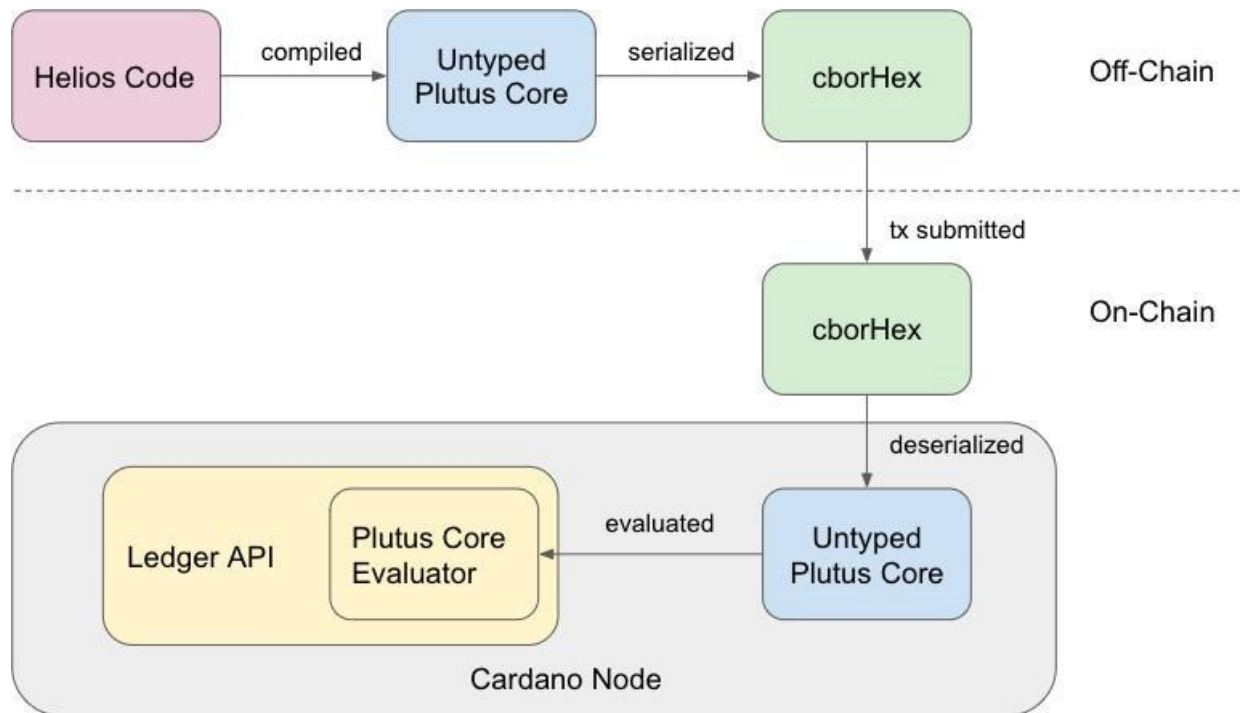
eUTXO 모델

eUTXO 모델 (예: Cardano)에서는, [튜링 완전](#) 스크립트가 스크립트 주소에 잠긴 UTXO를 사용할 수 있습니다. 아래는 10 Ada가 Plutus 스크립트 주소에 잠긴 스마트 계약 트랜잭션을 보여주는 간단화된 다이어그램입니다.



Plutus Scripts

Helios 코드는 **Plutus Core Evaluator**가 필요로하는 **plutus** 스크립트 형식인 **Untyped Plutus Core**로 컴파일됩니다. **Helios**로 트랜잭션을 컴파일하고 제출할 때, 스마트 계약은 먼저 로컬 **Helios** 가상 머신에서 평가됩니다. 이는 **Cardano**의 [결정론적](#) 특성을 활용합니다. 즉, 트랜잭션이 성공할지 실패할지 및 트랜잭션 수수료를 제출하기 전에 알 수 있습니다. 아래는 **plutus** 스크립트를 **Cardano** 노드에 전달하고 트랜잭션의 일부로서 그것을 평가하는 과정을 보여주는 단순화된 다이어그램입니다.



트랜잭션 서명

Cardano 블록체인 상에서 트랜잭션이 성공적으로 실행되기 위해서는, 소비하려는 **UTXO(s)**의 소유자가 이에 동의한 것을 증명하는 유효한 서명(**signature**)을 포함해야 합니다. 이 동의는 트랜잭션 본문의 해시를 소유자의 개인키로 서명하는 것으로 이루어집니다. 그리고 이 서명들은 묶음(**witness set**) 트랜잭션에 포함되어집니다. 각 소비하려는 **UTXO**의 서명이 묶음(**witness set**)에 포함되면, 해당 트랜잭션은 블록체인으로 검증을 위해 전송될 수 있습니다.

Cardano-cli

Cardano-cli는 **Cardano** 블록체인에서 트랜잭션을 조회하고 실행하는 데에 널리 사용되는 명령 줄 도구입니다. 이 도구를 사용하면 트랜잭션이 어떻게 구성되고, 어떻게 서명되며, 제출되는지에 대한 이해도를 높일 수 있습니다. **Cardano-cli**는 백엔드 서버 및 배치 처리에도 사용됩니다.

Bash 셸 스크립트

`cardano-cli` 도구를 명령 줄에서 사용하는 것은 불편하므로, `bash` 셸 스크립트나 `Node.js` 내에서 사용하는 것이 더 편리합니다. 다음은 간단한 `cardano-cli` 트랜잭션을 실행하는 데 사용되는 `bash` 셸 스크립트입니다.

```
#!/usr/bin/env bash
source_addr=addr_test1vq5s7k4kwqz4rrfe8mm9jz9tpm7c5u93yfwsw708yxs5sm70qjg
source_utxo=3188f0f28667f753f864b39475acb3d55cf27e146fa6414ac57f5c6c63c705c2#0
destination_addr=addr_test1qqq967dwdp009smfeqtzhve89fyuqjydkvwc9md5atyg2429gnmszjc7hyf
685vp7qxeffjd568s3p234fg5ryhrkvjsn7muqm
user_skey=/config/workspace/repo/.keys/user/key.skey
network="--testnet-magic 1"

# cardano-cli 도구에서 매개변수 파일을 생성합니다.
cardano-cli query protocol-parameters $network --out-file pparams.json

cardano-cli transaction build --babbage-era --cardano-mode \
  $network \
  --change-address "$source_addr" \
  --tx-in "$source_utxo" \
  --tx-out "$destination_addr+2000000" \
  --protocol-params-file pparams.json \
  --out-file transfer-tx-alonzo.body
echo "tx has been built"

cardano-cli transaction sign \
  --tx-body-file transfer-tx-alonzo.body \
  $network \
  --signing-key-file "$user_skey" \
  --out-file transfer-tx-alonzo.tx
echo "tx has been signed"

echo "Submit the tx with plutus script and wait 5 seconds..."
cardano-cli transaction submit --tx-file transfer-tx-alonzo.tx $network
```


Test Drive

여기는 소스 주소와 목적지 주소의 이전 스냅샷입니다.

소스 지갑 주소

```
$ cardano-cli query utxo --address addr_test1vq5...qjg --cardano-mode
--testnet-magic 1
TxHash TxIx Amount
-----
db34...3ee 0 10000000 lovelace + TxOutDatumNone
```

목적지 지갑 주소

```
$ cardano-cli query utxo --address addr_test1vq7k...pds --cardano-mode
--testnet-magic 1
TxHash TxIx Amount
-----
```

그럼 이제 스크립트를 실행합니다.

```
$ ./transfer-tx.sh preprod
Estimated transaction fee: Lovelace 165721
tx has been built
tx has been signed
Submit the tx with plutus script and wait 5 seconds...
Transaction successfully submitted.
```

다음은 소스와 목적지 주소의 트랜잭션 후 스냅샷입니다.

소스 지갑 주소

```
$ cardano-cli query utxo --address addr_test1vq5s...qjg --cardano-mode  
--testnet-magic 1
```

```
TxHash TxIx Amount
```

```
-----  
996...95c 1 7834279 lovelace + TxOutDatumNone
```

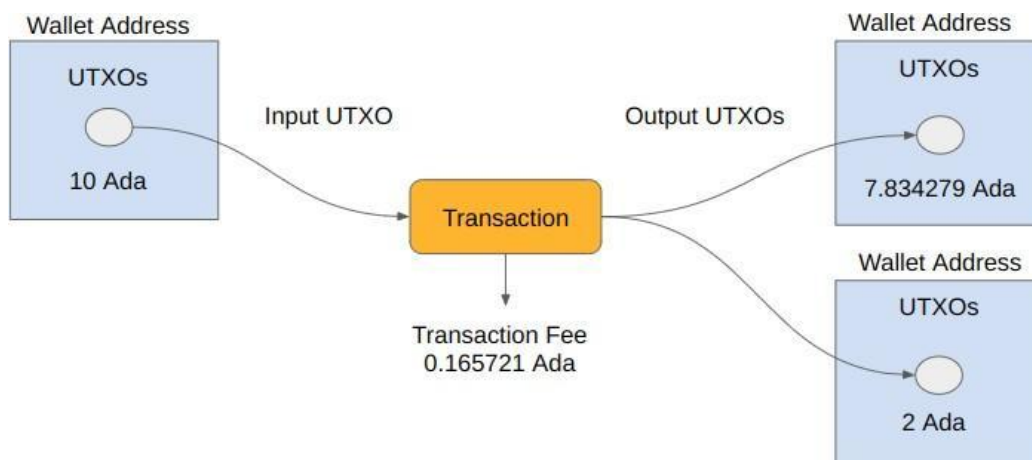
목적지 지갑 주소

```
$ cardano-cli query utxo --address addr_test1vq7...pds --cardano-mode  
--testnet-magic 1
```

```
TxHash TxIx Amount
```

```
-----  
996...95c 0 2000000 lovelace + TxOutDatumNone
```

따라서 수수료를 포함한 간단한 트랜잭션 다이어그램은 다음과 같습니다.



Ada 전송

Helios는 **plutus**로 컴파일되는 언어이자 오프체인 트랜잭션 빌더입니다. 이 섹션에서는 프론트엔드 웹 애플리케이션을 사용하여 **Helios**를 사용하여 **Ada**를 전송하는 트랜잭션을 구성하는 방법을 살펴보겠습니다.

지갑

크롬 브라우저에 설치된 지갑 확장 프로그램이 필요합니다. 이 책의 예제에서는 [Nami 지갑](#)을 사용합니다. **Nami** 지갑을 설치한 후, 다음 단계를 따라서 지갑이 올바른 네트워크에 연결되어 있는지 확인하세요.

1. Select the Extensions icon on your browser and select Nami wallet
2. Select the account icon image (top right)
3. Select Settings near the bottom of the popup window
4. Select Network
5. Select Preprod in the drop down menu
6. Go back to the main view and select the account icon image again
7. Now select New Account
8. Create another account name & password
9. You can switch between accounts by selecting the account icon image

테스트넷 포셋(Faucet)

당신은 일부 테스트 Ada를 구할 필요가 있습니다.

1. Go to the Nami account you want to receive some Ada with and select the Receive button
2. Copy the receiving address to your clipboard
3. Open a new browser tab and go to the [Cardano testnet faucet](#) page
4. In the Environment dropdown, select the Preprod Network
5. In the Address field, past the receiving address you copied in step 2
6. Select Request Funds
7. You may have to wait 10-60 seconds before the funds are in your wallet.

Note: 24시간 내에는 한 번만 테스트 Ada를 요청할 수 있습니다.

포트 노출

Demeter를 사용하여 웹 앱에 액세스하려면 웹 앱이 바인드하는 포트를 노출해야 합니다. 기본 포트는 **3000**이므로 이 포트를 노출하고 웹 애플리케이션에 액세스할 수 있는 **URL**을 제공받습니다.

1. Select your workspace on your Demeter Dashboard
2. Select the Exposed Ports tab
3. Select Expose Port+ button
4. Enter a port name (eg Next.js) and port number (eg 3000)

Running Next.js 어플리케이션

다음과 같이 실행하여 **VSCode Web** 터미널 창에서 **Next.js** 어플리케이션을 설정하고 시작하세요.

```
$ cd transfer-ada  
$ npm install  
$ npm run dev
```

위에서 제공된 노출된 포트 **URL**을 사용하여 **Next.js** 애플리케이션에 액세스하세요.

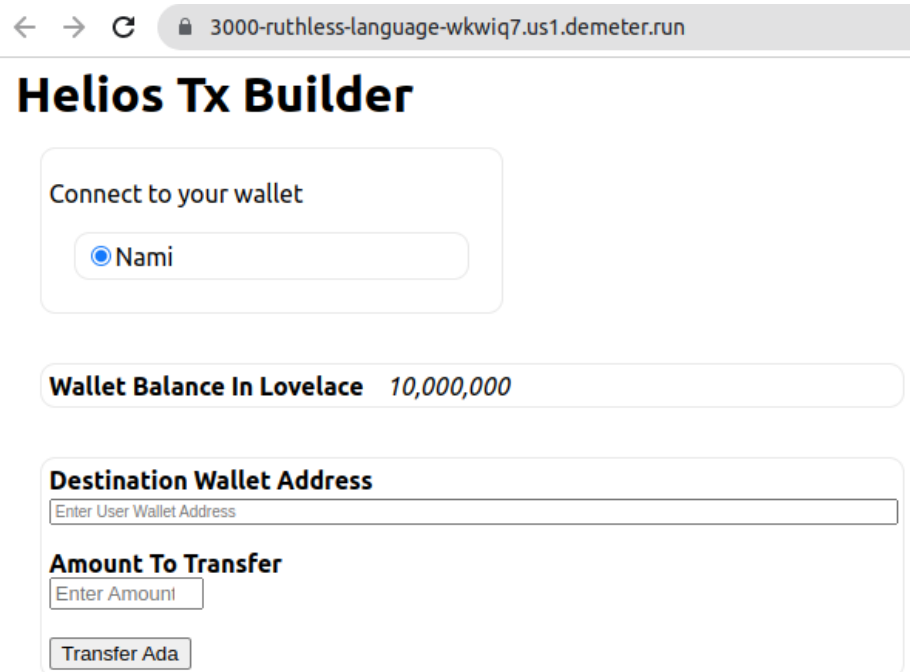
Test Drive

다음으로 Next.js 어플리케이션이 실행 중이므로, 지갑을 연결하기 위해 Nami 라디오 버튼을 선택하세요.



A screenshot of a web browser showing the Helios Tx Builder interface. The browser's address bar displays the URL `3000-ruthless-language-wkwiq7.us1.demeter.run`. The page title is "Helios Tx Builder". Below the title, there is a section titled "Connect to your wallet" which contains a single radio button labeled "Nami".

선택한 지갑의 사용 가능한 러브레이스 잔액이 표시됩니다.

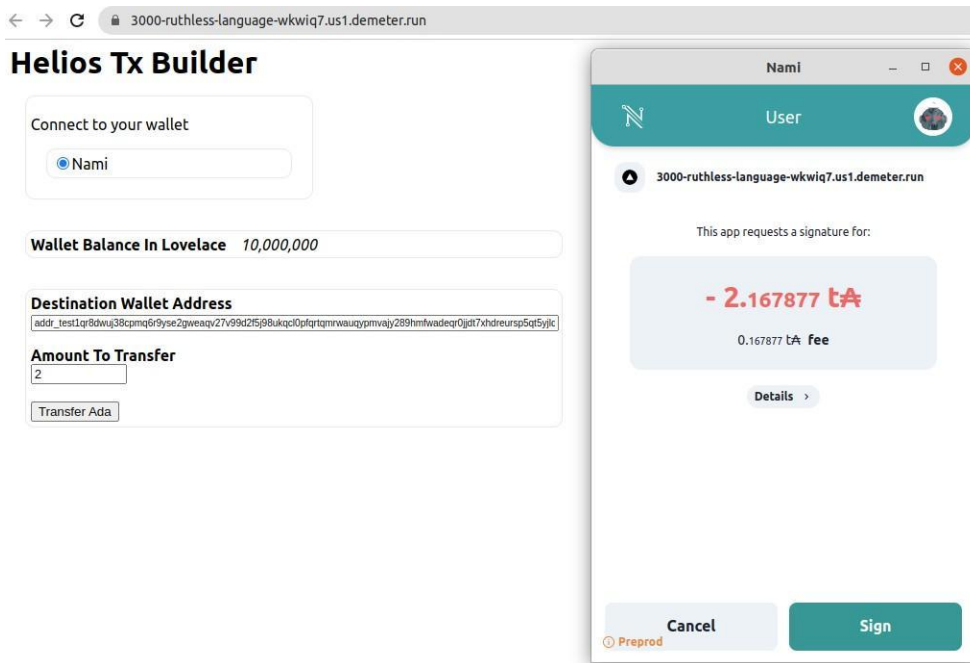


A screenshot of the Helios Tx Builder interface. The browser's address bar shows the same URL. The "Connect to your wallet" section now shows the "Nami" radio button as selected. Below this, a box displays "Wallet Balance In Lovelace 10,000,000". Further down, there is a section titled "Destination Wallet Address" with a text input field containing the placeholder "Enter User Wallet Address". Below that is a section titled "Amount To Transfer" with a text input field containing the placeholder "Enter Amount". At the bottom of this section is a button labeled "Transfer Ada".

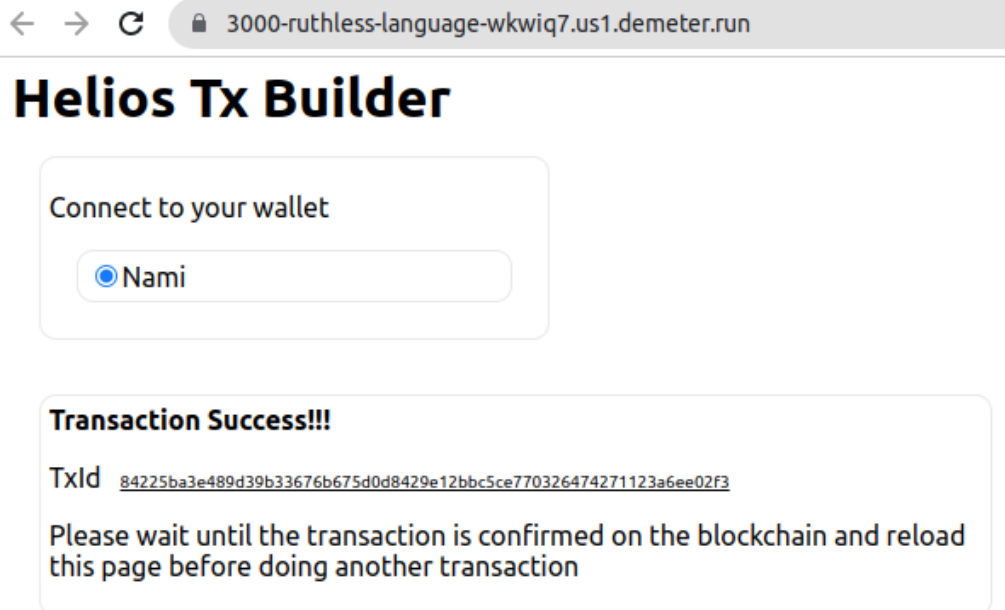
계정 잔액이 없는 다른 **Nami** 계정의 경우, 해당 계정으로 전환하고 목적지 주소를 복사합니다. 이체할 수 있는 금액이 있는 계정으로 다시 전환해야 합니다.

목적지 주소와 송금하려는 **Ada** 금액을 입력하고 **Transfer Ada** 버튼을 선택합니다.

Nami 지갑이 나타나면 거래에 서명할 수 있습니다.



지갑 비밀번호를 입력하여 서명한 후, 다음과 같은 성공 메시지가 표시되어야 합니다.



블록체인 익스플로러 [cexplorer](#)를 사용하여 거래를 확인하거나 Nami 지갑에서 확인할 수 있습니다. Txid 링크를 선택하여 확인할 수 있습니다.

자세한 내용

첫 번째 단계는 브라우저 지갑이 존재하는지 여부를 감지하는 것입니다.

```
const checkIfWalletFound = async () => {

  let walletFound = false;
  const walletChoice = whichWalletSelected;
  if (walletChoice === "nami") {
    walletFound = !!window?.cardano?.nami;
    return walletFound;
  }
}
```

두 번째 단계는 Tx builder가 사용할 수 있는 API 객체를 생성하는 것입니다.

```
const enableWallet = async () => {

  try {
    const walletChoice = whichWalletSelected;
    if (walletChoice === "nami") {
      const handle: Cip30Handle
        = await window.cardano.nami.enable();
      const walletAPI = new Cip30Wallet(handle);
      return walletAPI;
    }
  } catch (err) {
    console.log('enableWallet error', err);
  }
}
```

지갑의 UTXO와 지갑의 잔액을 확인하고 거스름돈 주소를 가져오는 데 사용되는 WalletHelper를 사용하여 Helios Tx를 빌드할 수 있습니다.

```
...
const adaAmountVal = new Value(BigInt((adaQty)*1000000));

// 지갑의 UTXO를 가져오기
const walletHelper = new WalletHelper(walletAPI);
const utxos = await walletHelper.pickUtxos(adaAmountVal);

// 거스름돈 주소 가져오기
const changeAddr = await walletHelper.changeAddress;

// 트랜잭션 빌드 시작

const tx = new Tx();
tx.addInputs(utxos[0]);

// 목적지 주소와 보낼 Ada의 양을 추가합니다.
tx.addOutput(new TxOutput(Address.fromBech32(address), adaAmountVal));

const networkParams = new NetworkParams(
  await fetch(networkParamsUrl)
    .then(response => response.json())
)
// 구매자에게 거스름돈을 돌려보내세요.
await tx.finalize(networkParams, changeAddr);

console.log("Verifying signature...");
const signatures = await walletAPI.signTx(tx);
tx.addSignatures(signatures);

console.log("Submitting transaction...");
const txHash = await walletAPI.submitTx(tx);
...
```

민팅

민팅은 **UTXO**에 포함되어 주소에 잠긴 새로운 네이티브 토큰을 생성하는 블록체인 상의 과정입니다. 네이티브 토큰은 **Ada**와 동일한 특성을 가지며 블록체인에서 **1급 시민 (first class citizen)**으로 간주됩니다. 민팅 정책 스크립트는 토큰을 생성/소각하는 효율적인 매커니즘으로, 민팅 또는 소각 트랜잭션의 규칙을 정의합니다.

NFT

NFT는 분할할 수 없는 대체 불가능한 토큰으로, 단 하나만 존재한다는 의미입니다. 따라서 단 하나의 고유 토큰만 생성하는 가장 좋은 방법은 **UTXO**의 고유 속성을 활용하는 것입니다. 다시 한 번 말씀드리자면, **UTXO**는 사용되지 않은 트랜잭션 출력입니다. 따라서 **UTXO**가 소비되면 더 이상 사용되지 않은 트랜잭션 출력이 아닙니다. 따라서 트랜잭션 입력에 **UTXO**를 포함시켜서 소비하기만 한다면, 다른 트랜잭션에서는 해당 **UTXO**를 소비할 수 없게 됩니다.

민팅 스마트 컨트랙트

각 **Helios** 스마트 컨트랙트는 검증 로직을 실행하는 **func main**을 포함해야 합니다. **Plutus script**의 종류를 지정하기 위해 첫 번째 줄에 **minting**과 **NFT** 프로그램 이름을 명시합니다.

```
minting nft
```

다음으로 사용할 **UTXO**를 지정하기 위해 매개변수 값을 하드 코딩해야 합니다. 따라서 **"6e1...996"**의 트랜잭션 ID와 **0**의 트랜잭션 인덱스를 모두 포함하는 **TxOutputId** 객체를 생성해야 합니다.

```
const TX_ID: ByteArray = #6e1...996
const txId: TxId = TxId::new(TX_ID)
const outputId: TxOutputId = TxOutputId::new(txId, 0)
```

컨트랙트 파라미터로 **UTXO**를 설정했기 때문에, 이 파라미터 값에 따라 (고유한) 민팅 정책 해시가 달라집니다.

func main은 **Helios plutus** 스크립트에서 필수적인 함수입니다. 여기에서 **ScriptContext**와 같은 중요한 객체에 액세스할 수 있습니다. **ScriptContext** 객체는 현재 트랜잭션(**#1**)에 대한 정보를 가지고 있으며 이 정보는 검증 로직에 사용됩니다. 또한, 이 민팅 정책 스크립트의 민팅 정책 해시(**#2**)를 얻고 이를 사용하여 우리가 민팅하는 토큰(**#3**)을 나타내는 자산 클래스 객체를 구성합니다. 마지막으로, **ScriptContext Tx** 객체를 통해 이 트랜잭션의 실제로 민팅된 토큰(**#4**)을 얻습니다.

```
func main(ctx: ScriptContext) -> Bool {
    tx: Tx = ctx.tx; // #1
    mph: MintingPolicyHash = ctx.get_current_minting_policy_hash(); // #2
    assetclass: AssetClass = AssetClass::new(
        mph,
        "NFT Token Name".encode_utf8()
    ); // #3
    value_minted: Value = tx.minted; // #4
```

이제 실제 유효성 검사를 수행할 준비가 되었습니다. 먼저 **NFT**가 올바른 민팅 정책 해시, 토큰 이름, 1개의 수량을 가지고 이 트랜잭션의 일부로 발행된 것인지 확인합니다.

```
(value_minted == Value::new(assetclass, 1)).trace("NFT1: ")
```

두 번째로는, 스마트 컨트랙트에 파라미터로 지정한 **UTXO**가 실제로 트랜잭션 입력으로 포함되었는지 확인합니다.

```
tx.inputs.any((input: TxInput) -> Bool {  
    (input.output_id == outputId).trace("NFT2: ")
```

`.trace("NFT: ")` 함수는 유효성 검사기가 거짓을 반환하는 경우 평가된 값과 함께 추적 오류 메시지를 생성합니다. 이는 어떤 조건이 참이고 어떤 조건이 거짓인지 구분하는 데 매우 유용합니다.

따라서 전체적으로 완성된 **NFT** 민팅 스마트 컨트랙트는 다음과 같습니다.

```
minting nft
```

```
const TX_ID: ByteArray = #6e1...996  
const txId: TxId = TxId::new(TX_ID)  
const outputId: TxOutputId = TxOutputId::new(txId, 0)  
func main(ctx: ScriptContext) -> Bool {  
    tx: Tx = ctx.tx;  
    mph: MintingPolicyHash = ctx.get_current_minting_policy_hash();  
    assetclass: AssetClass = AssetClass::new(  
        mph,  
        "NTF Token Name".encode_utf8()  
    );  
    value_minted: Value = tx.minted;  
    // 유효성 검사기 로직 시작  
    (value_minted == Value::new(assetclass, 1)).trace("NFT1: ") &&  
    tx.inputs.any((input: TxInput) -> Bool {  
        (input.output_id == outputId).trace("NFT2: ")  
    })  
}
```


스크립트 커파일

스마트 컨트랙트 코드를 블록체인으로 가져올 수 있는 다양한 옵션이 있습니다.

- 스크립트를 **Next.js** 앱 내부에 저장하고 브라우저에서 컴파일한 다음 트랜잭션의 일부로 제출하는 방법
- 미리 스크립트를 컴파일하고 트랜잭션의 일부로 제출하는 방법
- 미리 스크립트를 컴파일하고, **plutus** 참조 스크립트로 로드하여 체인에 저장합니다. 다른 트랜잭션에서는 이를 참조하여 사용할 수 있으며, 이 경우 스크립트의 복사본이 필요하지 않습니다.

간단한 스마트 컨트랙트의 경우, 우리는 구축한 **Next.js** 앱에 스크립트를 포함시키고, 필요할 때마다 즉석에서 **plutus** 코드로 컴파일하고 트랜잭션을 제출할 것입니다. **Helios** 컴파일러가 자바스크립트로 작성되어 있다는 점은 **plutus** 코드로 컴파일할 때 유연성을 제공합니다.

아래는 **mintNFT** 함수에 대한 온체인 및 오프체인 코드이며, 여기에는 런타임 변수를 계약 매개변수로 사용하여 **Helios** 코드를 동적으로 컴파일하는 것을 포함합니다. 또한 트랜잭션에 첨부할 수 있도록 메타데이터 데이터 타입이 구체적으로 지정되어야 함에 유의하세요.

```
const mintNFT = async (params : any) => {

    const address = params[0];
    const name = params[1];
    const description = params[2];
    const img = params[3];
    const minAda: number = 2000000; // NFT 전송에 필요한 최소 러브레이스 수
    const maxTxFee: number = 500000; // 최대 예상 트랜잭션 수수료
    const minChangeAmt: number = 1000000; // 거스름돈으로 돌려보내야 하는 최소 러브레이스 수

    const minAdaVal = new Value(BigInt(minAda));
    const minUTXOVal = new Value(BigInt(minAda + maxTxFee + minChangeAmt));

    // 지갑의 UTXO를 가져오기
    const walletHelper = new WalletHelper(walletAPI); const
    utxos = await walletHelper.pickUtxos(minUTXOVal);
    // 거스름돈 주소 가져오기
    const changeAddr = await walletHelper.changeAddress;
    // 담보로 사용되는 UTXO를 결정합니다.
    const colatUtxo = await walletHelper.pickCollateral();
    // 트랜잭션 빌드 시작
    const tx = new Tx();
    // UTXO를 입력으로 추가합니다.
    tx.addInputs(utxos[0]);
```

```

const mintScript = `minting nft

const TX_ID: ByteArray = #` + utxos[0][0].txId.hex + `
const txId: TxId = TxId::new(TX_ID)
const outputId: TxOutputId =
    TxOutputId::new(txId, ` + utxos[0][0].utxoIdx + `)

func main(ctx: ScriptContext) -> Bool {
    tx: Tx = ctx.tx;
    mph: MintingPolicyHash = ctx.get_current_minting_policy_hash();

    assetclass: AssetClass = AssetClass::new(
        mph,
        "` + name + `".encode_utf8()
    );
    value_minted: Value = tx.minted;

    // Validator logic starts
    (value_minted == Value::new(assetclass, 1)).trace("NFT1: ") &&
    tx.inputs.any((input: TxInput) -> Bool {
        (input.output_id == outputId).trace("NFT2: ")
    })
} `

// Helios 민팅 스크립트를 컴파일합니다.
const mintProgram = Program.new(mintScript).compile(optimize);
// 스크립트를 트랜잭션에 증인(witness)으로 추가합니다.
tx.attachScript(mintProgram);
// NFT를 출력으로 보낼 것으로 구성합니다.
const nftTokenName = ByteArrayData.fromString(name).toHex();
const tokens: [number[], bigint][] =
    [[hexToBytes(nftTokenName), BigInt(1)]];

```

```

// 다음은 plutus 스크립트 트랜잭션에 대해 항상 리더머를 보내야 하므로 (사용하지 않는 경우도 있음)
// 빈 Redeemer를 생성합니다.
const mintRedeemer = new ConstrData(0, []);
// 이 트랜잭션의 일부로 포함할 민팅을 표시합니다.
tx.mintTokens(
  mintProgram.mintingPolicyHash,
  tokens,
  mintRedeemer
)
// 최소 Ada와 함께 민팅된 NFT를 포함하여 출력물을 구성합니다.
tx.addOutput(new TxOutput(
  Address.fromBech32(address),
  new Value(minAdaVal.lovelace, new Assets([[mintProgram.mintingPolicyHash,
                                              tokens]]))
));
// 담보로 사용할 UTXO를 추가합니다.
tx.addCollateral(colatUtxo);
const networkParams = new NetworkParams(
  await fetch(networkParamsUrl)
    .then(response => response.json())
)
// NFT 생성 트랜잭션에 메타데이터를 첨부합니다.
tx.addMetadata(721, {"map":
  [[mintProgram.mintingPolicyHash.hex,
    {"map": [[name, { "map": [{"name", name},
                             ["description", description],
                             ["image", img]
                           ]
                        ]
    }
  ]
}
]);

```

```
// 해당 거래에서 생긴 잔돈을 구매자에게 보내줍니다.  
await tx.finalize(networkParams, changeAddr);  
  
console.log("Verifying signature...");  
const signatures = await walletAPI.signTx(tx);  
tx.addSignatures(signatures);  
  
console.log("Submitting transaction...");  
const txHash = await walletAPI.submitTx(tx);  
  
console.log("txHash", txHash.hex );  
setTx({ txId: txHash.hex });  
}
```

Test Drive

당신은 이제 **Next.js** 애플리케이션을 시작하고 트랜잭션을 제출하여 자신의 **NFT**를 발행할 수 있습니다.

```
$ cd nft
$ npm install
$ npm run dev
```

Demeter Run에서 워크스페이스의 포트 **3000**을 노출시켜야 합니다. 이를 위해서는 **Demeter Run**의 **"Exposed Ports"** 탭에서 노출시키고자 하는 포트를 추가해야 합니다. 그 후 **"Exposed Ports"** 탭에 있는 **URL**을 사용하여 **Next.js** 애플리케이션에 접속할 수 있습니다.

트랜잭션 제출

지갑을 선택한 후 필요한 정보를 입력하세요.

[←](#) [→](#) [↻](#) [🔒 3000-precious-baseball-lfkchu.us1.demeter.run](#)

Helios Tx Builder

Connect to your wallet

☒ Nami

Wallet Balance In Lovelace 1,455,260,289

Destination Wallet Address

addr_test1qz7gedd4dv6y57dxc0l2awp2uq08tzt5ygyuzrs2qvnrcr5378p8rj3dgxkjg5knpa3uvtwgqd48t666ad05eexasgq5pnv

NFT Token Name

Mad Dog

NFT Description

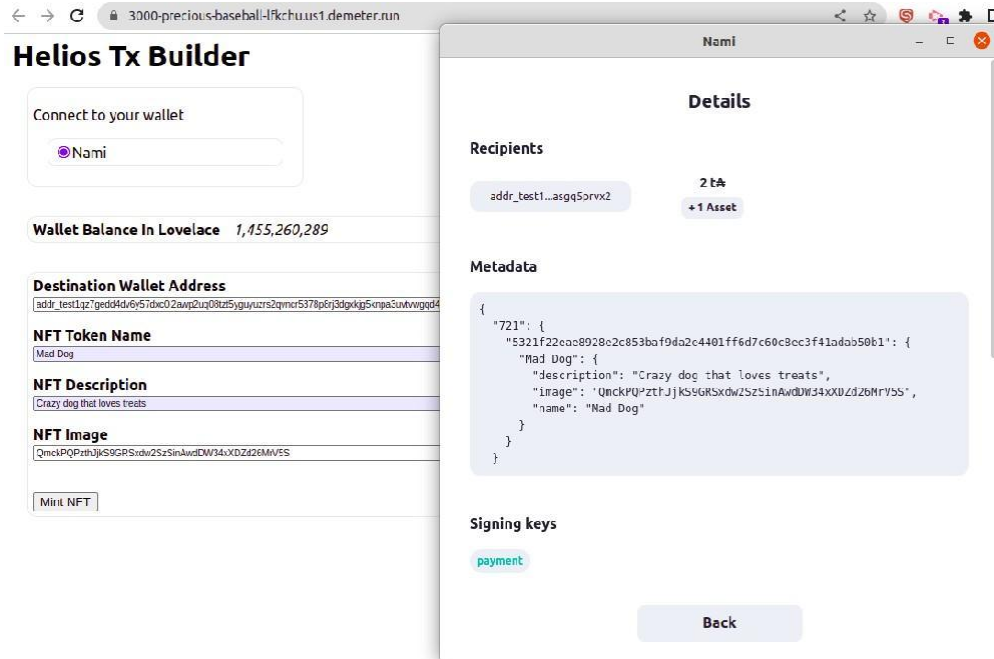
Crazy dog that loves treats

NFT Image

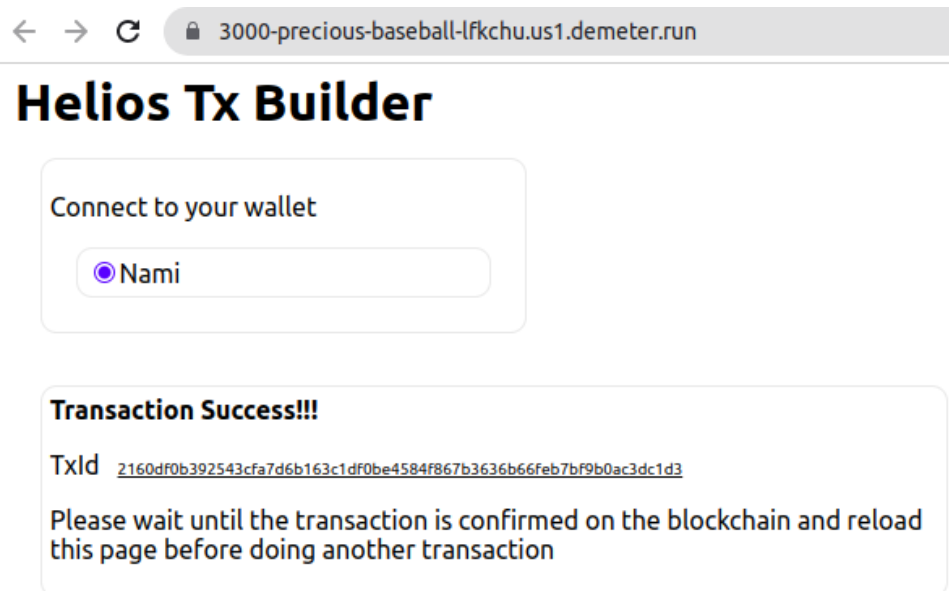
QmckPQPzthJjkS9GRSxdw2SzSinAwdDW34xXDZd26MrV5S

Mint NFT

Mint NFT 버튼을 선택하고 거래에 서명하라는 메시지가 나오면, Nami 지갑의 세부 정보 탭에서 메타데이터를 포함한 NFT가 블록체인 상에 발행되는 것을 확인할 수 있습니다.



트랜잭션을 제출한 후 10-60초를 기다린 후 [블록체인 익스플로러](#)에서 NFT를 확인할 수 있어야 합니다.




<https://preprod.cexplorer.io/tx/2160df0b392543cfa7d6b163c1df0be4584f867b3636b66feb7bf9b0ac3dc1d3>

Cexplorer

Cexplorer에서 NFT를 보려면 다음 단계를 수행하세요:

1. Select the TxId link in the Transaction Success box
2. Scroll down and select the Mint Tab
3. Select that Asset (in this case Mad Dog)
4. And then you will see all the properties of this NFT

← → ↻ 🔒 preprod.cexplorer.io/asset/asset1urm04nzzj2qru43p08swg7rzztw9kl7ru73hdy/preview#data

 **Cexplorer.io** preprod

CARDANO EXPLORER

Dashboard

Watchlist

Pools

Assets

Blocks

Transactions

dApps

Metadata

More

EDUCATION

Articles

Videos

Wiki


ANALYTICS

Decentralization

Network


Assets / Policy 5321f22eae8928e2c853baf... / Mad Dog / NFT preview

☆ Mad Dog > NFT preview

Fingerprint:  asset1urm04nzzj2qru43p08swg7rzztw9kl7ru73hdy

Name (onchain)

Mad Dog

Encoded:  4d616420446f67


Onchain data (epoch snapshot)


Supply


1


Owner


stake_test...7n4dr


 Preview

 TXs

 Mints


 Metadata

 Owners

 Embed

Mad Dog

Note: Data are loaded directly from blockchain and decentralised ipfs storage and are NOT stored or hosted on our website.



Metadata

```
{
  "name": "Mad Dog",
  "image": "QmckPQPzthjK59GRSxdw2SzSInAwdDw34xxDZd26MrV5S",
  "description": "Crazy dog that loves treats"
}
```

다중 서명(Multisig) NFT

NFT 민팅시 다중 서명(**multisig**) 서명을 사용하는 이유는 바로 블록체인에 실제로 민팅 및 제출하기 전에 구매자가 얻는 것을 확인할 수 있도록 하기 위해서입니다. 민팅 정책에 구매자와 판매자의 공개 키 해시를 포함시켜 둘으로써 양측 서명이 모두 완료될 때까지 **NFT** 민팅이 가능하지 않도록 보장할 수 있습니다.

일반적으로 애플리케이션은 트랜잭션 및 증인(**witness**) 업데이트를 저장해야 합니다. 이 예제에서는 간단함을 위해 **Next.js** 애플리케이션에서 트랜잭션을 메모리에 저장합니다. 이는 데이터베이스, 파일 시스템 또는 서명을 추가해야 하는 수신자에게 전달되는 메시지에 포함 될 수도 있습니다.

이전 섹션의 **NFT** 채굴 스크립트를 사용하여 트랜잭션에 서명을 포함해야 하는 서명자 **2**명을 추가하기만 하면 됩니다. 브라우저에서 **plutus** 스크립트를 동적으로 컴파일 할 수 있으므로 스크립트를 빌드하는 동안 서명자를 스크립트에 추가 할 수 있습니다.

코드 변경 사항

다음은 업데이트된 오프체인 코드와 온체인 코드, 그리고 NFT 민팅 스크립트입니다.

```
const mintNFT = async (params : any) => {

    const address = params[0];
    const name = params[1];
    const description = params[2];
    const img = params[3];
    const sellerAddr = params[4];

    const buyerPkh = Address.fromBech32(address).pubKeyHash;
    const sellerPkh = Address.fromBech32(sellerAddr).pubKeyHash;
    const minAdaVal = new Value(BigInt(2000000)); // NFT 전송에 필요한 최소 러브레이스 수

    // 지갑의 UTXO를 가져오기
    const walletHelper = new WalletHelper(walletAPI);
    const utxos = await
    walletHelper.pickUtxos(minAdaVal);
    // 거스름돈 주소 가져오기
    const changeAddr = await walletHelper.changeAddress;
    // 담보에 사용되는 UTXO 확인
    const colatUtxo = await walletHelper.pickCollateral();
    // 트랜잭션 빌드 시작
    const tx = new Tx();
    // UTXO를 입력으로 추가합니다.
    tx.addInputs(utxos[0]);

    const mintScript = `minting nft

    const TX_ID: ByteArray = #` + utxos[0][0].txId.hex + `
    const txId: TxId = TxId::new(TX_ID)
    const outputId: TxOutputId = TxOutputId::new(txId, ` + utxos[0][0].utxoIdx + `)
    const BUYER: PubKeyHash = PubKeyHash::new(#` + buyerPkh.hex + `)
    const SELLER: PubKeyHash = PubKeyHash::new(#` + sellerPkh.hex + `)
```

```

func main(ctx: ScriptContext) -> Bool {
    tx: Tx = ctx.tx;
    mph: MintingPolicyHash = ctx.get_current_minting_policy_hash();
    assetclass: AssetClass = AssetClass::new(
        mph,
        "` + name + `".encode_utf8()
    );
    value_minted: Value = tx.minted;

    // Validator logic starts
    (value_minted == Value::new(assetclass, 1)).trace("NFT1: ") &&
    (tx.inputs.any((input: TxInput) -> Bool {
        (input.output_id == outputId).trace("NFT2: ")
    }) &&
    tx.is_signed_by(BUYER).trace("NFT3: ") &&
    tx.is_signed_by(SELLER).trace("NFT4: ")
    )
}

// helios 민팅 스크립트 컴파일
const mintProgram = Program.new(mintScript).compile(optimize);
// 스크립트를 트랜잭션에 증인(witness)으로 추가합니다.
tx.attachScript(mintProgram);
// 출력으로 전송할 NFT를 구성합니다.
const nftTokenName = ByteArrayData.fromString(name).toHex();
const tokens: [number[], bigint][] = [[hexToBytes(nftTokenName), BigInt(1)]];

// 실제로 사용하지 않더라도 항상 리더머를 보내야 하므로 빈 리더머를 생성합니다.
// plutus 스크립트 트랜잭션과 함께 리더머를 보내야 하므로 빈 리더머를 생성합니다.
const mintRedeemer = new ConstrData(0, []);
// 이 트랜잭션의 일부로 포함할 민팅을 표시합니다.
tx.mintTokens(
    mintProgram.mintingPolicyHash,
    tokens,
    mintRedeemer
)

```

```

// 출력을 구성하고 최소 Ada와 민팅된 NFT를 모두 포함합니다.
tx.addOutput(new TxOutput(
    Address.fromBech32(address),
    new Value(minAdaVal.lovelace, new Assets([[mintProgram.mintingPolicyHash,
tokens]])))
));
// 담보 utxo 추가
tx.addCollateral(colatUtxo);
// TX에 필요한 구매자 및 판매자 PKH(s) 추가하기
tx.addSigner(buyerPkh);
tx.addSigner(sellerPkh);

const networkParams = new NetworkParams(
    await fetch(networkParamsUrl)
        .then(response => response.json())
)
// 민팅 트랜잭션의 메타데이터를 첨부했습니다.
tx.addMetadata(721, {"map": [[mintProgram.mintingPolicyHash.hex, {"map": [[name,
{
    "map": [{"name", name},
        ["description", description],
        ["image", img]
    ]
}
]]}
]]}
]);

console.log("tx before final", tx.dump());
// 구매자에게 거스름돈을 보냅니다.
await tx.finalize(networkParams, changeAddr);
console.log("tx after final", tx.dump());
// 구매자와 판매자가 서명할 수 있도록 트랜잭션을 저장합니다.
setTxBodyBuyer(tx);
}

```

새로운 기능

다음으로, 오프체인 코드에 구매자의 서명 처리와 판매자의 서명 및 제출 처리를 위한 새로운 함수 2개를 추가해야 합니다.

```
const buyerSign = async () => {

  console.log("Verifying buyer signature...");
  const signatures = await walletAPI.signTx(txBodyBuyer);
  txBodyBuyer.addSignatures(signatures);

  console.log("buyerSigned", txBodyBuyer);
  setTxBodySeller(txBodyBuyer);

}

const sellerSignSubmit = async () => {
  console.log("Verifying seller signature...");
  const signatures = await walletAPI.signTx(txBodySeller);
  txBodySeller.addSignatures(signatures);

  console.log("Submitting transaction...");
  const txHash = await walletAPI.submitTx(txBodySeller);

  console.log("txHash", txHash.hex);
  setTx({ txId: txHash.hex });
}
```

Test Drive

첫 번째 화면에서는 판매자 주소가 필요하며, 이 주소는 판매자의 공개 키 해시(PKH)로 변환됩니다. 구매자(목적지) 주소도 구매자의 PKH로 변환됩니다.

Jan 23 09:48

Helios Tx Builder

3000-material-baseball-zwzft4.us1.demeter.run

Helios Tx Builder

Connect to your wallet

☒ Nami

Wallet Balance In Lovelace 14,000,000

Buyer Wallet Address

addr_test1qq5vczw9q0tag5v8k5tr35kzd06ur3grdv7afmt3us5mdq56r869vgx0wnpw5cerwfgs5glfml9069dys57z0lgvq7vhg

NFT Token Name

Mad Dog

NFT Description

Crazy dog that loves treats

NFT Image

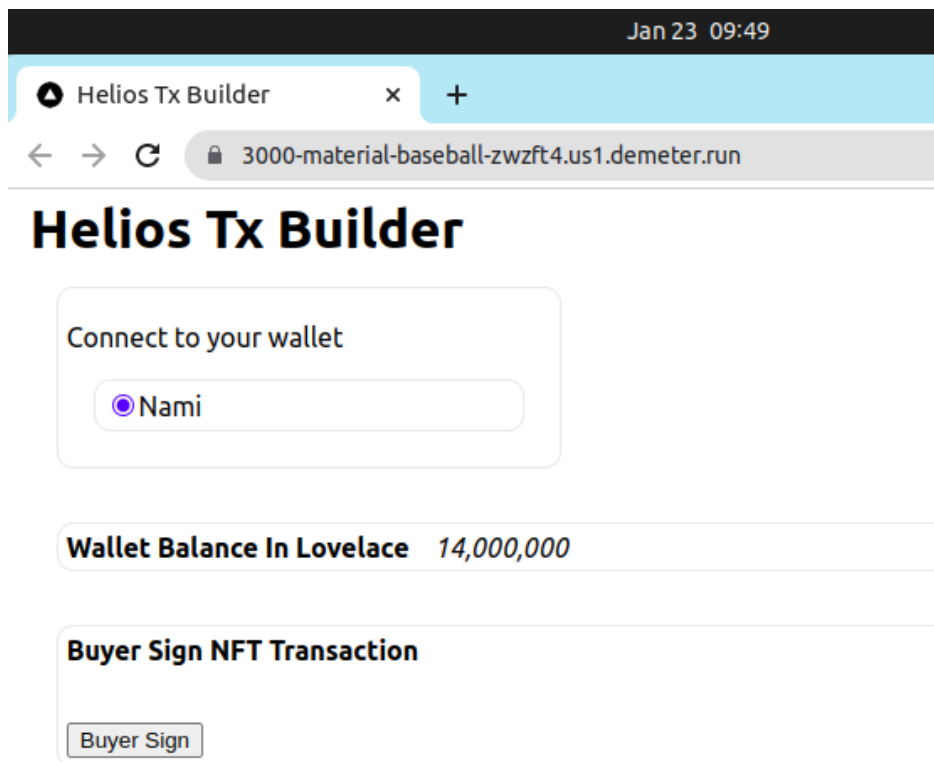
QmckPQPzthJkS9GRSxdw2SzSinAwdDW34xXDZd26MrV5S

Seller Wallet Address

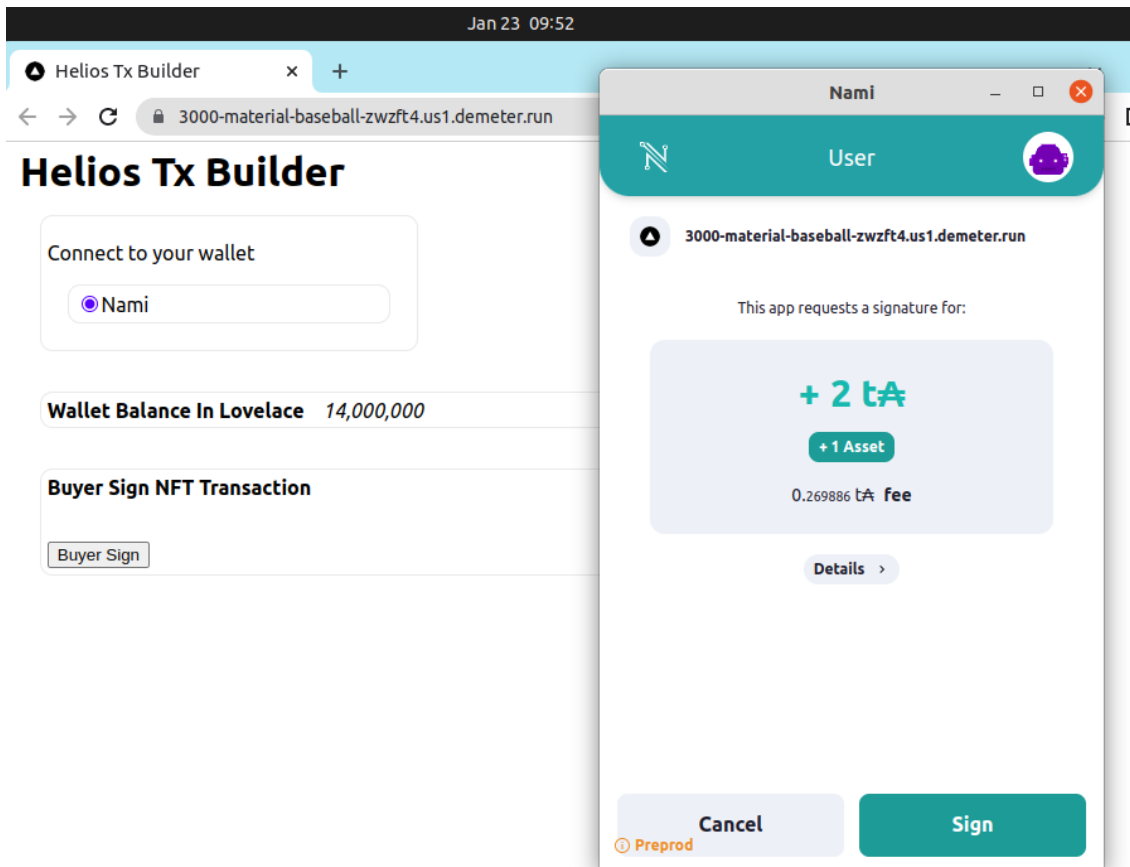
addr_test1qrrkvmepf8lzh6mj4yn237f63tm4tzu7gchahvimqnn2jkevlv9p983fmyphfw2g6nlplq60pv4207wwhqf97tj68s58rzv7

Mint NFT

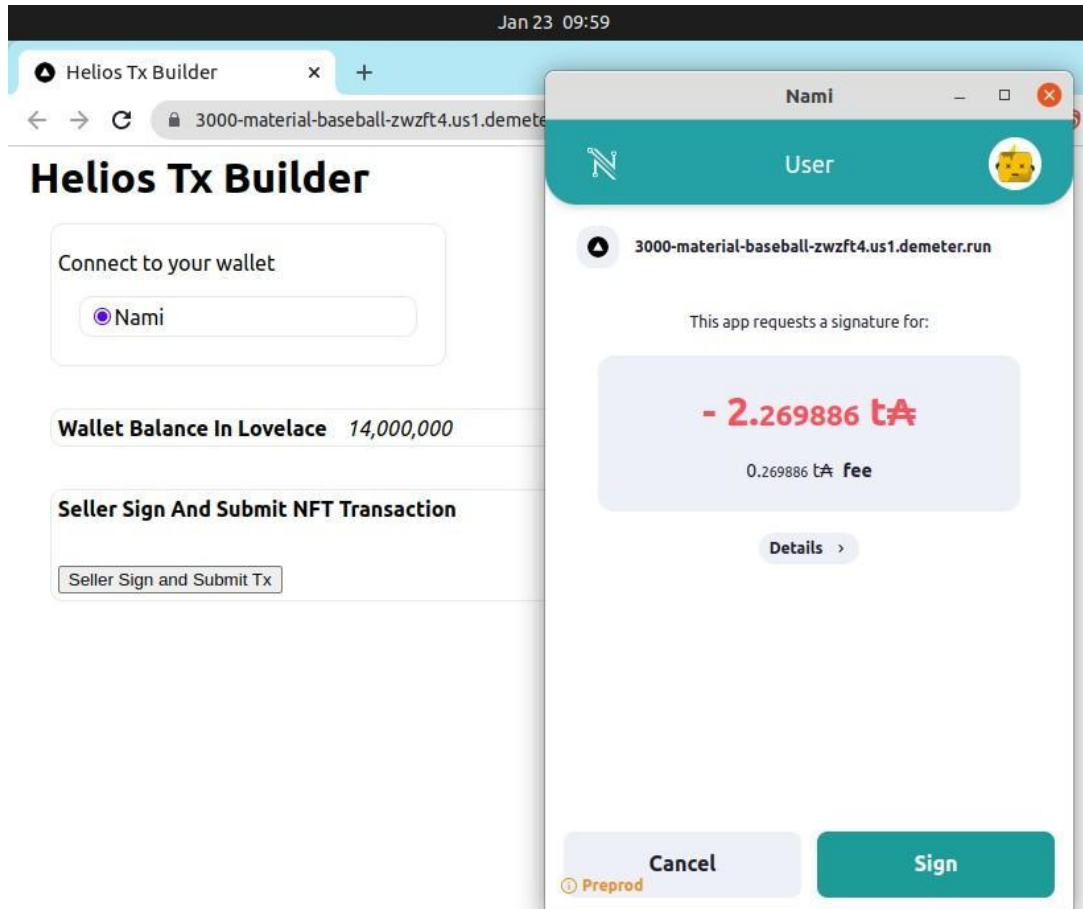
Mint NFT 버튼이 선택되면, Buyer Sign 버튼이 나타납니다. 트랜잭션이 빌드되는 동안, 이 트랜잭션은 Next.js 애플리케이션 내의 상태 변수로 저장됩니다.



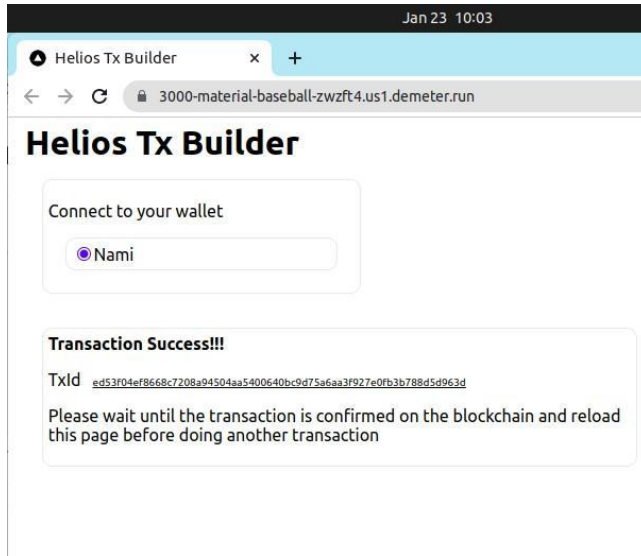
이제 Nami 지갑의 계정을 구매자 계정으로 전환합니다. 이 계정은 구매자 주소를 제공하는 데 사용한 계정과 동일합니다. 이 작업이 완료되면 트랜잭션에 서명하라는 요청이 나타납니다.



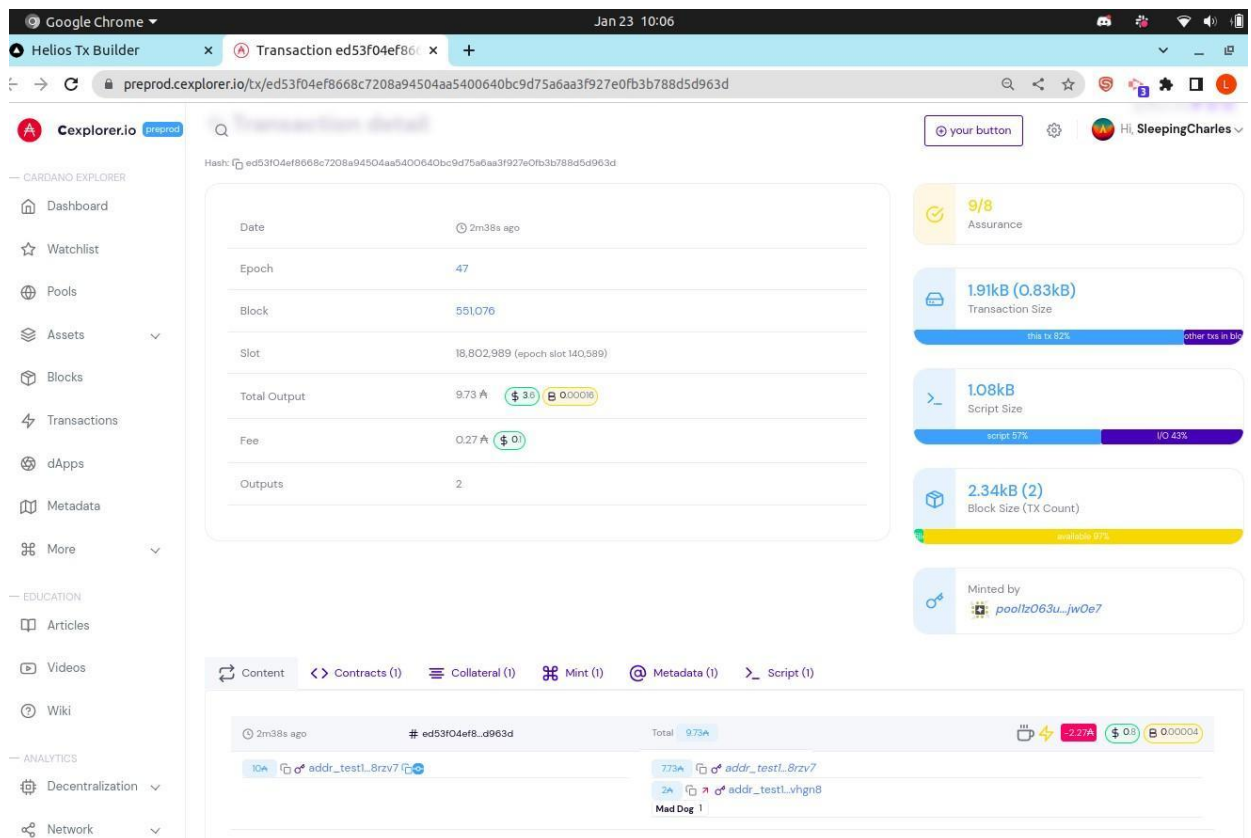
올바른 구매자 계정이 아닌 계정을 선택한 경우, 민팅 컨트랙트에 있는 PKH와 일치하지 않기 때문에 트랜잭션에 서명할 수 없습니다. 올바른 구매자 지갑으로 서명하면 이제 판매자가 서명할 수 있는 버튼이 나타납니다. 판매자 지갑 계정으로 전환하고, **Seller Sign and Submit Tx** 버튼을 선택하세요.



판매자로서 트랜잭션에 서명하면 트랜잭션이 제출되고 NFT가 발행되었다는 것을 확인할 수 있습니다.



cexplorer.io에서 트랜잭션을 확인합니다.



유효성 검사기

검증자는 트랜잭션이 검증자 스크립트 주소에 잠긴 **UTXO**를 사용할 수 있는지 확인하는 데 사용됩니다. 검증자 스마트 컨트랙트 스크립트를 실행할 때 트랜잭션의 일부로 리더머, 데이텀, **UTXO**가 모두 필요합니다.

베스팅

보호 기간이 있는 스마트 컨트랙트는 자산을 특정 기간 동안 잠그고, 기한이 지나면 수익자(**beneficiary**)가 그 자산에 접근할 수 있도록 합니다. **Helios** 문서의 이 예에서는, 기한이 아직 지나지 않은 경우 보호된 토큰을 취소하는 기능도 있습니다.

스마트 컨트랙트 트랜잭션의 핵심 구성 요소인 데이텀(**Datum**)과 리디머(**Redeemer**)를 소개하겠습니다. 데이텀은 온체인에 영구 데이터를 저장하는 데 사용됩니다. 리디머는 스마트 컨트랙트에 어떤 종류의 트랜잭션이 발생하는지 알려줍니다. 이 경우 트랜잭션은 취소(**Cancel**) 또는 청구(**Claim**) 중 하나이며, 그에 따라 검증 로직을 처리합니다.

Datum은 구조체이며, 다음과 같이 표시되며 시간은 **POSIX** 초 형식입니다.

```
struct Datum {  
    creator: PubKeyHash  
    beneficiary: PubKeyHash  
    deadline: Time  
}
```

리디머는 열거형(**enum**) 데이터타입이며 다음과 같습니다.

```
enum Redeemer {  
    Cancel  
    Claim  
}
```

전체 유효성 검사기 **Helios** 코드는 다음과 같습니다. 데이텀(**Datum**)은 리디머(**Redeemer**) 및 스크립트 컨텍스트(**ScriptContext**)와 함께 메인 함수에 전달됩니다. **tx.time_range.start**를 사용하여 이 트랜잭션의 시간 범위를 가져와 현재 시간을 얻습니다. 이 트랜잭션의 시간 범위를 설정하는 방법은 이 섹션의 뒷부분에서 오프체인 코드에서 살펴보겠습니다.

spending vesting

```
struct Datum {
    creator: PubKeyHash
    beneficiary: PubKeyHash
    deadline: Time
}

enum Redeemer {
    Cancel
    Claim
}

func main(datum: Datum, redeemer: Redeemer, context: ScriptContext) -> Bool {
    tx: Tx = context.tx;
    now: Time = tx.time_range.start;
    redeemer.switch {
        Cancel => {
            // Check if deadline hasn't passed
            (now < datum.deadline).trace("VS1: ") &&

            // Check that the owner signed the transaction
            tx.is_signed_by(datum.creator).trace("VS2: ")
        },
        Claim => {
            // Check if the deadline has passed.
            (now > datum.deadline).trace("VS3: ")
            &&

            // Check that the beneficiary signed the transaction.
            tx.is_signed_by(datum.beneficiary).trace("VS4: ")
        }
    }
}
```

더 복잡한 스마트 컨트랙트를 다룰 때는 별도의 파일로 직접 작업하고 미리 컴파일하는 것이 더 쉽습니다. 또한 **Helios**용 **VS Code** 확장 프로그램도 사용할 수 있습니다.

또한, 스마트 컨트랙트를 컴파일하여 생성된 **plutus** 코드를 얻기 위해 **deno**를 사용하는 것을 추천합니다.

deno를 설정하려면 **VS Code** 웹 터미널 창에서 다음을 수행하세요.

```
$ cd ~/workspace/repo
```

간단한 웰컴 타입스크립트 프로그램을 테스트하는 데노를 설치합니다.

```
$ npx deno-bin run https://deno.land/std/examples/welcome.ts
```

다음으로, 베스팅 디렉토리로 이동하여 다음 **deno** 명령을 실행하여 **Helios** 코드를 컴파일합니다.

```
$ cd vesting
```

```
$ npx deno-bin run --allow-read --allow-write ./src/deploy-vesting.js
```

컴파일러는 구문 또는 타입 관련 오류가 있는 경우 오류를 표시합니다. 따라서 나중에 **Next.js** 애플리케이션에서 코드를 실행할 때가 아니라 지금 바로 수정할 수 있습니다. 배포 디렉터리에서 결과 파일을 확인할 수도 있습니다.

```
$ ls -l deploy/
```

```
total 12
```

```
-rw-rw-r-- 1 lawrence lawrence 63 Jan 19 10:58 vesting.addr
```

```
-rw-rw-r-- 1 lawrence lawrence 56 Jan 19 10:58 vesting.hash
```

```
-rw-rw-r-- 1 lawrence lawrence 788 Jan 19 10:58 vesting.plutus
```

이 예제에서 생성된 파일들이 필요하지는 않지만, 미리 생성된 **plutus** 스크립트, 검증자 해시 및/또는 검증자 주소가 필요한 경우도 있을 수 있습니다.

Helios 소스 코드가 오류 없이 컴파일되는 것을 확인한 후, **contracts** 디렉토리로 복사하겠습니다.

```
$ cp src/vesting.hl contracts/
```


Next.js 설정

이제 이 예제에서 필요한 몇 가지 환경 변수를 설정해야 합니다. 애플리케이션 코드 외부에서 환경 변수를 관리하는 것이 항상 모범 사례입니다.

1. Using VS Code Web, select File -> Open
2. In the popup window enter the following path and filename /config/.bashrc
3. Authorize VS Code Web access if requested
4. Add the following lines to the bottom of the .bashrc file

```
export NEXT_PUBLIC_BLOCKFROST_API_KEY="get-your-blockfrost-api-key"  
export NEXT_PUBLIC_BLOCKFROST_API="https://cardano-preprod.blockfrost.io/api/v0"  
export NEXT_PUBLIC_NETWORK_PARAMS_URL="https://d1t0d7c2nekuk0.cloudfront.net/preprod.json"
```

5. Now, in the Terminal window type the following command to load these environment variables into your shell. These environment variables will automatically load next time you log in as well.
6. `$ source ~/.bashrc`

Note: 4단계에서 [Blockfrost](#) 계정을 설정하고 API 키를 받으려면 직접 계정을 만들어야 합니다.

Next.js 시작

우리는 이제 **Next.js** 애플리케이션을 초기화하고 시작할 준비가 되었습니다.

```
$ npm install  
$ npm run dev
```

Demeter Run 워크스페이스 **Exposed Port** 탭에서 포트 3000을 노출시켰는지 확인하세요. 익스포트 포트 URL 링크를 선택하여 애플리케이션을 시작하세요. 지갑을 선택하면 다음과 같이 표시됩니다.

← → ↻ 🔒 3000-troubled-sympathy-5oh0q3.us1.demeter.run

Helios Tx Builder

Connect to your wallet

☒ Nami


View Smart Contract: [vesting.ht](#)

Wallet Balance In Lovelace 511,331,294

Beneficiary Wallet Address

Amount Of Ada To Lock

Vesting Expiry Date



Claim Funds

Cancel Vesting

스마트 컨트랙트 뷰어

Helios Tx 트랜잭션 빌더 생성 중에 사용되는 파일 시스템의 동일한 파일을 읽는 **API**로 연결되는 **View Smart Contract** 링크를 선택하여 애플리케이션 내에서 스마트 컨트랙트를 확인할 수 있습니다.

```
← → ↻ 3000-troubled-sympathy-5oh0q3.us1.demeter.run/api/vesting

spending vesting

struct Datum {
  creator: PubKeyHash
  beneficiary: PubKeyHash
  deadline: Time
}

enum Redeemer {
  Cancel
  Claim
}

func main(datum: Datum, redeemer: Redeemer, context: ScriptContext) -> Bool {
  tx: Tx = context.tx;
  now: Time = tx.time_range.start;

  redeemer.switch {
    Cancel => {
      // Check if deadline hasn't passed
      (now < datum.deadline).trace("VS1: ") &&

      // Check that the owner signed the transaction
      tx.is_signed_by(datum.creator).trace("VS2: ")
    },
    Claim => {
      // Check if deadline has passed.
      (now > datum.deadline).trace("VS3: ") &&

      // Check that the beneficiary signed the transaction.
      tx.is_signed_by(datum.beneficiary).trace("VS4: ")
    }
  }
}
```

Ada 잠금

이제 오픈 체인 **Helios TX** 빌더 코드를 살펴보겠습니다. 이 예제를 위해 추가된 핵심적인 새로운 코드 영역 중 일부에 초점을 맞춥니다. **lockAda** 함수는 데이텀을 생성하고, 제출될 값과 소유자 지갑의 **PKH(Payment Key Hash)**로 채웁니다.

```
const lockAda = async (params : any) => {
  const benAddr = params[0] as string;
  const adaQty = params[1] as number;
  const dueDate = params[2] as string;
  const deadline = new Date(dueDate + "T00:00");
  const benPkh = Address.fromBech32(benAddr).pubKeyHash;
  const adaAmountVal = new Value(BigInt((adaQty)*1000000));

  ...
  // 데이텀 생성
  const datum = new ListData([new ByteArrayData(ownerPkh.bytes),
                                new ByteArrayData(benPkh.bytes),
                                new IntData(BigInt(deadline.getTime()))]);

  const inlineDatum = Datum.inline(datum);

  ...
}
```

마지막 변경 사항은 **Ada**와 베스팅 키 토큰을 모두 포함하도록 트랜잭션 출력을 생성하는 것입니다.

```
...
// 출력으로 전송할 NFT를 구성합니다.
const nftTokenName = ByteArrayData.fromString("Vesting Key").toHex();
const tokens: [number[], bigint][] = [[hexToBytes(nftTokenName), BigInt(1)]];

...
const lockedVal = new Value(adaAmountVal.lovelace,
  new Assets([[mintProgram.mintingPolicyHash, tokens]]));

// 잠긴 Ada의 양과 자산, 데이텀, 목적지 주소를 추가한다.
tx.addOutput(new TxOutput(valAddr, lockedVal, inlineDatum));

...
}
```

베스팅 키 발행 정책

베스팅 키 토큰을 만들기 위해 인라인 민팅 정책을 추가합니다. 이는 우리가 고유한 NFT를 발행하는 데 사용한 기술과 동일한 기술을 활용하며, 잠긴 UTXO를 찾는 데 도움이 될 것입니다.

```
...
const mintScript = `minting nft

const TX_ID: ByteArray = #` + utxos[0][0].txId.hex + `
const txId: TxId = TxId::new(TX_ID)
const outputId: TxOutputId =
    TxOutputId::new(txId, ` + utxos[0][0].utxoIdx + `)

func main(ctx: ScriptContext) -> Bool {
    tx: Tx = ctx.tx;
    mph: MintingPolicyHash = ctx.get_current_minting_policy_hash();

    assetclass: AssetClass = AssetClass::new(
        mph,
        "Vesting Key".encode_utf8()
    );
    value_minted: Value = tx.minted;

    // Validator logic starts
    (value_minted == Value::new(assetclass, 1)).trace("NFT1: ") &&
    tx.inputs.any((input: TxInput) -> Bool {
        (input.output_id == outputId).trace("NFT2: ")
    })
} `
...

```

자금 청구 또는 베스팅 취소

베스팅 컨트랙트에서 청구하거나 취소하려면 사용자는 베스팅 키 토큰을 제공해야 합니다. 이 키를 사용하여 스마트 컨트랙트 스크립트 주소에서 잠긴 올바른 **UTXO**를 찾습니다. `getKeyUtxo` 함수는 조회를 수행하고 **Helio UTXO** 객체를 반환합니다.

```
// 스크립트 주소에서 베스팅 키 토큰이 포함된 utxo를 받습니다.
const getKeyUtxo = async (scriptAddress : string,
                           keyMPH : string, keyName : string ) => {

  const blockfrostUrl : string = blockfrostAPI + "/addresses/" +
    scriptAddress + "/utxos/" + keyMPH + keyName;

  let resp = await fetch(blockfrostUrl, {
    method: "GET",
    headers: {
      accept: "application/json",
      project_id: apiKey,
    },
  });

  if (resp?.status > 299) {
    throw console.error("vesting key token not found", resp);
  }

  const payload = await resp.json();

  if (payload.length == 0) {
    throw console.error("vesting key token not found");
  }

  const lovelaceAmount = payload[0].amount[0].quantity;
  const mph = MintingPolicyHash.fromHex(keyMPH);
  const tokenName = hexToBytes(keyName);

  const value = new Value(BigInt(lovelaceAmount),
    new Assets([[mph,
      [[tokenName, BigInt(1)]]]]
  ));
};
```

```
return new UTxO(  
    TxId.fromHex(payload[0].tx_hash),  
    BigInt(payload[0].output_index),  
    new TxOutput(  
        Address.fromBech32(scriptAddress),  
        value,  
        Datum.inline(ListData.fromCbor(hexToBytes(payload[0].inline_datum)))  
    )  
);  
}
```

`claimFunds` 또는 `cancelVesting` 함수 모두 매우 유사합니다. 리디머를 만들고, 스크립트 주소에서 잠긴 **UTXO**를 가져와 기한이 지났는지 확인하는 코드는 거의 동일합니다.

```
...
// 스크립트 주소에서 잠긴 UTXO를 소비하기 위해 claim 리디머를 생성하세요.
const valRedeemer = new ConstrData(1, []);

// 베스팅 키 토큰이 들어있는 UTXO를 받습니다.
const valUtxo = await getKeyUtxo(valAddr.toBech32(), keyMPH,
    ByteArrayData.fromString("Vesting Key").toHex());

// 스크립트 주소에서 UTXO를 사용하려면 리디머를 포함해야 합니다.
tx.addInput(valUtxo, valRedeemer);

// valUTXO의 자산을 수취인에게 보내세요.
tx.addOutput(new TxOutput(claimAddress, valUtxo.value));

// 이 트랜잭션이 유효한 시간을 지정합니다.
// 이는 스크립트에서 사용될 트랜잭션에 시간이 포함되도록 필요합니다.
// 수명이 두 시간인 시간을 추가하고 현재 시간을 5분 미룹니다.

const currentTime = new Date().getTime();
const earlierTime = new Date(currentTime - 5 * 60 * 1000);
const laterTime = new Date(currentTime + 2 * 60 * 60 * 1000);

tx.validFrom(earlierTime);
tx.validTo(laterTime);
...
```


Test Drive

소유자가 스마트 콘트랙트에서 자금을 잠그기 위해 트랜잭션을 제출하면, 나중에 자금을 잠금 해제하는 데 사용되는 베스팅 키가 표시됩니다.

← → ↻ 3000-troubled-sympathy-5oh0q3.us1.demeter.run

Helios Tx Builder

Connect to your wallet

☒ Nami

View Smart Contract: [vesting.hl](#)

Transaction Success!!!

TxId [36ae88746977376c8f12d34a96a195d7d0d06929b4533e3eaa134b8259bf89c5](#)

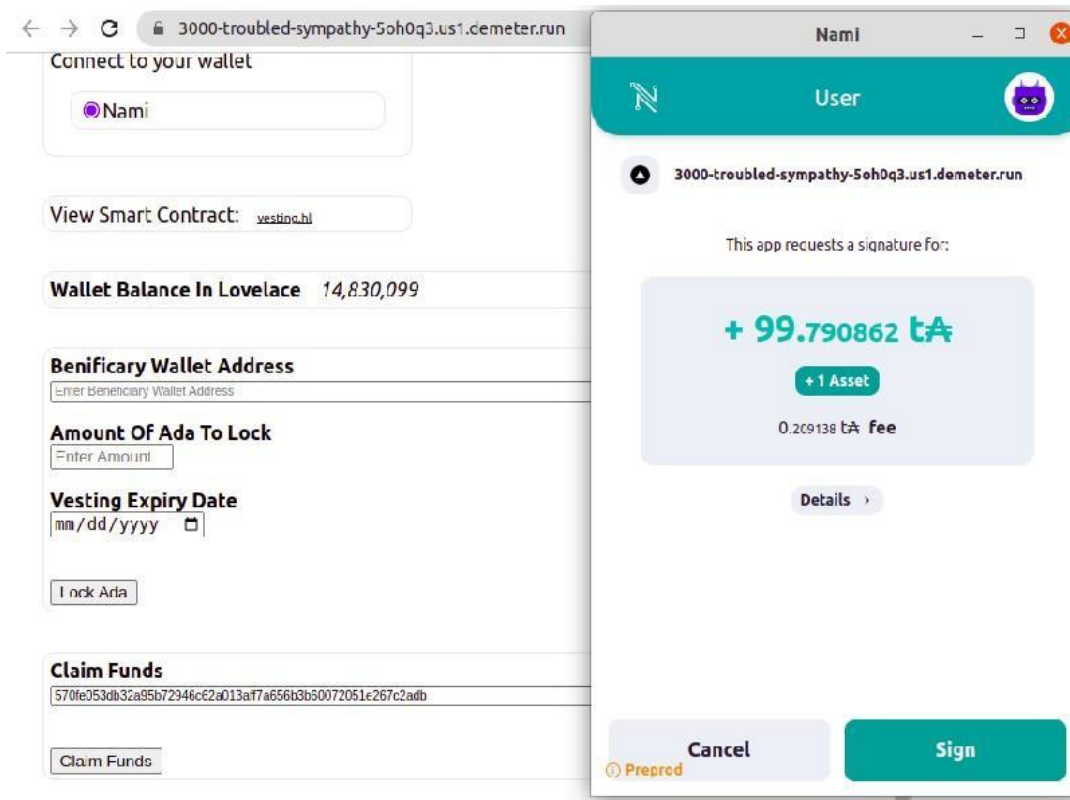
Please wait until the transaction is confirmed on the blockchain and reload this page before doing another transaction

Please copy and save your vesting key

570fe053db32a95b72946c62a013aff7a656b3b60072051e267c2adb

You will need this key to unlock your funds

테스팅 키를 입력하고 자금 청구를 선택한 후 수혜자는 트랜잭션에 서명하고 제출해야 합니다. 이 예시에서 볼 수 있듯이, 수혜자는 **99.790862 tAda**를 받게 됩니다.



Preprod 익스플로러에서 Ada가 잠긴 토큰과 베스팅 키 토큰이 모두 수혜자에게 전송되었습니다. 스마트 계약은 과거에 기한이 설정되어 있고 사용된 수혜자 지갑에 올바른 PKH가 있었기 때문에 이 트랜잭션이 성공할 수 있었습니다.

The screenshot shows the Preprod Explorer interface for a transaction. The URL bar displays the transaction ID: `preprod.cexplorer.io/tx/f527d6db27bbcd9ac8bcf4d4b4f37f5764d20ed278355e2fb8af9ac8c193dd9`. The left sidebar contains navigation links: Dashboard, Watchlist, Pools, and Assets. The main content area shows the transaction details for `# f527d6db27...93dd9`, which occurred 1m54s ago. The total value is 109.62 ADA. The transaction includes two outputs: 100 ADA to `addr_test1...qekxy` and 9.62 ADA to `addr_test1...zcsuy`. The vesting key for the first output is `-1` and for the second output is `1`.

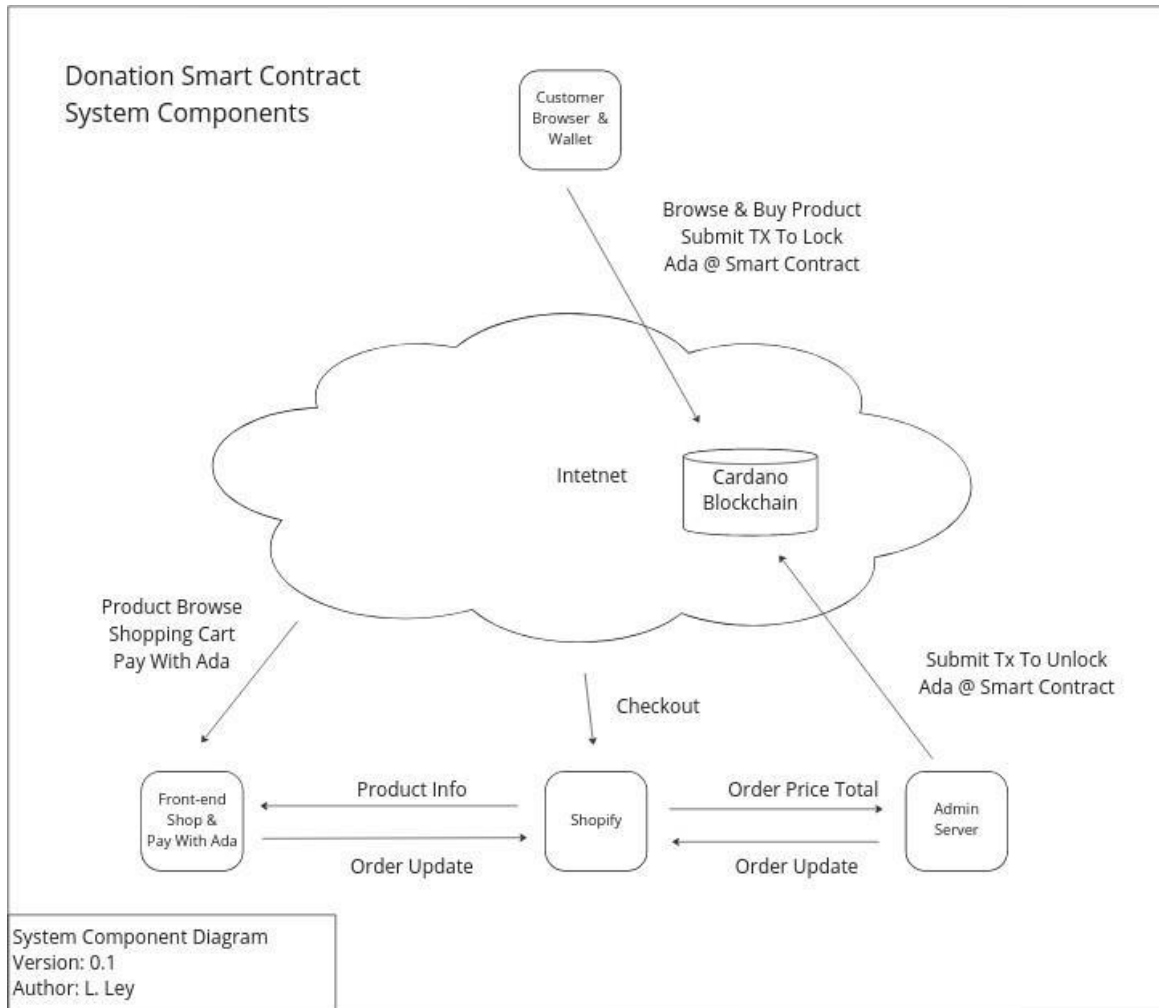
Output	Value	Address	Vesting Key
1	100 ADA	<code>addr_test1...qekxy</code>	<code>-1</code>
2	9.62 ADA	<code>addr_test1...zcsuy</code>	<code>1</code>

기부금 추적 (Ada 잠금)

이번 최종 예제에서는 기부금 추적성 스마트 컨트랙트를 검토합니다. 이는 전자 상거래 주문에서 기부되는 금액의 비율에 대한 투명성과 감사성을 제공합니다.

시스템 구성 요소

다음 다이어그램은 스마트 컨트랙트와 상호작용하는 시스템과 관련된 시스템을 보여줍니다.



이 아키텍처에 관련된 주요 시스템은 다음과 같습니다:

- 고객 브라우저
- **Next.js** 프론트엔드 애플리케이션
- **Shopify**
- 배치 처리를 위한 관리자 서버
- **Cardano** 블록체인

사용자가 **Ada**로 제품을 구매하면 주문 금액이 스마트 컨트랙트에 잠깁니다. 스마트 컨트랙트는 일정 비율 할당과 **Ada**를 보낼 수 있는 지갑 주소에 대한 매우 구체적인 규칙을 갖고 있습니다. 이는 모든 사람이 상인과 자선단체에 대한 **Ada** 분배를 볼 수 있도록 기부금 추적성을 제공합니다.

스마트 컨트랙트 코드

기부금 검증을 위한 스마트 컨트랙트의 코드는 다음과 같습니다. 지갑 주소와 기부금 할당은 컨트랙트 매개변수로 하드코딩되어 있습니다. 이는 스마트 컨트랙트가 컴파일되고 체인에 로드된 후에는 변경할 수 없다는 것을 의미합니다. 또한 보안을 강화하기 위해 관리자만이 스크립트를 실행할 수 있도록 스크립트를 보호하고자 합니다. 하지만 이는 선택적인 설계 선택 사항입니다.

spending vesting

```
struct Datum {
    orderAmount: Int
    orderId: ByteArray
    adaUsdPrce: ByteArray
}

enum Redeemer {
    Spend
    Refund
}

// 판매자의 PKH 정의
const MERCHANT_PKH: ByteArray = #3d6...38c
const merchantPkh: PubKeyHash = PubKeyHash::new(MERCHANT_PKH)

// 기부금의 PKH 정의
const DONOR_PKH: ByteArray = #b2b...7a9f
const donorPkh: PubKeyHash = PubKeyHash::new(DONOR_PKH)

// 환불의 PKH 정의
const REFUND_PKH: ByteArray = #a0a...9a9
const refundPkh: PubKeyHash = PubKeyHash::new(REFUND_PKH)

// 관리자의 PKH 정의
const ADMIN_PKH: ByteArray = #b9a...7682
const adminPkh: PubKeyHash = PubKeyHash::new(ADMIN_PKH)

const SPLIT: Int = 90 // 분할 판매자와 기부자 정의
const minAda = 1000000 // 기부를 위한 최소 Ada 정의
const version = 2 // 필요한 경우 컨트랙트의 버전 번호 늘리기
```

```

func getDonationAmt (orderAmt: Int) -> Int {
    donationAmt: Int = (orderAmt * (100 - SPLIT)) / 100;
    if (donationAmt < minAda) {
        minAda
    } else {
        donationAmt
    }
}

func main(datum: Datum, redeemer: Redeemer, context: ScriptContext) -> Bool {
    tx: Tx = context.tx;
    redeemer.switch {
        Spend => {

            orderAmt: Int =
            datum.orderAmount;
            donationAmt: Int =
            getDonationAmt (orde
            rAmt); merchantAmt:
            Int = orderAmt -
            donationAmt;
            donationAmtVal: Value =
            Value::lovelace(donationAmt);
            merchantAmtVal: Value =
            Value::lovelace(merchantAmt);
            merchOutTxS : []TxOutput =
            tx.outputs_sent_to(merchantPkh);
            donorOutTxS : []TxOutput =
            tx.outputs_sent_to(donorPkh);
            tx.is_signed_by(adminPkh).trace("DN1:
            ") && (merchOutTxS.head.value ==
            merchantAmtVal).trace("DN2: ") &&
            (donorOutTxS.head.value ==
            donationAmtVal).trace("DN3: ")

            },
        Refund => {

            orderAmt: Int = datum.orderAmount;
            returnAmtVal: Value =
            Value::lovelace(orderAmt);
            refundOutTxS : []TxOutput =
            tx.outputs_sent_to(refundPkh);
            tx.is_signed_by(adminPkh).trace("
            DN4: ") &&
            (refundOutTxS.head.value ==
            returnAmtVal).trace("DN5: ")

        }
    }
}

```



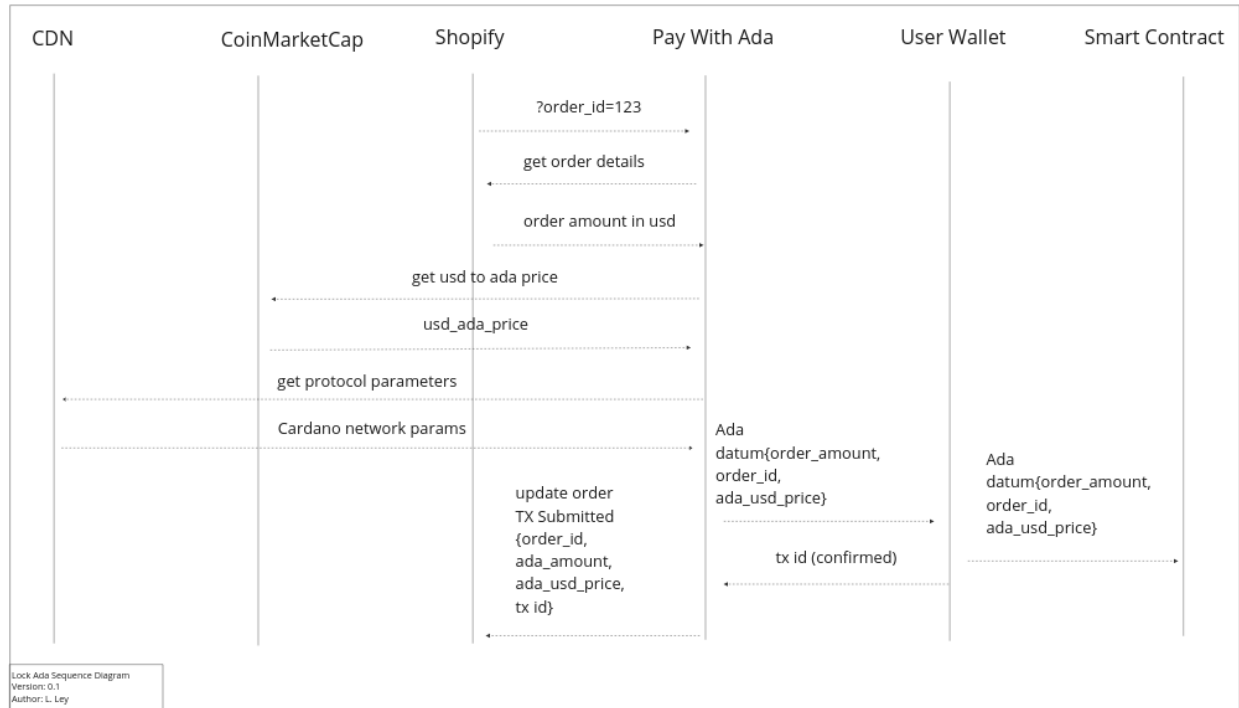
```
}  
}
```

상인, 기부자, 환불 및 관리자 **PKH**에 대해 하드코딩된 컨트랙트 매개변수를 사용할 것입니다. 이를 통해 주문 금액의 정확한 비율이 실제로 판매자, 기부자 또는 환불 지갑에 잠겨 있는지 확인할 수 있습니다. 또한 보안을 강화하기 위해 관리자만이 스크립트를 실행할 수 있도록 스크립트를 보호하고자 합니다. 하지만 이는 선택적인 설계 선택 사항입니다.

Ada 결제

다음 시퀀스 다이어그램은 **Ada**로 결제 구성 요소의 로직을 보여줍니다. 이는 고객이 이커머스 주문을 완료할 때 **Ada**로 결제할 수 있도록 하기 위해 필요합니다.

Locking Ada At The Smart Contract



Ada 결제 구성 요소는 브라우저 **URL**의 쿼리 문자열 매개 변수를 통해 **Shopify**에서 주문 **ID**를 가져옵니다. 그러면 **Next.js** 앱은 주문 금액을 가져오기 위해 **Shopify** 관리자 **API** 조회를 수행합니다. 주문 금액이 확인되면 다시 한 번 조회를 수행하여 **ADA/USD** 변환을 가져와 필요한 **ADA**의 양을 파악합니다. 마지막으로 사용자가 **Ada** 결제 버튼을 선택하면 사용자가 서명하고 제출할 수 있는 데이터로 지출 거래가 구성됩니다.

컴파일 & 배포

VS Code 웹에서 터미널 창을 엽니다.

```
$ cd donation
```

상인, 기부자, 환불 및 관리자 **PKH**와 같은 매개변수에 대한 변경 사항이 있으면 **src/donation.hl** 스마트 컨트랙트를 업데이트하세요. 변경 사항이 완료되면 **donation** 디렉토리에서 **Helios** 코드를 컴파일하세요.

deno가 설치되지 않은 경우 다음을 실행합니다.

```
$ npx deno-bin run https://deno.land/std/examples/welcome.ts
```

이제 다음 명령을 실행하여 **Helios** 스마트 컨트랙트를 **plutus** 스크립트로 컴파일합니다.

```
$ npx deno-bin run --allow-read --allow-write ./src/deploy-donation.js
```

그런 다음 생성된 파일을 스크립트 **data** 디렉토리와 **Next.js contracts** 디렉토리 각각에 복사하세요.

```
$ cp deploy/* scripts/cardano-cli/preprod/data
$ cp src/donation.hl contracts
```

다음으로 **.bashrc** 파일을 편집하여 다음 환경 변수를 추가/업데이트해야 합니다. 이를 위해서는 **VS Code Web** -> **Open File**로 이동하여 팝업 창에 **/config/.bashrc**를 붙여넣으면 됩니다.

참고: 필요한 **API** 키를 가져오기 위해 [Shopify](#) 및 [CoinMarketCap](#)에 가입해야 할 수도 있습니다.

```
export NEXT_PUBLIC_NETWORK="preprod"
export NEXT_PUBLIC_BLOCKFROST_API_KEY="blockfrost api key"
export NEXT_PUBLIC_BLOCKFROST_API="https://cardano-preprod.blockfrost.io/api/v0"
export NEXT_PUBLIC_NETWORK_PARAMS_URL="https://d1t0d7c2nekuk0.cloudfront.net/preprod.json"
export NEXT_PUBLIC_MIN_ADA=2000000
export NEXT_PUBLIC_SHOP="https://shopify-store-url" export
NEXT_PUBLIC_ACCESS_TOKEN="shopify access token" export
NEXT_PUBLIC_COIN_API_KEY="coin market cap api key" export
NEXT_PUBLIC_SERVICE_FEE=500000
export NEXT_PUBLIC_ORDER_API_KEY="create a unique base64 hash key"
```

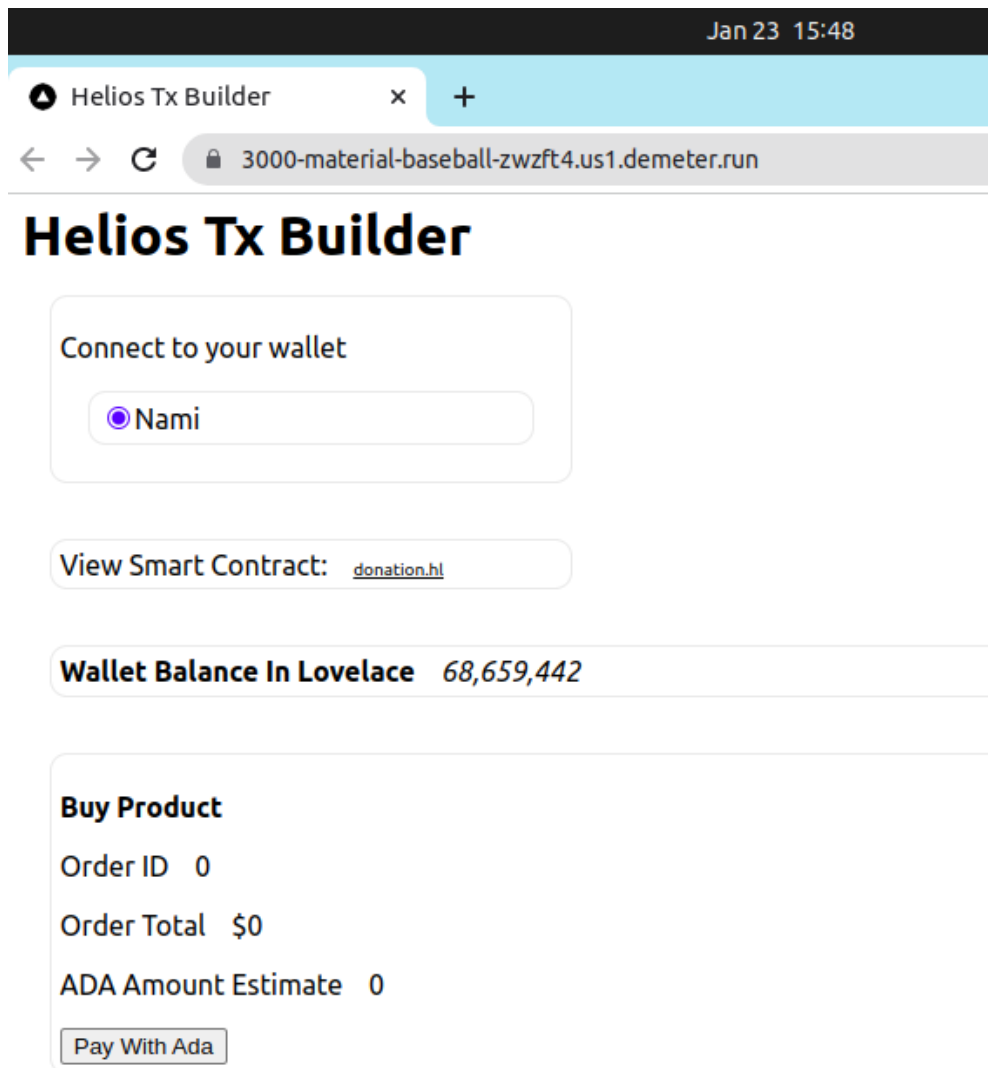
환경 변수를 현재 셸에 로드합니다. **.bashrc** 파일에 있는 경우 로그인할 때마다 자동으로 로드됩니다.

Next.js 설정

이제 **npm** 모듈을 설치하고 **Next.js**를 시작하세요. **Demeter Run workspace**의 노출된 포트 탭에서 포트 **3000**을 노출하는 것을 잊지 마세요.

```
$ source ~/.bashrc
$ npm install
$ npm run dev
```

쿼리 스트링에 주문 ID가 전달되지 않은 경우 다음과 같은 내용이 표시되어야 합니다. 다음 단계에서 필요한 **Next.js** 앱 웹 URL을 복사합니다.



Shopify 설정

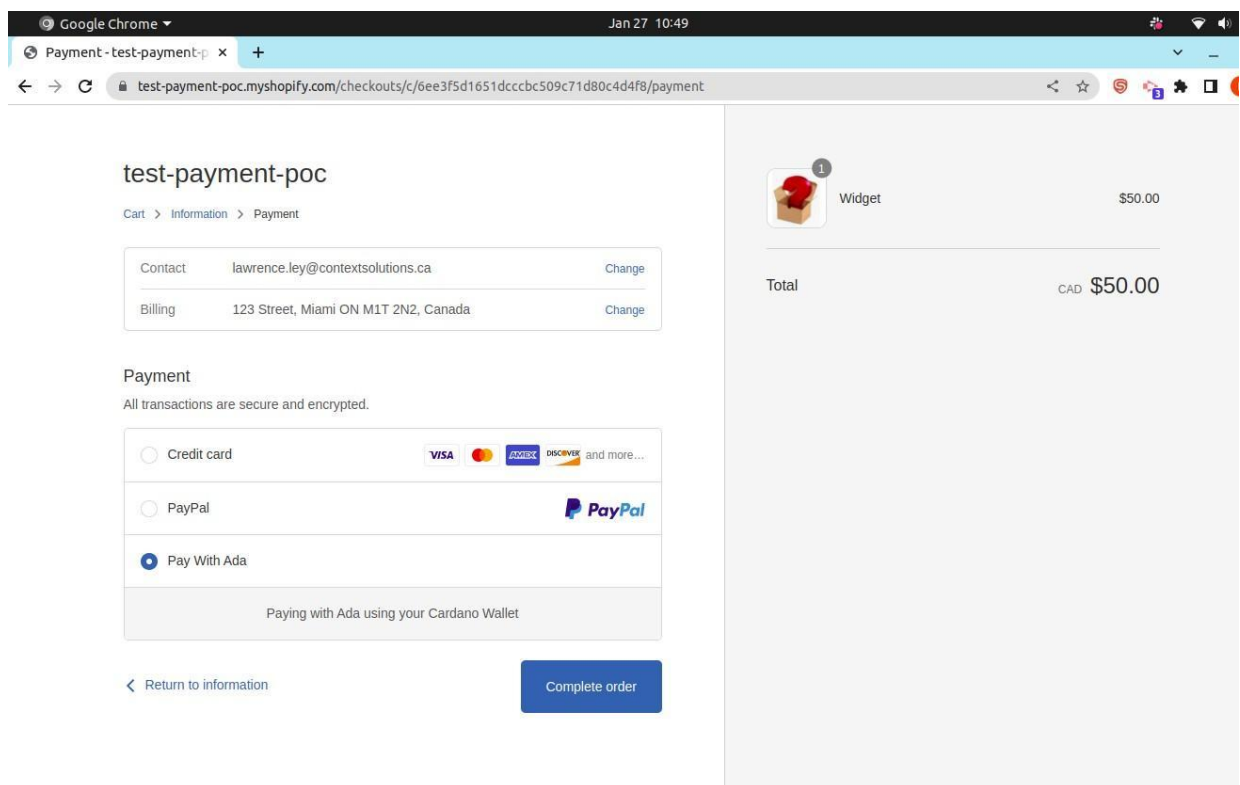
Ada를 지불하는 데 사용되는 URL을 업데이트해야 합니다. 만약 Ada 결제를 위한 Shopify 스토어 구성에 도움이 필요하다면 부록의 "Shopify Ada Payments"를 참조하세요.

1. Go to your shopify store admin section
2. Select Settings
3. Select Checkout
4. Scroll to the bottom and update the variable in the Additional Scripts dialog box with the URL you copied in the last step. For example:

```
var urlStr = "https://3000-material-baseball-zwzft4.us1.demeter.run/";
```
5. Select Save and close the Settings Window

Test Drive

이제 Shopify 스토어로 이동하여 제품 결제를 진행하고 결제 옵션으로 “Pay With Ada”를 선택합니다.



주문 완료를 선택하면 주문 상태 페이지로 이동합니다.

Google Chrome Jan 27 10:50

Thank you for your purchase

test-payment-poc.myshopify.com/checkouts/c/51399cc5f7a9dba3ce62879b4e741b2c/thank_you

test-payment-poc

5247401754903
Thank you!

Your order is confirmed
Please select to the Pay Now With Ada link below to pay using your Cardano Wallet

☐ Email me with news and offers

Pay To Complete Your Order


[Pay Now In Ada](#)

Customer Information

Contact information lawrence.ley@contextsolutions.ca	Payment method Pay With Ada - \$50.00
	Billing address Test 123 Street Miami ON M1T 2N2 Canada

Need help? [Contact us](#)

[Continue shopping](#)

 **Widget** \$50.00

Total CAD **\$50.00**

지금 **Play With Ada** 링크를 선택하면 **Next.js** 웹 애플리케이션으로 이동합니다. URL에는 애플리케이션이 주문 달러 금액을 검색할 수 있는 쿼리 스트링과 결제해야 하는 **Ada** 금액에 대한 **ADA/USD** 변환이 포함됩니다.

Google Chrome ▾

Jan 27 10:51

Helios Tx Builder × +

← → ↻ 3000-venomous-decision-kd0ey1.us1.demeter.run/?id=5247401754903

Helios Tx Builder

Connect to your wallet

☒ Nami

View Smart Contract:

[donation.hi](#)

Wallet Balance In Lovelace

361,762,455

Buy Product

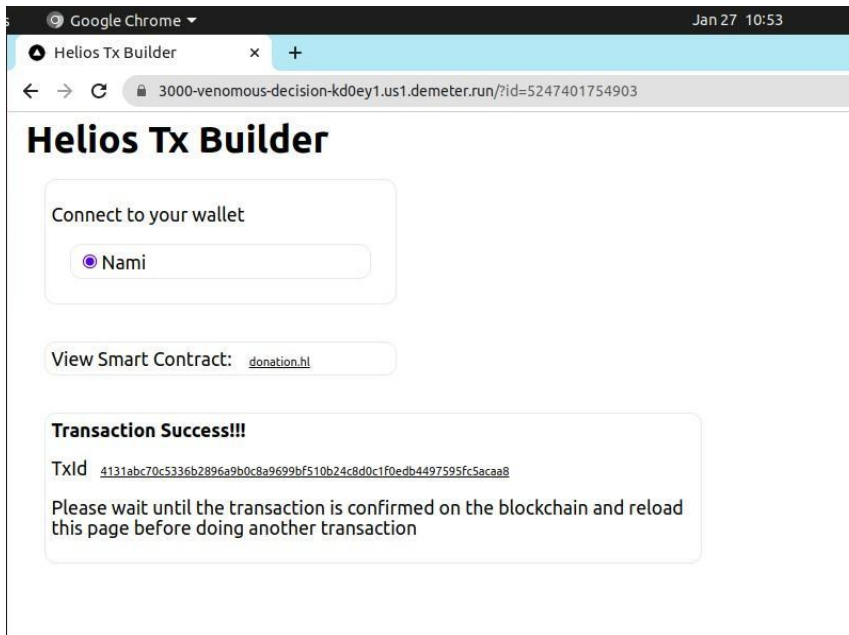
Order ID 5247401754903

Order Total \$50.00

ADA Amount Estimate 132.28

Pay With Ada

Pay With Ada 버튼을 선택한 후, 지갑을 사용하여 트랜잭션에 서명하고 제출하세요. 이제 주문 금액을 지불하고 **Ada**를 기부 스마트 컨트랙트 주소에 잠갔습니다. 또한 담보가 필요하지 않았음을 알 수 있습니다. 우리는 단순히 (데이텀과 함께) **Ada**를 주소에 잠그고 스마트 컨트랙트를 실행하지 않았을 뿐입니다. 이 설계 선택은 이전에 **Cardano dapp**을 사용해 본 적이 없는 사용자들에게 쇼핑 경험에서 마찰을 줄이는 데 도움이 됩니다.



Ada와 데이텀을 잠그는 트랜잭션은 cexplorer.io에서 확인할 수 있습니다.

The screenshot shows the Cexplorer.io website in a Google Chrome browser. The page displays the details of a transaction with the hash 4131abc70c5336b2896a9b0c8a9699bf510b24c8d0c1f0edb4497595fc5acaa8. The transaction is titled "Transaction detail" and includes a table of metadata and a list of outputs.

Transaction detail

Hash: 4131abc70c5336b2896a9b0c8a9699bf510b24c8d0c1f0edb4497595fc5acaa8

Field	Value
Date	41s ago
Epoch	48
Block	564,494
Slot	19,151,618 (epoch slot 57,218)
Total Output	361.59 ADA (\$ 137.7) 0.00598 B
Fee	0.17 ADA (\$ 0.1)
Outputs	2

Transaction Size

0.29kB (0.29kB)

Transaction Size

this tx 99%

Block Size (TX Count)

0.29kB (1)

Block Size (TX Count)

available 132B

Minted by

pool1z063u...jw0e7

Content

41s ago # 4131abc70c...acaa8 Total 361.59 ADA

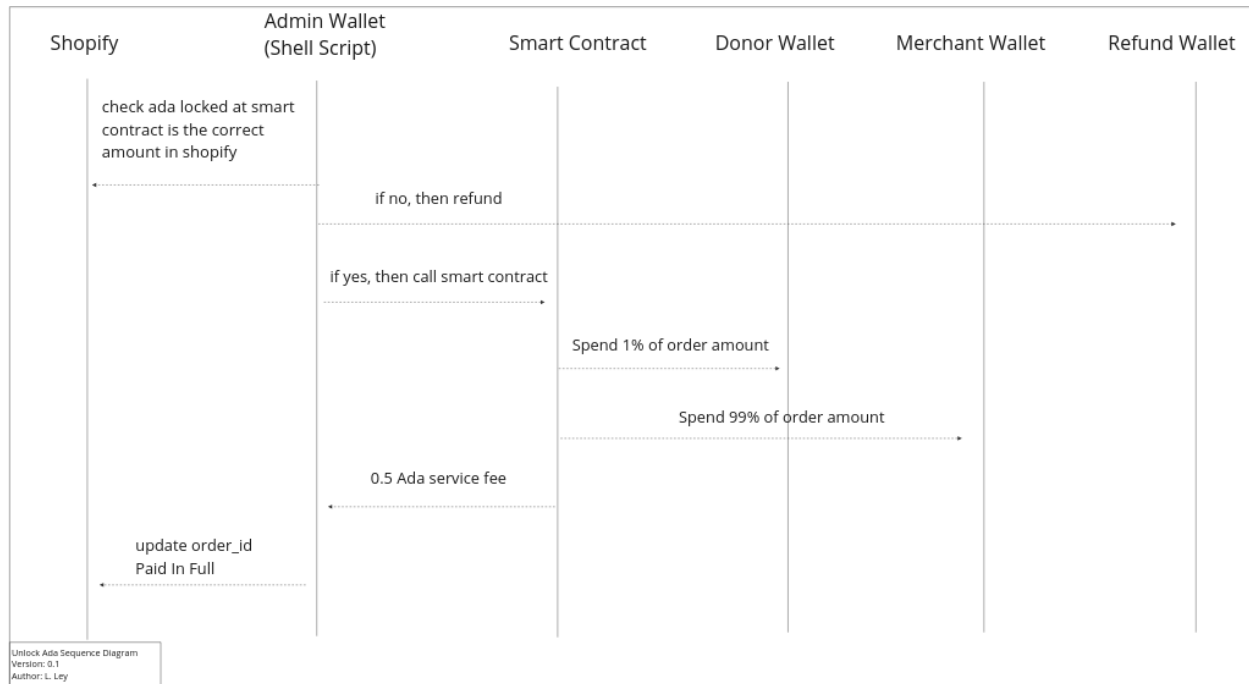
361.76 ADA addr_test1.pmr54 229.31 ADA addr_test1.pmr54

132.28 ADA addr_test1.ckkOK

기부금 추적 기능(Ada 잠금 해제)

이전 섹션에서는 이커머스 주문을 구매하는 동안 기부 추적 스마트 컨트랙트에 **Ada**를 잠갔습니다. 이 섹션에서는 해당 **Ada**를 잠금 해제하고 그에 따라 판매자 및 자선 단체에 보내는 방법을 안내합니다.

스마트 컨트랙트에서 Ada 잠금 해제하기



Ada의 잠금 해제는 관리자가 **bash** 셸 스크립트를 실행해야 합니다. 기억하시다시피, 기부 스마트 컨트랙트에 잠긴 **Ada**를 사용하는 경우에만 **Ada** 지출이 성공합니다. 이 경우 성공적인 지출은 다음과 같은 조건들이 충족되어야 합니다:

1. 트랜잭션은 관리자 **PKH**가 서명합니다.
2. 출력은 그에 따라 판매자 및 기부자 주소로 전달됩니다.
3. 판매자와 기부자에게 전송된 **Ada**의 금액은 스마트 컨트랙트에 정의된 기부금 분할과 일치합니다.

처리 중에 데이터의 주문 금액이 **Shopify**의 주문 금액과 일치하는지 확인하기 위해 몇 가지 추가 확인 사항이 있습니다. 이는 누구나 스마트 컨트랙트 스크립트 주소의 데이터 값으로 **Ada**를 잠글 수 있기 때문에 필요합니다. 잠긴 **UTXO**가 유효한지 확인할 수 있는 추가 오프체인 로직이 필요합니다. 마지막으로 거래가 성공하면 주문이 전액 결제되었음을 **Shopify**에 업데이트합니다.

Bash 셸 스크립트

VS Code 웹의 터미널 창에서 `cd donation/scripts`로 이동하여 `bash` 셸 스크립트와 이를 실행하는 데 필요한 파일을 확인합니다.

Bash 셸 스크립트

```
./cardano-cli/init-tx.sh  
./cardano-cli/refund-tx.sh  
./cardano-cli/spend-tx.sh
```

환경별 설정

```
./cardano-cli/preprod/global-export-variables.sh
```

환경별 데이터 파일

```
./cardano-cli/preprod/data/donation.plutus  
./cardano-cli/preprod/data/donation.addr  
./cardano-cli/preprod/data/redeemer-spend.json  
./cardano-cli/preprod/data/black-list-utxo.txt  
./cardano-cli/preprod/data/donation.hash  
./cardano-cli/preprod/data/redeemer-refund.json  
./cardano-cli/preprod/data/donation-spend-metadata.json
```

`init-tx.sh` 셸 스크립트를 실행하여 참조 스크립트로 스마트 컨트랙트를 블록체인에 로드해야 합니다. 그러나 이전에 **2개의 UTXO**가 관리자 지갑 주소에 있어야 하며, 각각 **5 Ada**의 담보금과 업로드 비용 및 트랜잭션 수수료를 지불하기 위한 **25 Ada** 이상이 있어야 합니다.

다음과 같은 방식이 적합합니다.

```
$ cardano-cli query utxo --address addr_test1vzu6...dxn7 --cardano-mode
--testnet-magic 1
TxHash TxIx Amount
```

```
-----
9f0...d82 1 79759567 lovelace + TxOutDatumNone
e06...8d6 0 5000000 lovelace + TxOutDatumNone
```

타겟 환경(preprod)에 맞게 **scripts/cardano-cli/preprod/global-export-variables.sh** 파일을 업데이트하세요. 초기 설정 시 대부분의 값은 기본적으로 작동하지만, 주소들과 키들은 당신이 제어하는 주소들과 키들로 변경해야 합니다.

```
# Define export variables
export BASE=/config/workspace/repo/donation
export WORK=$BASE/work
export TESTNET_MAGIC=1
export ADMIN_VKEY=/config/workspace/repo/.keys/admin/key.vkey
export ADMIN_SKEY=/config/workspace/repo/.keys/admin/key.skey
export ADMIN_PKH=/config/workspace/repo/.keys/admin/key.pkh
export MIN_ADA_OUTPUT_TX=2000000
export MIN_ADA_OUTPUT_TX_REF=25000000
export COLLATERAL_ADA=5000000
export
MERCHANT_ADDR=addr_test1vq7k90717e59t52skm8e0ezsnmmc7h4xy30kg2klwc5n8rqug2pds
export
DONOR_ADDR=addr_test1vzetpfww4aaunft0ucvcrxugj8nt4lhltsktya0rx0uh48cqghjfg
export
REFUND_ADDR=addr_test1vzs24vjh8salzqt5pahgvr34ewfwagxaxr5pz7eswugzn2gmw4f5w
export
VAL_REF_SCRIPT=b8a3af2835b5f2789b1f9e1d53c953a26db9a7dfab0e204e8629d85772c9fc3
0#1
export SPLIT=90
export MIN_ADA_DONATION=1000000
```

참조 스크립트

참조 스크립트는 트랜잭션에 입력으로 포함할 수는 있지만 사용할 수 없는 읽기 전용 **UTXO**입니다. 이렇게 하면 참조 스크립트 **UTXO**를 둘 이상의 트랜잭션에서 병렬 또는 순차적으로 재사용할 수 있습니다.

init-tx.sh 쉘 스크립트를 실행하여 **plutus** 스크립트를 블록체인에 업로드하세요.

```
$ cd scripts/cardano-cli
$ ./init-tx.sh preprod
```

트랜잭션이 성공적으로 제출되면 스크립트 주소에서 **25 Ada**가 잠긴 참조 **UTXO**를 볼 수 있을 것입니다. 이전 섹션의 주문에서 **Ada**가 잠긴 **UTXO**도 볼 수 있습니다. 여기에는 주문 총액, 주문 **ID**, **ADA/USD** 가격 변환이 포함된 인라인 데이텀이 있습니다.

```
$ cardano-cli query utxo --address addr_test1wr...kk0k --cardano-mode
--testnet-magic 1
```

Amount		TxHash	TxIx

413...aa8	0	132280000 lovelace + TxOutDatumInline	
ReferenceTxInsScriptsInlineDatumsInBabbageEra (ScriptDataList			
[ScriptDataNumber 131780000,ScriptDataBytes "5247401754903",ScriptDataBytes			
"0.37941"])			
d6c...2ce	0	25000000 lovelace + TxOutDatumNone	

그런 다음 **preprod/global-environment-variables.sh** 파일을 업데이트하여 참조 스크립트 **UTXO TxHash** 및 **TxIdx**를 설정해야 합니다.

```
export VAL_REF_SCRIPT=d6c...2ce3#0
```

지출(Spend) 스크립트

이제 **spend-tx.sh** 쉘 스크립트를 실행할 수 있습니다.

```
./spend-tx.sh preprod
```

이것이 **spend-tx.sh bash** 쉘 스크립트입니다.

```
#!/usr/bin/env bash
set -e
set -o pipefail

# bash 쉘에 대한 디버그 플래그를 활성화합니다.
set -x

# 명령줄 인수(arguments)가 비어 있거나 없는지 확인합니다.
if [ -z $1 ];
then
    echo "process-tx.sh: Invalid script arguments"
    echo "Usage: process-tx.sh [devnet|preview|preprod|mainnet]"
    exit 1
fi
ENV=$1

# 전역 변수를 가져옵니다.
MY_DIR=$(dirname $(readlink -f $0))
source $MY_DIR/$ENV/global-export-variables.sh
if [ "$ENV" == "mainnet" ];
then
    network="--mainnet"
else
    network="--testnet-magic $TESTNET_MAGIC"
fi
mkdir -p $WORK
mkdir -p $WORK-backup
rm -f $WORK/*
rm -f $WORK-backup/*

# cardano-cli 툴에서 값 생성
cardano-cli query protocol-parameters $network --out-file $WORK/pparms.json

# 로컬 변수 값으로 로드
validator_script="$BASE/scripts/cardano-cli/$ENV/data/donation.plutus"
validator_script_addr=$(cardano-cli address build --payment-script-file
"$validator_script" $network)
redeemer_file_path="$BASE/scripts/cardano-cli/$ENV/data/redeemer-spend.json"
admin_pkh=$(cat $ADMIN_PKH)
```



```

#####
# 기부금 UTXO 사용
#####
# 1단계: 관리자로부터 UTXO 가져오기
#####
# 소비할 수 있는 UTXO는 2개 이상이어야 합니다.
# 하나는 토큰 소비용, 다른 하나는 담보용 UTXO입니다.
#####

admin_utxo_addr=$(cardano-cli address build $network
--payment-verification-key-file "$ADMIN_VKEY")
cardano-cli query utxo --address "$admin_utxo_addr" --cardano-mode $network
--out-file $WORK/admin-utxo.json

cat $WORK/admin-utxo.json | jq -r 'to_entries[] | select(.value.value.lovelace
> '$COLLATERAL_ADA' ) | .key' > $WORK/admin-utxo-valid.json

readarray admin_utxo_valid_array < $WORK/admin-utxo-valid.json

admin_utxo_in=$(echo $admin_utxo_valid_array | tr -d '\n')

cat $WORK/admin-utxo.json | jq -r 'to_entries[] | select(.value.value.lovelace
== '$COLLATERAL_ADA' ) | .key' > $WORK/admin-utxo-collateral-valid.json

readarray admin_utxo_collateral_array < $WORK/admin-utxo-collateral-valid.json

admin_utxo_collateral_in=$(echo $admin_utxo_collateral_array | tr -d '\n')

readarray black_list_utxo_array <
$BASE/scripts/cardano-cli/$ENV/data/black-list-utxo.txt

#####
# 2단계: 기부 스마트 컨트랙트 utxos 받기
#####
cardano-cli query utxo --address $validator_script_addr $network --out-file
$WORK/validator-utxo.json

cat $WORK/validator-utxo.json | jq -r 'to_entries[] |
select(.value.inlineDatum | length > 0) | .key' > $WORK/order_utxo_in.txt
readarray order_utxo_in_array < $WORK/order_utxo_in.txt
order_array_length="${#order_utxo_in_array[@]}"

order_utxo_in=""

# 블랙리스트에 없는 utxo 찾기
for (( c=0; c<$order_array_length; c++ )) do
    if printf '%s' "${black_list_utxo_array[@]}" | grep -q -x
        "${order_utxo_in_array[$c]}";

```

```

then
    echo "UTXO on blacklist: ${order_utxo_in_array[$c]}"
else
    order_utxo_in=$(echo ${order_utxo_in_array[$c]} | tr -d '\n')
break
fi
Done

# 유효성 검사기에 사용할 수 있는 utxos가 있는지 확인합니다.
# 사용할 수 있는지 확인하고, 없다면 종료합니다.
if [ -z $order_utxo_in ];
then
exit 0
Fi

# Get the correct datum UTXO
order_datum_in=$(jq -r 'to_entries[]
| select(.key == "'$order_utxo_in'")
| .value.inlineDatum ' $WORK/validator-utxo.json)
echo -n "$order_datum_in" > $WORK/datum-in.json
# Get the order details from the datum
order_ada=$(jq -r '.list[0].int' $WORK/datum-in.json)

order_id_encoded=$(jq -r '.list[1].bytes' $WORK/datum-in.json) echo
-n "$order_id_encoded" > $WORK/order_id.encoded order_id=$(python3
hexdump.py -r $WORK/order_id.encoded)

ada_usd_price_encoded=$(jq -r '.list[2].bytes' $WORK/datum-in.json)
echo -n "$ada_usd_price_encoded" > $WORK/ada_usd_price.encoded
ada_usd_price=$(python3 hexdump.py -r $WORK/ada_usd_price.encoded)

merchant_split=$SPLIT
donor_split=$((100 - $SPLIT))
donor_ada_amount=$(( $order_ada * $donor_split / 100))

if (($donor_ada_amount < $MIN_ADA_DONATION ));
Then
    donor_ada=$MIN_ADA_DONATION
else
    donor_ada=$donor_ada_amount
Fi

merchant_ada=$(( $order_ada - $donor_ada))

now=$(date '+%Y/%m/%d-%H:%M:%S')

```

```

# 주문 결제 금액이 Shopify의 주문 금액과 동일한지 확인 동일한지
확인합니다.
shopify_order_amount=$(curl -H "X-Shopify-Access-Token:
$NEXT_PUBLIC_ACCESS_TOKEN"
"$NEXT_PUBLIC_SHOP/admin/api/2022-10/orders/"$order_id".json" | jq -r
'.order.total_price')

shopify_order_ada=$(python3 -c "print(round(($shopify_order_amount /
$ada_usd_price), 3))")

shopify_order_lovelace=$(python3 -c "print($shopify_order_ada * 1000000)")
shopify_order_ada_truncated=${shopify_order_lovelace%.*}

difference=$(( $order_ada - $shopify_order_ada_truncated ))
difference_abs=$(echo ${difference#-})

if (( $difference_abs > 10000 ));
then
    echo "Order amount mismatch between order amount in datum vs order amount
    in shopify for $order_id"
    exit -1
fi

metadata="{
  \"1\" : {
    \"order_detail\" : {
      \"date\" : \"$now\",
      \"donation_ada_amount\" : \"$donor_ada\",
      \"donation_split\" : \"$donor_split%\",
      \"order_id\" : \"$order_id\",
      \"order_ada_amount\" : \"$order_ada\",
      \"ada_usd_price\" : \"$ada_usd_price\",
      \"version\" : \"0.1\"
    }
  }
}"

echo $metadata >
$BASE/scripts/cardano-cli/$ENV/data/donation-spend-metadata.json
metadata_file_path="$BASE/scripts/cardano-cli/$ENV/data/donation-spend-metadat
a.json"

#####
# 3단계: 트랜잭션 빌드 및 제출
#####
cardano-cli transaction build \
--babbage-era \
--cardano-mode \
$network \
--change-address "$admin_utxo_addr" \

```

```

--tx-in-collateral "$admin_utxo_collateral_in" \
--tx-in "$admin_utxo_in" \
--tx-in "$order_utxo_in" \
--spending-tx-in-reference "$VAL_REF_SCRIPT" \
--spending-plutus-script-v2 \
--spending-reference-tx-in-inline-datum-present \
--spending-reference-tx-in-redeemer-file "$redeemer_file_path" \
--tx-out "$MERCHANT_ADDR+$merchant_ada" \
--tx-out "$DONOR_ADDR+$donor_ada" \
--required-signer-hash "$admin_pkh" \
--protocol-params-file "$WORK/pparms.json" \
--metadata-json-file "$metadata_file_path" \
--out-file $WORK/spend-tx-alonzo.body

echo "tx has been built"

cardano-cli transaction sign \
--tx-body-file $WORK/spend-tx-alonzo.body \
$network \
--signing-key-file "${ADMIN_SKEY}" \
--out-file $WORK/spend-tx-alonzo.tx

echo "tx has been signed"

echo "Submit the tx with plutus script and wait 5 seconds..."
cardano-cli transaction submit --tx-file $WORK/spend-tx-alonzo.tx $network

# 주문이 전액 결제되도록 Shopify 업데이트
curl -s -S -d '{"order":{"id":"$order_id"},"tags":["PAID IN FULL"]}' \
-X PUT "${NEXT_PUBLIC_SHOP}admin/api/2022-10/orders/$order_id.json" \
-H "X-Shopify-Access-Token: $NEXT_PUBLIC_ACCESS_TOKEN" \
-H "Content-Type: application/json" > /dev/null

```

Test Drive

tx-spend.sh 실행 전

```
$ cardano-cli query utxo --address  
addr_test1wr3m0vrrcccxygtmexw7ur7vujjz5gqg4fjn0k2sjsjn04sdckk0k --cardano-mode --testnet-magic  
1
```

TxHash	TxIx	Amount
4131abc70c5336b2896a9b0c8a9699bf510b24c8d0clf0edb4497595fc5acaa8	0	132280000 lovelace
+ TxOutDatumInline ReferenceTxInsScriptsInlineDatumsInBabbageEra (ScriptDataList [ScriptDataNumber 131780000,ScriptDataBytes "5247401754903",ScriptDataBytes "0.37941"])		
d6c4a585ffbbdcc371a3472e829133643c9dfa4f1b61a4ccb8e75fd4480c22ce	0	25000000 lovelace
+ TxOutDatumNone		

tx-spend.sh 실행 후

```
$ cardano-cli query utxo --address  
addr_test1wr3m0vrrcccxygtmexw7ur7vujjz5gqg4fjn0k2sjsjn04sdckk0k --cardano-mode  
--testnet-magic 1
```

TxHash	TxIx	Amount
d6c4a585ffbbdcc371a3472e829133643c9dfa4f1b61a4ccb8e75fd4480c22ce	0	25000000 lovelace
+ TxOutDatumNone		

explorer.io에서 성공적인 트랜잭션을 확인할 수 있습니다.

Google Chrome Jan 27 11:38

Helios Tx Builder Transaction c3b2f3721139c7d38885f012bab6cae0cff96a405c14a228fe382f6b7b21f074

preprod.cexplorer.io/tx/c3b2f3721139c7d38885f012bab6cae0cff96a405c14a228fe382f6b7b21f074

Cexplorer.io

CARDANO EXPLORER

- Dashboard
- Watchlist
- Pools
- Assets
- Blocks
- Transactions
- dApps
- Metadata
- More

EDUCATION

- Articles
- Videos
- Wiki

ANALYTICS

- Decentralization
- Network

Transaction detail

Hash: c3b2f3721139c7d38885f012bab6cae0cff96a405c14a228fe382f6b7b21f074

Date	6m20s ago
Epoch	48
Block	564,597
Slot	19,153,936 (epoch slot 59,536)
Total Output	634.07 ₳ \$ 241.5 ₮ 901048
Fee	0.23 ₳ \$ 0.1
Outputs	3

22 Assurance

0.71kB (0.71kB) Transaction Size

0.71kB (1) Block Size (TX Count)

Minted by [BLADE] BLADE Pool - bladepool.com

Content Metadata (1)

6m20s ago # c3b2f3721139c7d38885f012bab6cae0cff96a405c14a228fe382f6b7b21f074 Total 634.07 ₳ \$ 241.5 ₮ 901048

502.02 ₳ \$ 192.2 ₮ 700000 addr_test1_gdxxn7	502.29 ₳ \$ 192.4 ₮ 700000 addr_test1_gdxxn7
132.28 ₳ \$ 50.3 ₮ 180048 addr_test1_ckkOk	118.6 ₳ \$ 44.8 ₮ 160000 addr_test1_g2pds
	1318 ₳ \$ 50.3 ₮ 180048 addr_test1_gfyyg

블록체인 모니터링

이 장에서는 애플리케이션이 반응해야 하는 변경 사항이 있는지 블록체인을 모니터링하는 방법을 중점적으로 살펴보겠습니다. 가장 기본적인 모니터링 유형은 먼저 설명할 폴링입니다. 그런 다음 오류 처리와 이벤트 필터링에 더 많은 기능을 갖춘 고급 모니터링인 이벤트 기반 모니터링에 대해 살펴보겠습니다.

폴링

이전 기부 예시를 참조하여, 스마트 컨트랙트에서 **Ada**가 잠기는 시점을 모니터링해야 합니다. 이런 상황이 발생하면 스크립트가 트랜잭션을 실행하여 그에 따라 **UTXOs**를 소비합니다.

Cron

한 가지 접근 방식은 정기적으로 **spend-tx.sh** **bash** 스크립트를 시작하도록 크론 작업을 설정하는 것입니다. 이 스크립트는 **Node.js**로도 작성할 수 있지만, 이 경우에는 **bash** 셸 스크립트를 그대로 재사용하겠습니다.

매 분마다 스크립트를 실행하는 간단한 크론 작업은 다음과 같습니다.

```
* * * * * (cd /absolute-path-to-script-directory/; ./spend-tx.sh preprod >>
/absolute-path-to-log-directory/preprod.out)
```

유효한 UTXOs

bash 셸 스크립트가 실행되면 현재 스마트 컨트랙트 스크립트 주소에 잠긴 모든 **UTXO**를 가져옵니다. 유효한 **UTXO** 항목에 대해 **bash** 스크립트는 트랜잭션을 빌드하고 서명하여 제출합니다. 누구나 스크립트 주소에 무엇이든 잠글 수 있기 때문에, 필요한 경우 특정 **UTXOs**를 블랙리스트에 추가하는 메커니즘이 만들어졌습니다. 사용할 수 없는 **UTXO**는 부정확하거나 누락된 기준값의 결과일 수 있습니다.

모니터링

아래는 간단한 모니터링 스크립트로, 스크립트 주소에서 **UTXOs**의 개수가 **1**을 초과하면 수동 조사가 필요합니다. 사용할 수 없는 **UTXO**인 경우 블랙리스트에 추가하여 스크립트에서 특정 **UTXO**를 건너뛰도록 할 수 있습니다. 스크립트 주소에 **1** 개 이상의 **UTXO**가 잠겨있는 경우, 텔레그램 그룹에 콜투액션을 위한 메시지가 전송됩니다.

다음은 **bash** 셸 모니터링 스크립트입니다

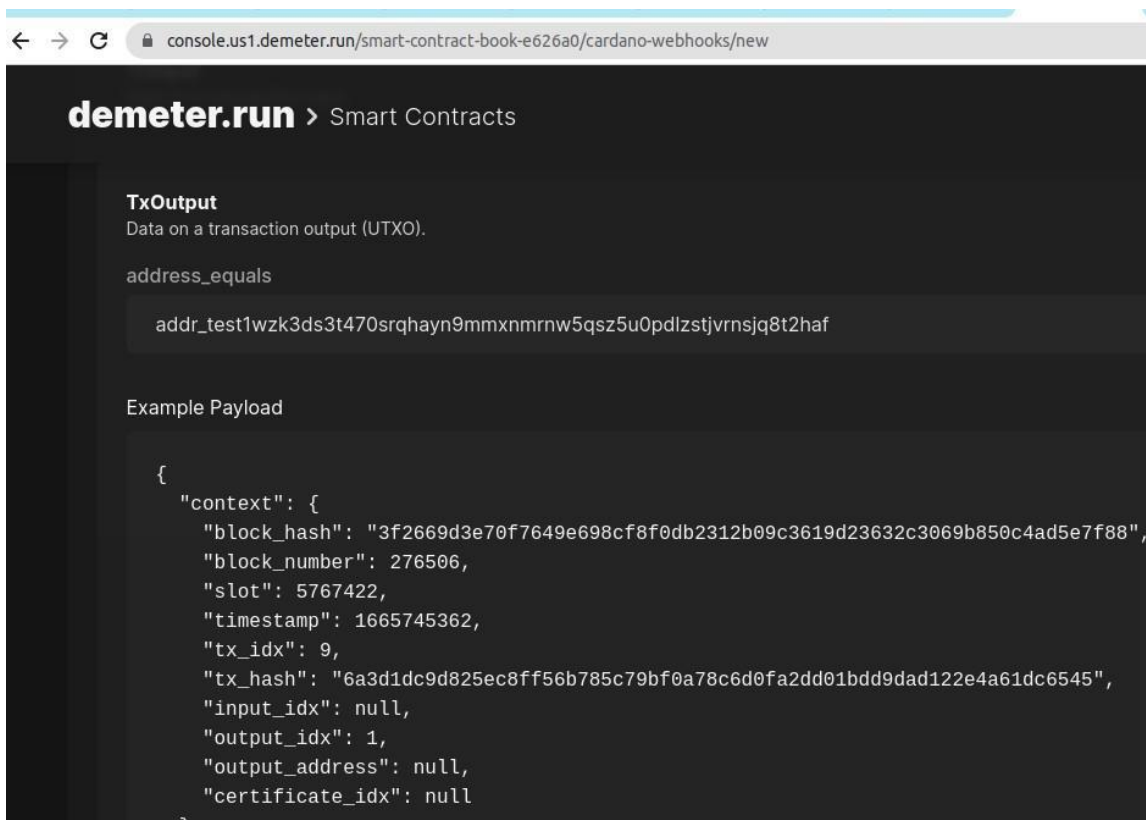
```
#!/usr/bin/bash
count=$(/usr/local/bin/cardano-cli query utxo --address
addr_test1wzk3ds3t470srqhayn9mmxnmrnw5qsz5u0pdlzstjvrnsjq8t2haf --cardano-mode
--testnet-magic 1 | wc -l)
if (( $count > 3 ));
then
    /usr/bin/curl -X POST \
    -H 'Content-Type: application/json' \
    -d '{"chat_id": "XXXXXXXXXX", "text": "Prod utxo count: '$count'",
    "disable_notification": true}' \
    https://api.telegram.org/botYYYYYY:ZZZZZZZZZZ\_xtc/sendMessage
fi
```

이벤트

이벤트 기반 모니터링은 주로 블록체인 활동을 실시간으로 모니터링하는 것을 포함합니다. 예를 들어, 스마트 컨트랙트 주소에서 **Ada**를 잠그는 트랜잭션이 이벤트를 생성하고 애플리케이션이 특정 유형의 작업으로 응답할 수 있습니다. 필요한 경우 여러 번의 [블록 확인](#)이 발생한 후에야 이벤트가 트리거될 수도 있습니다. 또한 블록체인 [롤백](#)이 발생하여 애플리케이션이 이를 적절히 처리해야 하는 경우에도 이벤트가 트리거될 수 있습니다.

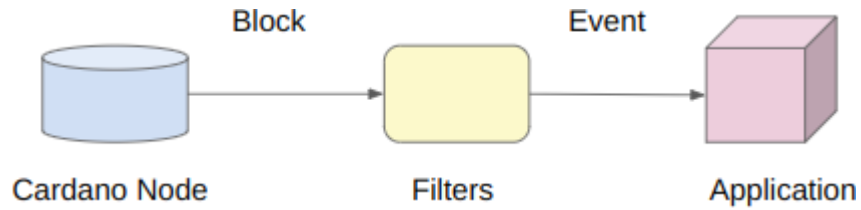
Oura

이제는 **Cardano** 블록체인에 대한 모든 인프라 및 연결이 설정되어 있고 바로 사용할 수 있는 서비스 제공 업체가 있습니다. 당신이 해야 할 일은 실행할 블록체인 이벤트를 선택하는 것입니다. 이번 경우에는 **Oura**를 사용하여 **TxOutput** 이벤트를 선택하고 기부 스마트 컨트랙트 스크립트 주소를 감시할 것입니다. 그런 다음 해당 이벤트가 발생하면 웹훅이 호출되어 트랜잭션 해시와 인덱스가 전달되며, 이를 처리하기 위해 **Next.js** 애플리케이션을 실행합니다.



이벤트 흐름

다음 추상 다이어그램은 이벤트가 생성되고 처리되는 방식을 보여줍니다. **Cardano** 노드에서 수신된 블록에 대해 필터가 설정됩니다. 필터가 일치하면 이벤트가 트리거되고 어플리케이션 웹훅이 호출됩니다.



Oura에 대한 자세한 정보는 <https://txpipe.github.io/oura/>에서 찾을 수 있습니다.

웹훅

모니터링하려는 이벤트와 이러한 이벤트가 발생할 때 **Oura**가 호출해야 하는 웹훅의 **URL**을 선택하세요. 웹훅은 **Next.js** 어플리케이션에서 **API**를 설정하여 생성됩니다. 이 **API**는 특정 트랜잭션 해시 및 인덱스로 **spend-tx.sh** 스크립트를 실행합니다.

새로운 **API**에서 **Oura**로부터 수신된 웹훅을 처리하기 위해 만든 **events.ts** 코드는 다음과 같습니다.

```
import path from 'path';
import type { NextApiRequest, NextApiResponse } from 'next'
import initMiddleware from '../lib/init-middleware'
import validateMiddleware from '../lib/validate-middleware'
import { check, validationResult } from 'express-validator'

const validateBody = initMiddleware(
  validateMiddleware([
    check('context.tx_idx').isInt({ min: 0, max: 255}),
    check('context.tx_hash').isBase64(),
  ], validationResult)
)

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse ) {
  const network = process.env.NEXT_PUBLIC_NETWORK as string;
  const eventAPIKey =
    process.env.NEXT_PUBLIC_EVENT_API_KEY as string;
  if (req.method == 'POST') {
    // 기본 인증 헤더 확인
    if (!req.headers.authorization ||
      req.headers.authorization.indexOf('Basic ') === -1) {
      throw { status: 401,
        message: 'Missing Authorization Header' };
    }
    // 인증 자격 증명 확인
    const apiKey = req.headers.authorization.split(' ')[1];
    if (eventAPIKey != apiKey) {
      throw { status: 401,
        message: 'Invalid Authentication Credentials' };
    }
  }
}
```

```

// 본문 입력값 처리
await validateBody(req, res)
const errors = validationResult(req)
if (!errors.isEmpty()) {
    return res.status(422).json({ errors: errors.array() })
}
const txId = req.body.context.tx_hash + ` ` +
    req.body.context.tx_idx;

const shell = require('shelljs');
const scriptsDirectory = path.join(process.cwd(),
    'scripts/cardano-cli');
const cmd = `(cd ` + scriptsDirectory + `; ./spend-tx.sh ` +
    network + ` ` + txId + `)`;

if (shell.exec(cmd).code !== 0) {
    res.setHeader('Tx-Status', 'Tx
    Failed');
    // Oura가 재시도하지 않도록 200 상태를 반환합니다.
    res.status(200).json(`Tx Failed: ` + txId );
} else {
    res.setHeader('Tx-Status', 'Tx Submitted');
    res.status(200).json(`Tx Submitted: ` + txId);
}
}
else {
    res.status(400);
    res.send(`Invalid API Request`);
}
}

```

spend-tx.sh 스크립트는 명령줄 인수(arguments)를 통해 트랜잭션 해시 (tx_hash) 및 인덱스 (tx_idx)를 받도록 변경되었습니다.

스마트 컨트랙트 & Next.js 설정

모든 것이 올바르게 설정되었는지 확인합니다.

1. donation.hl 버전 번호 늘리기 (필요한 경우)
2. `$ npx deno-bin run --allow-read --allow-write ./src/deploy-donation.js`
3. `$ cp deploy/* scripts/cardano-cli/preprod/data/`
4. `$ cp src/donation.hl contracts/`
5. `$ cd scripts/cardano-cli/`
6. `$./init-tx.sh preprod`
7. Query the new script address and make sure the reference script is there.

```
$ cardano-cli query utxo --address addr_test1wpg...42k --cardano-mode --testnet-magic 1
```

		TxHash		TxIx	Amount

78d...9ac	0	25000000	lovelace + TxOutDatumNone		

8. Update the global-export-variables.sh file with scripts reference UTXO from the previous step
9. Ensure the correct env variables are set (see Donation section)
10. Start the Next.js application

```
$ npm install
$ npm run dev
```
11. Create a webhook for the new smart contract address in Oura

Oura 설정

Oura 호스팅 솔루션을 사용하여 **Demeter Run** 웹훅을 설정하는 데 필요한 단계는 다음과 같습니다.

1. Go to Demeter Run and log in
2. Open or create a project
3. Select the Features tab
4. Select Cardano Webhooks
5. Select Create Webhook button
6. Enter the URL of your webhook (<https://some-url/api/events>)
7. Select Headers -> Add
 - a. Authorization
 - b. Basic YOUR-API-KEY
8. Toggle TxOutput Event on

Test Drive

주문은 온체인으로 제출되며 스마트 컨트랙트 주소에 잠깁니다.

Google Chrome Jan 27 13:59

Transaction 704b195961b0c4c72cb496971d81e57f0344e3edb1030d81d6239d2d7e0654b9

preprod.cexplorer.io/tx/704b195961b0c4c72cb496971d81e57f0344e3edb1030d81d6239d2d7e0654b9

Cexplorer.io preprod

your button

Hi, SleepingCharles

CARDANO EXPLORER

- Dashboard
- Watchlist
- Pools
- Assets
- Blocks
- Transactions
- dApps
- Metadata
- More

EDUCATION

- Articles
- Videos
- Wiki

ANALYTICS

- Decentralization
- Network

Transactions / Transaction 704b195961b0c4c72c...

Transaction detail

Hash: 704b195961b0c4c72cb496971d81e57f0344e3edb1030d81d6239d2d7e0654b9

Date	4m20s ago
Epoch	48
Block	564,953
Slot	19,102,514 (epoch slot 68,114)
Total Output	203.43 ₳ \$ 77.9 ₮ 0.00336
Fee	0.17 ₳ \$ 0
Outputs	2

16 Assurance

0.29kB (0.29kB)
Transaction Size
this tx 99%

0.29kB (1)
Block Size (TX Count)
Available 100%

Minted by
BGR BURGER STAKE POOL

Content

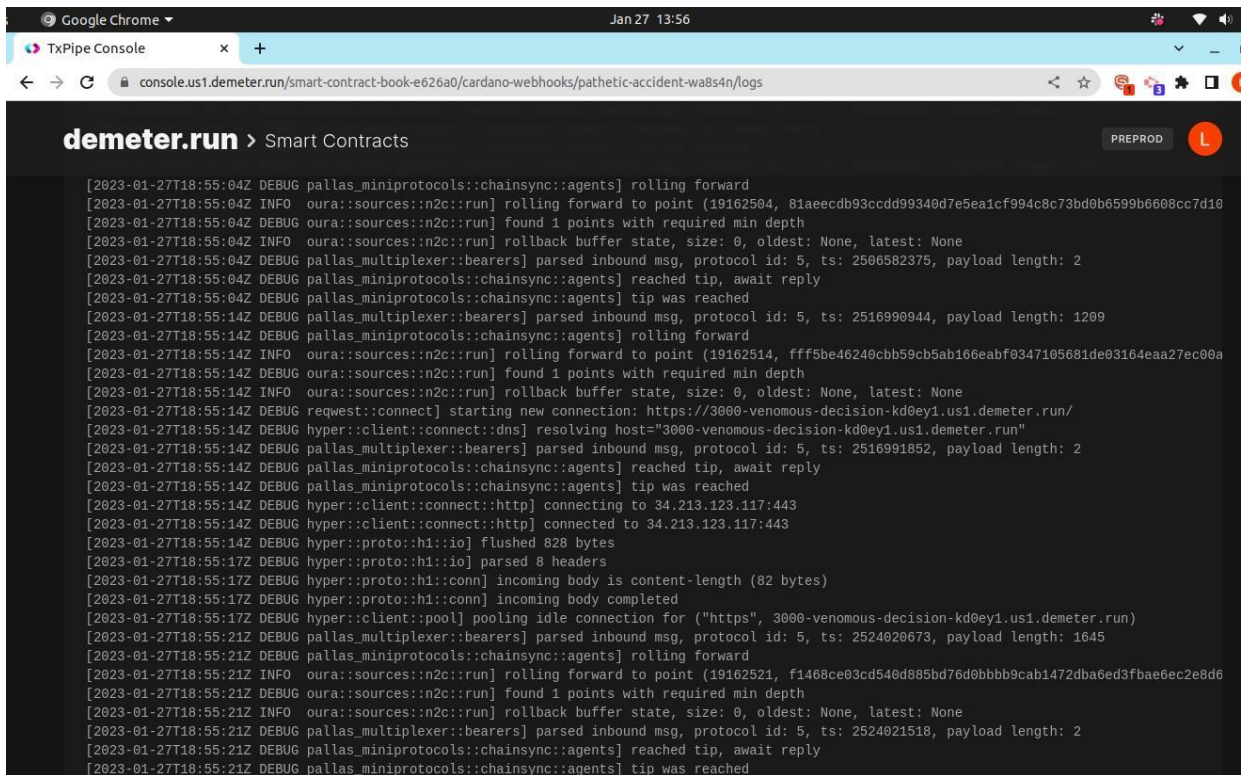
4m20s ago # 704b195961b0c4c72cb496971d81e57f0344e3edb1030d81d6239d2d7e0654b9 Total 203.43 ₳

203.59 ₳ \$ 77.9 ₮ 0.00336 addr_test1.pmr54

130.97 ₳ \$ 49.9 ₮ 0.00277 addr_test1.f842k

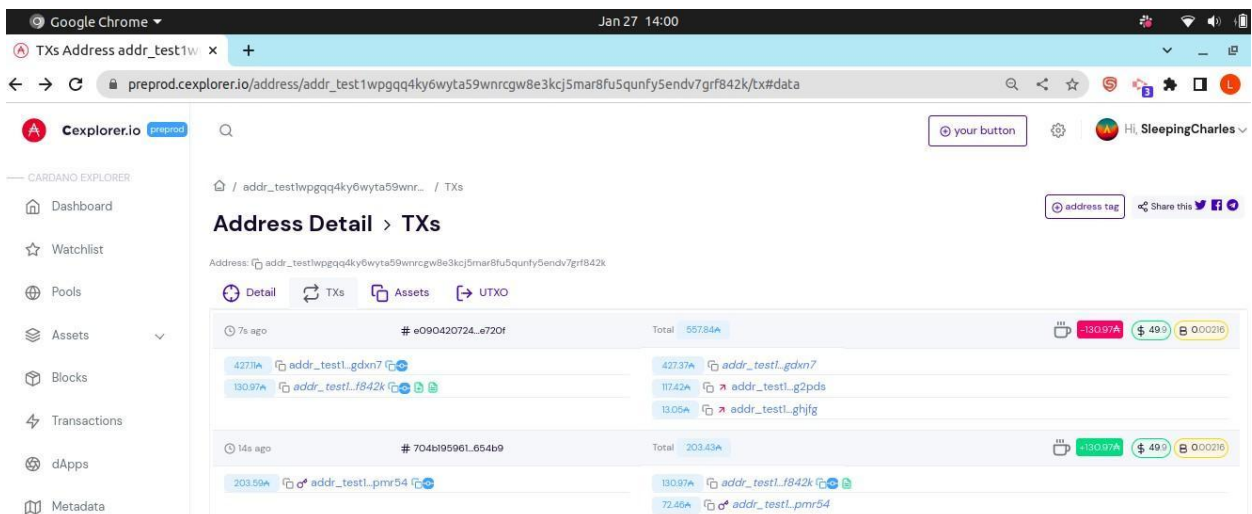
72.56 ₳ \$ 27.9 ₮ 0.00277 addr_test1.pmr54

스마트 컨트랙트 스크립트 주소의 트랜잭션 이벤트가 감지되면 **Oura**에서 액션이 트리거됩니다.



```
[2023-01-27T18:55:04Z DEBUG pallas_miniprotocols::chainsync::agents] rolling forward
[2023-01-27T18:55:04Z INFO oura::sources::n2c::run] rolling forward to point (19162504, 81aeecdb93ccdd99340d7e5ea1cf994c8c73bd0b6599b6608cc7d10)
[2023-01-27T18:55:04Z DEBUG oura::sources::n2c::run] found 1 points with required min depth
[2023-01-27T18:55:04Z INFO oura::sources::n2c::run] rollback buffer state, size: 0, oldest: None, latest: None
[2023-01-27T18:55:04Z DEBUG pallas_multiplexer::bearers] parsed inbound msg, protocol id: 5, ts: 2506582375, payload length: 2
[2023-01-27T18:55:04Z DEBUG pallas_miniprotocols::chainsync::agents] reached tip, await reply
[2023-01-27T18:55:04Z DEBUG pallas_miniprotocols::chainsync::agents] tip was reached
[2023-01-27T18:55:14Z DEBUG pallas_multiplexer::bearers] parsed inbound msg, protocol id: 5, ts: 2516990944, payload length: 1209
[2023-01-27T18:55:14Z DEBUG pallas_miniprotocols::chainsync::agents] rolling forward
[2023-01-27T18:55:14Z INFO oura::sources::n2c::run] rolling forward to point (19162514, fff5be46240cbb59cb5ab166eabf0347105681de03164eaa27ec00a)
[2023-01-27T18:55:14Z DEBUG oura::sources::n2c::run] found 1 points with required min depth
[2023-01-27T18:55:14Z INFO oura::sources::n2c::run] rollback buffer state, size: 0, oldest: None, latest: None
[2023-01-27T18:55:14Z DEBUG request::connect] starting new connection: https://3000-venomous-decision-kd0ey1.us1.demeter.run/
[2023-01-27T18:55:14Z DEBUG hyper::client::connect::dns] resolving host="3000-venomous-decision-kd0ey1.us1.demeter.run"
[2023-01-27T18:55:14Z DEBUG pallas_multiplexer::bearers] parsed inbound msg, protocol id: 5, ts: 2516991852, payload length: 2
[2023-01-27T18:55:14Z DEBUG pallas_miniprotocols::chainsync::agents] reached tip, await reply
[2023-01-27T18:55:14Z DEBUG pallas_miniprotocols::chainsync::agents] tip was reached
[2023-01-27T18:55:14Z DEBUG hyper::client::connect::http] connecting to 34.213.123.117:443
[2023-01-27T18:55:14Z DEBUG hyper::client::connect::http] connected to 34.213.123.117:443
[2023-01-27T18:55:14Z DEBUG hyper::proto::h1::io] flushed 828 bytes
[2023-01-27T18:55:17Z DEBUG hyper::proto::h1::io] parsed 8 headers
[2023-01-27T18:55:17Z DEBUG hyper::proto::h1::conn] incoming body is content-length (82 bytes)
[2023-01-27T18:55:17Z DEBUG hyper::proto::h1::conn] incoming body completed
[2023-01-27T18:55:17Z DEBUG hyper::client::pool] pooling idle connection for ("https", 3000-venomous-decision-kd0ey1.us1.demeter.run)
[2023-01-27T18:55:21Z DEBUG pallas_multiplexer::bearers] parsed inbound msg, protocol id: 5, ts: 2524020673, payload length: 1645
[2023-01-27T18:55:21Z DEBUG pallas_miniprotocols::chainsync::agents] rolling forward
[2023-01-27T18:55:21Z INFO oura::sources::n2c::run] rolling forward to point (19162521, f1468ce03cd540d885bd76d0bbb9cab1472dba6ed3fbae6ec2e8d6)
[2023-01-27T18:55:21Z DEBUG oura::sources::n2c::run] found 1 points with required min depth
[2023-01-27T18:55:21Z INFO oura::sources::n2c::run] rollback buffer state, size: 0, oldest: None, latest: None
[2023-01-27T18:55:21Z DEBUG pallas_multiplexer::bearers] parsed inbound msg, protocol id: 5, ts: 2524021518, payload length: 2
[2023-01-27T18:55:21Z DEBUG pallas_miniprotocols::chainsync::agents] reached tip, await reply
[2023-01-27T18:55:21Z DEBUG pallas_miniprotocols::chainsync::agents] tip was reached
```

웹훅 API가 호출되어 트랜잭션을 성공적으로 제출합니다.



Cexplorer.io PREPROD

Address: [addr_test1wpgqq4ky6wytas59wnrcgw8e3kcj5mar8fu5qunfy5endv7grf842k/tx#data](#)

Address Detail > TXs

Address: [addr_test1wpgqq4ky6wytas59wnrcgw8e3kcj5mar8fu5qunfy5endv7grf842k](#)

[Detail](#) [TXs](#) [Assets](#) [UTXO](#)

7s ago # e090420724_e720f Total: 557.84A

TX ID	Address	Amount
42779A	addr_test1_gdvn7	427.37A
13097A	addr_test1_f842k	117.42A
1305A	addr_test1_gjhfg	13.05A

14s ago # 704b105061_054b9 Total: 203.43A

TX ID	Address	Amount
20359A	addr_test1_pmr54	130.97A
7248A	addr_test1_pmr54	72.48A

트랜잭션의 메타데이터도 온체인에 기록되었습니다.

Google Chrome

Jan 27 14:01

Metadata Transaction e0...

preprod.cexplorer.io/tx/e0904207245ed7b3c1e8ee7d23b28bdb9667e6635ca096309b0a8378663e720f/metadata#data

Cexplorer.io

preprod

Dashboard

Watchlist

Pools

Assets

Blocks

Transactions

dApps

Metadata

More

EDUCATION

Articles

Videos

Wiki

ANALYTICS

Decentralization

Network

your button

Hi, SleepingCharles

19 Assurance

0.71kB (0.71kB) Transaction Size

0.71kB (1) Block Size (TX Count)

Minted by [BGR] BURGER STAKE POOL

Date

6m13s ago

Epoch

48

Block

564,954

Slot

19,162,521 (epoch slot 68,121)

Total Output

557.84 ₳ \$ 212.4 B 0.00922

Fee

0.23 ₳ \$ 0.1

Outputs

3

Content

Metadata (1)

Label	Data
	<pre>{ "order_detail": { "date": "2023/01/27-18:55:10", "version": "0.1", "order_id": "6247526440999", "ada_usd_price": "0.30324", "donation_split": "10%", "order_ada_amount": "130470000", "donation_ada_amount": "13047000" }}</pre>

문제 해결

다음 섹션에서는 **Cardano** 스마트 계약을 개발하는 동안 발생할 수 있는 문제를 디버그하는 데 도움이 되는 팁과 트릭에 대해 설명합니다.

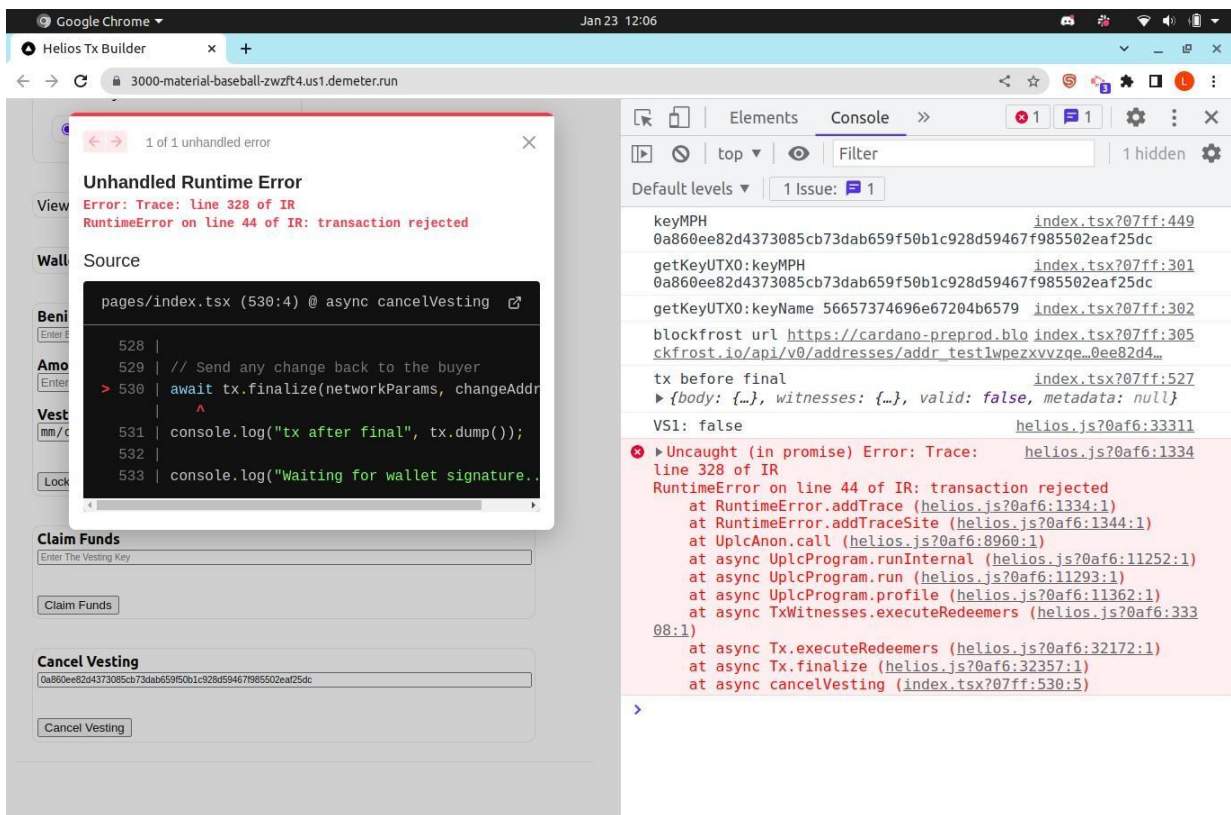
Trace()

.trace() 함수는 Helios 코드에 추적(trace)을 추가하기 위해 사용됩니다. .trace() 함수는 사실 .print() 함수의 래퍼(wrapper)이며, 편의를 위해 사용됩니다.

예제:

```
// Check if deadline hasn't passed
(now < datum.deadline).trace("VS1: ")
```

그러면 추적(trace) 내용은 plutus 스크립트가 실패한 콘솔 로그에서 확인할 수 있습니다.



Show()

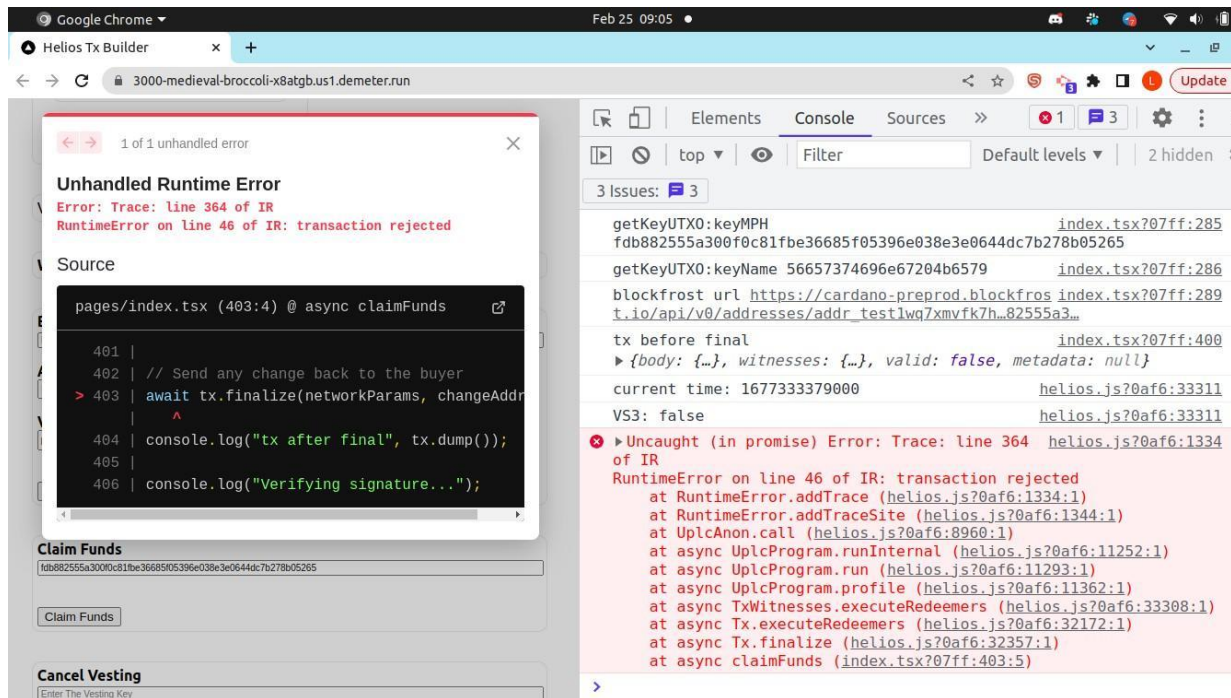
디버깅 시 더 많은 정보를 추가하려면, 기본 유형에 `.show()`를 추가하고 `.trace()` 또는 `.print()` 문에 포함시킬 수 있습니다. 예를 들어:

```
func main(datum: Datum, redeemer: Redeemer, context: ScriptContext) -> Bool {
    tx: Tx = context.tx;
    now: Time = tx.time_range.start;
    redeemer.switch {
    Cancel => {
    // Check if deadline hasn't passed
    (now < datum.deadline).trace("now: " + now.show() + " VS1: ")
```

Print()

스마트 컨트랙트 트랜잭션이 실패할 때 실행 스택의 변수를 출력하는 데 `print()` 함수를 사용할 수도 있습니다. 이 변수는 브라우저 콘솔에 표시됩니다.:

```
func main(datum: Datum, redeemer: Redeemer, context: ScriptContext) -> Bool {
  tx: Tx = context.tx;
  now: Time = tx.time_range.start;
  print("current time: " + now.show());
  redeemer.switch {
  Cancel => {
    // Check if deadline hasn't passed
    (now < datum.deadline).trace("VS1: ")
  }
```



일반적인 오류 원인

다음은 일반적인 오류의 원인과 해결 방법의 목록입니다.

Issue	Fix
Plutus script exceeds cpu/mem units	Set optimize=true when calling <code>Program.new(script).compile(optimize)</code> 불필요한 로직이나 트랜잭션 피하기
Invalid reference UTXO	Make sure you update the global-export-variables file has the correct reference UTXO
Wrong script address	버전 충돌을 방지하기 위해 <code>src</code> 디렉토리에 있는 <code>helio.js</code> 와 <code>helios</code> npm 모듈의 버전이 동일한지 확인하세요.
Collateral not set	지갑에 스마트 계약을 실행하는 데 필요한 담보가 설정되어 있는지 확인합니다.
Insufficient funds	지갑에 거래에 필요한 충분한 자금이 있는지, 올바른 네트워크에 연결되어 있는지 확인합니다.
Wallet account not set correctly	Nami 지갑의 경우, 계정 중에서 선택할 때 현재 선택한 계정이 활성 계정인지 확인하세요. 또한 브라우저 페이지를 새로 고쳐야 할 수도 있습니다.
Bash shell environment variables not set	<code>source ~/.bashrc</code> 명령어를 실행하여 환경 변수를 현재 셸에 로드해야 합니다.

Cardano-cli

plutus 스크립트가 사용할 CPU와 메모리 유닛의 양을 측정하려면, cardano-cli의 경우, bash 셸 스크립트의 출력 줄을 다음과 같이 바꾸기만 하면 됩니다.

```
--calculate-plutus-script-cost "$BASE/scripts/$ENV/data/mint-tx.costs"
```

그래서 스크립트의 끝 부분은 다음과 같습니다:

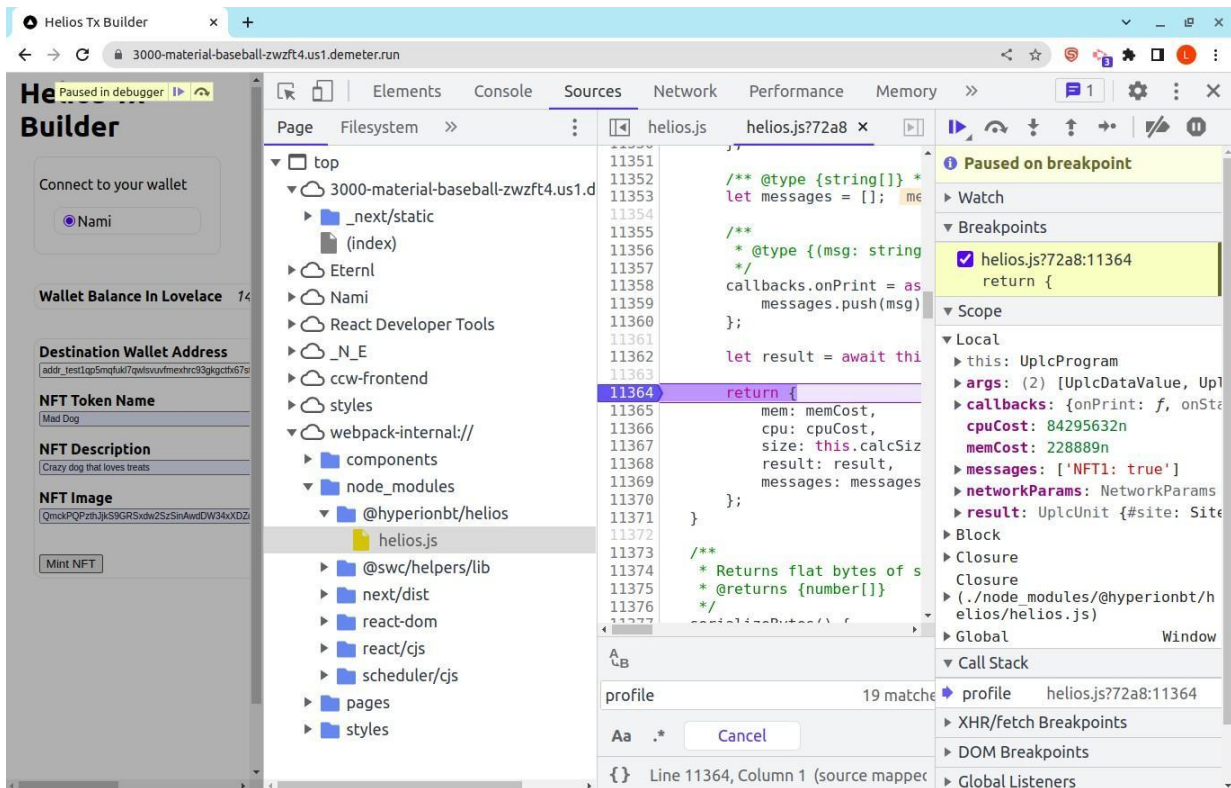
```
...
# Step 3: Build and submit the transaction
cardano-cli transaction build \
--babbage-era \
--cardano-mode \
$network \
--change-address "$admin_utxo_addr" \
--tx-in-collateral "$admin_utxo_collateral_in" \
--tx-in "$admin_utxo_in" \
--tx-in "$event_utxo" \
--spending-tx-in-reference "$VAL_REF_SCRIPT" \
--spending-plutus-script-v2 \
--spending-reference-tx-in-inline-datum-present \
--spending-reference-tx-in-redeemer-file "$redeemer_file_path" \
--tx-out "$MERCHANT_ADDR+$merchant_ada" \
--tx-out "$DONOR_ADDR+$donor_ada" \
--required-signer-hash "$admin_pkh" \
--protocol-params-file "$WORK/pparms.json" \
--metadata-json-file "$metadata_file_path" \
--calculate-plutus-script-cost "$BASE/scripts/$ENV/data/spend-tx.costs"
```

결과는 spend-tx.costs 파일에 생성됩니다.

```
$ more spend-tx.costs
[
  {
    "executionUnits": {
      "memory": 417629,
      "steps": 153838242
    },
    "lovelaceCost": 35189,
    "scriptHash": "4734d1f1927b26089874c6b3b369c860d450189f6c1844352623ede6"
  }
]
```

Helios Tx Builder

1. Start up the Next.js application and fill in input fields
2. Right click to inspect the page
3. Select sources -> Webpack internals -> @hyperionbt/helios
4. In the helio.js file search for the word profile
5. At the line with return for the profile function, set a breakpoint
6. Then select Mint in this case
7. The cpuCost and memCost are displayed in Scope -> Local as cpuCost and memCost variables.



Network Parameters

현재 [비용 모델 파라미터](#) 및 네트워크에서의 트랜잭션 최대 스크립트 크기, CPU 및 메모리 단위를 확인하려면 다음 명령을 실행하면 됩니다:

```
cardano-cli query protocol-parameters --testnet-magic 1
```

출력 결과에서 다음 값들을 확인하세요.

```
...
"maxTxExecutionUnits": {
  "memory": 14000000,
  "steps": 10000000000
},
"maxTxSize": 16384,
...
```

Baseline Comparison

하스켈 PlutusTx V2로 작성된 plutus 유효성 검사기 스크립트와 Helios로 다시 작성한 것을 비교한 내용입니다.

Metric	Haskell Plutus V2	Helios Plutus V2	Reduction
executionUnits Memory	1,899,564	805,465	~42.4 %
executionUnits Steps	528,441,393	297,796,866	~56.4 %
lovelaceCost	147,706	67,947	~46.0 %
Script size	8,125 KB	4,106 KB	~50.5 %

Helios Plutus V2 baseline testing log:

<https://github.com/lley154/littercoin/blob/preprod-5.0/testing/baseline.log>

Haskell Plutus V2 baseline testing log:

<https://github.com/lley154/littercoin/blob/baseline/testing/baseline.log>

CBOR.me

고급 문제 해결을 위해 블록체인 상의 트랜잭션을 검사해야 할 수도 있습니다. 이것은 CBOR 형식으로 저장되며 다음 웹 도구(<https://cbor.me/>)를 사용하여 디코딩할 수 있습니다.

cborHex를 복사하고 cbor.me에 붙여넣고 "as text" 옵션을 선택하십시오. 예를 들어:

```
$ cat work/spend-tx-alonzo.tx
{
  "type": "Witnessed Tx BabbageEra",
  "description": "Ledger Cddl Format",
  "cborHex": "84aa00828...63302e31"
}
```

The screenshot shows the CBOR playground interface. The URL is cbor.me/bytes=. The page title is "CBOR playground. See RFC 8949 for the CBOR specification, and cbor.io for more background information." The main heading is "CBOR". Below it, there are options: "Diagnostic" (selected), "plain hex", "0 Bytes", "as text" (selected), "utf8", "emb cbor", and "cborseq". There is a button "enter hex below Or Choose File" and a status "No file chosen". The "Diagnostic notation" shows "[1, [2, 3]]". The main area displays a large hex string representing the CBOR data.

Diagnostic notation: [1, [2, 3]]

84aa00828258201e75fabdd062ac9eb670a5bfb5adcc7cac0e8ce2789f3d907d769c8d69f65eb02825820bc730168d07eec6f11385242046978c02fbf1b94fbeb0ec408ac0da6d2a399c53000d81825820e06d00315b692ac604cbd2105019dc7d524d3e500d6a0b8c0258ce38699978d60012818258206af1c7ea85bd3222be2e31e5fcd664318b0897b2ab5b68a4db667508aa21a5f8000183a200581d603d62bdfdf66855d150b6cf97e4509ef78f5ea6245f642adf7629338c011a071f5188a200581d60b2b0a5ceaf7bc9a56fe619819b8891e6bafeff5c2cb275e333f97a9f011a00ca9748a200581d60b9abcf6867519e28042048aa11207214a52e6d5d3288b752d1c27682011a1074f50510a200581d60b9abcf6867519e28042048aa11207214a52e6d5d3288b752d1c27682011a0046fd0f111a00054e31021a000389760e81581cb9abcf6867519e28042048aa11207214a52e6d5d3288b752d1c276820b5820c1d4ececde03173d11592082528d907393fa67815f59d291712adaab8a471353075820b0f31849f135a4b646a0356561d93bb42a167f8618dc56682debcdd5ae46c4750a200818258207afdf742e52fc46d9a6f2183204c3d5ca3533450b2674f61adff33172d6d9ad65840fcd063509755eed8b17d206d921e0460ee82e4ad35c0166a26383dcfb1d7e04806adbb151081c554c26b9c7f5383e71886d94e1bab1e1ec69dcf72eea8e4c040581840001d87980821a00065f5d1a092b62a2f5d90103a100a101a16cf726465725f64657461696ca76d6164615f7573645f707269636567302e3337363630646461746573323032332f30312f32342d31333a34353a34373646f6e6174696f6e5f6164615f616d6f756e746831333237373030306e646f6e6174696f6e5f73706c697463313025706f726465725f6164615f616d6f756e7469313332373730303030686f726465725f69646d35323435343633313334343736776657273696f6e63302e31

그런 다음, 왼쪽에 트랜잭션 세부 정보를 볼 수 있는 작은 녹색 화살표를 선택하십시오.

CBOR playground

cbor.me/?bytes=

CBOR playground. See [RFC 8949](https://rfc8949) for the CBOR specification, and cbor.io for more background information.

CBOR

Diagnostic ☒ plain hex

728 Bytes ☒ as text ☐ utf8 ☐ emb cbor ☐ cborseq

enter hex below or

```
[{0:
  [[h'1E75FABDD062AC9EB670A5BFB5ADCC7CAC0E8CE2789F3D907D769C8D69F6
  5EB', 2],
  [h'BC730168D07EEC6F11385242046978C02F8F1B94FBE0EC408AC0DA6D2A309C
  53', 0]], 13:
  [[h'E06D00315B692AC604CBD2105019DC7D524D3E500D6A0B8C0258CE3869997
  8D6', 0]], 18:
  [[h'6AF1C7EA85BD3222BE2E31E5FCD664318B0897B2AB5B68A4DB667508AA21A
  5F8', 0]], 1: [{0:
    h'603D62BDFDF66855D150B6CF97E4509EF78F5EA6245F642ADF7629338C', 1:
    119493000}, {0:
    h'60B2B0A5CEAF7BC9A56FE619819B8891E6BAFEFF5C2CB275E333F97A9F', 1:
    13277000}, {0:
    h'60B9ABCF6867519E28042048AA11207214A52E6D5D3288B752D1C27682', 1:
    276100357}], 16: {0:
    h'60B9ABCF6867519E28042048AA11207214A52E6D5D3288B752D1C27682', 1:
    4652303}, 17: 347697, 2: 231798, 14:
    [h'B9ABCF6867519E28042048AA11207214A52E6D5D3288B752D1C27682'],
    11:
    h'C1D4ECECDE03173D11592082528D907393FA67815F59D291712ADAAB8A47135
    3', 7:
    h'B0F31849F135A4B646A0356561D93BB42A167F8618DC56682DEBCD5AE46C475
    0'}, {0:
    [[h'7AFDF742E52FC46D9A6F2183204C3D5CA3533450B2674F61ADFF33172D6D9
    AD6',
    h'FDC063509755EED8B17D206D921E0460EE82E4AD35C0166A26383DCFB1D7E0
    4806ADBB151081C554C26B9C7F5383E71886D94E1BAB1E1EC69DCF72EEA8E4C04
    ']], 5: [[0, 1, 121([[]], [417629, 153838242]]], true, 259({0:
    {1: {"order_detail": {"ada_usd_price": "0.37660", "date":
    "2023/01/24-13:45:48", "donation_ada_amount": "13277000",
```

```
84 AA # array(4)
00 # map(10)
82 # unsigned(0)
82 # array(2)
58 20 # array(2)
# bytes(32)
1E75FABDD062AC9EB670A5BFB5ADCC7CAC0E8CE2789F3D907D769C8D69F65EB
#
"\u001Eu\xFA\xBB\xDD\u0006*\xC9\xEBg\n[\xFBZ\xDC\xC7\xCA\xC0\xE8
\xCE'\x89\xF3\xD9\xa\xD7i\xC8e\xEB"
02 # unsigned(2)
82 # array(2)
58 20 # bytes(32)
BC730168D07EEC6F11385242046978C02F8F1B94FBE0EC408AC0DA6D2A309C53
#
"\xBCs\u0001h\xD0~\xECo\u00118RB\u0004ix\C0/\xBF\xe\x94\xFB\xE0\
xEC@\x8A\xC0\xDAm*\0\x9CS"
00 # unsigned(0)
00 # unsigned(13)
81 # array(1)
82 # array(2)
58 20 # bytes(32)
E06D00315B692AC604CBD2105019DC7D524D3E500D6A0B8C0258CE38699978D6
#
"\xE0m\u00001[i*\xC6\u0004\xCB\xD2\u0010P\u0019\xDC]RM>P\rj\v\x8
Cu0002X\xCE8i\x99x\xD6"
00 # unsigned(0)
```

현재 [Babbage 시대](#)에 우리는 CDDL 스펙을 [여기](#)서 확인할 수 있습니다.

프로덕션

마지막 섹션에서는 스마트 컨트랙트를 프로덕션에 배포할 때 고려해야 할 몇 가지 사항을 살펴보겠습니다.

법적 고지

MIT License

Copyright (c) 2023 Context Solutions Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

감사

고객 자금이 관련된 모든 프로젝트에서는 스마트 컨트랙트 개발을 검토할 [독립적인 감사인](#)을 고용할 것을 강력히 권장합니다. 또한 **Helios**는 매우 새로운 언어이며 현재 하스켈 **PlutusTx**와 동일한 수준의 감사 및 프로덕션 검증이 이루어지지 않았다는 점에 유의하시기 바랍니다. 하지만 더 많은 개발자와 프로젝트 팀이 **Helios** 사용의 이점을 활용함에 따라 이러한 위험은 시간이 지남에 따라 감소할 것입니다.

Test, Test, Test

스마트 계약을 배포하면 롤백하거나 되돌릴 수 없습니다. 이것이 블록체인이 불변해야 하는 이유이므로, 개발자는 블록체인 애플리케이션을 구축할 때 이 사실을 인식해야 합니다.

메인넷에서 예기치 않은 일이 발생하는 위험을 줄이기 위해, 테스트는 개발 라이프 사이클의 중요한 부분이어야 합니다. **Helios**는 [Fuzzy Test](#)라는 단위 테스트 프레임워크를 제공하며, 이는 통합, 기능 및 사용자 인증 테스트와 함께 사용할 수 있습니다.

모든 코드 예제는 [테스팅 폴더](#)에 문서화된 통합 및 기능 테스트 결과와 함께 완료되었습니다. 기억하세요. 긍정적인 시나리오와 부정적인 시나리오 모두를 테스트하는 것이 중요합니다.

스마트 컨트랙트 생명 주기

프로덕션에 배포하기 전에 스마트 컨트랙트 생명 주기를 이해하는 것이 중요합니다.

- 업그레이드 경로는 어떻게 생겼으며, 업그레이드가 가능한가요?
- 프론트엔드 애플리케이션이 새로운 스마트 컨트랙트를 가리킬 수 있나요, 아니면 최종 사용자와의 조율이 필요한가요?
- 메인넷 출시 후 스마트 컨트랙트에서 확인된 문제를 어떻게 해결하나요?
- 스마트 컨트랙트를 어떻게 폐기할 것인가요?
- 스마트 컨트랙트 기능에 영향을 줄 수 있는 **plutus** 코어 평가자의 변경 사항이 있는지 [Cardano CIP](#)를 모니터링합니다.

Sources and Acknowledgements

Open Source

The complete source code for all examples in this book are open source and located at <https://github.com/lley154/helios-examples>. There may be minor differences between the code samples in this book and the repository and this is mainly due to formatting to fit the page layout and for readability.

Without open source projects, this book and countless projects in our ecosystem would not have been possible. Please consider donating to one of your favorite open source projects so they can continue to build great things!

References

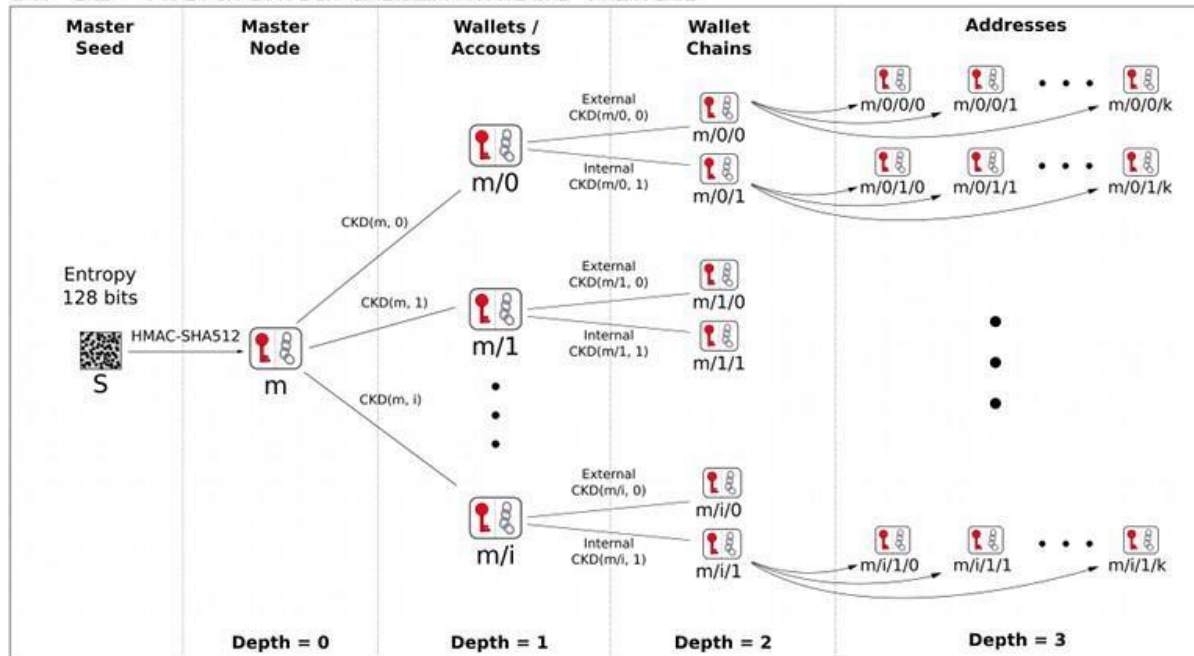
- <https://github.com/Hyperion-BT/helios>
- <https://www.hyperion-bt.org/helios-book/>
- <https://txpipe.github.io/oura/>
- <https://github.com/berry-pool/nami>
- <https://developers.cardano.org/>
- <https://docs.cardano.org/plutus/plutus-resources>
- <https://github.com/input-output-hk/cardano-addresses>
- <https://github.com/input-output-hk/cardano-node>
- <https://github.com/dendorferpatrick/nami-wallet-examples/blob/master/Multi-Signature.md#4-hash-metadata-for-transaction>
- <https://www.youtube.com/@AndrewWestberg/videos>
- <https://www.essentialcardano.io/>
- <https://github.com/input-output-hk/plutus-pioneer-program>
- <https://medium.com/@blainemalone01/hd-wallets-why-hardened-derivation-matters-89efcdc71671>

부록

주소 키 파생

키 파생은 복잡한 주제이며, 그 기본 개념은 개인 키와 공개 키를 사용하여 다양한 용도로 사용되는 여러 유형의 키를 파생하는 것입니다. 블록체인 암호화에서 개인 키의 시작점은 **BIP-39**로 인코딩된 시드 구문입니다. 다음 다이어그램¹은 키 도출과 관련된 단계를 보여줍니다.

BIP 32 - Hierarchical Deterministic Wallets



Child Key Derivation Function ~ $CKD(x,n) = \text{HMAC-SHA512}(x_{\text{Chain}}, x_{\text{PubKey}} || n)$

¹ Source: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>

다음은 새로 생성한 암호 구문에서 **cardano-cli**와 함께 사용할 수 있는 개인 키를 얻는 단계입니다.

1. Go to your Web VS Code in your browser
2. Select the hamburger menu (top left) and Terminal -> New Terminal
3. `mkdir ~/.local/keys`
4. `cd ~/workspace`
5. `wget https://github.com/input-output-hk/cardano-wallet/releases/download/v2022-12-14/cardano-wallet-v2022-12-14-linux64.tar.gz`
6. `tar -xvzf cardano-wallet-v2022-12-14-linux64.tar.gz`
7. `cd cardano-wallet-v2022-12-14-linux64`
8. `./cardano-address recovery-phrase generate --size 24 > ~/.local/keys/key.prv`
9. `./cardano-address key from-recovery-phrase Shelley < ~/.local/keys/key.prv > ~/.local/keys/key.xprv`
10. `./cardano-address key child 1852H/1815H/0H/0/0 < ~/.local/keys/key.xprv > ~/.local/keys/key.xsk`
11. `./cardano-cli key convert-cardano-address-key --shelley-payment-key --signing-key-file ~/.local/keys/key.xsk --out-file ~/.local/keys/key.skey`
12. `./cardano-cli key verification-key --signing-key-file ~/.local/keys/key.skey --verification-key-file ~/.local/keys/key.vkey`
13. `./cardano-cli address key-hash --payment-verification-key-file ~/.local/keys/key.vkey --out-file ~/.local/keys/key.pkh`
14. `./cardano-cli address build --payment-verification-key-file ~/.local/keys/key.vkey --out-file ~/.local/keys/key.addr --testnet-magic 1`

그런 다음 **cardano-cli**에 사용할 수 있는 개인 및 공개 키와 함께 비밀번호로 생성된 주소를 확인할 수 있습니다.

```
$ more ~/.local/keys/key.addr  
addr_test1v83ynr979e4xpjj28922y4t3sh84d0n08juy58am7jxmp4g6cgxr4
```

Installing Next.js

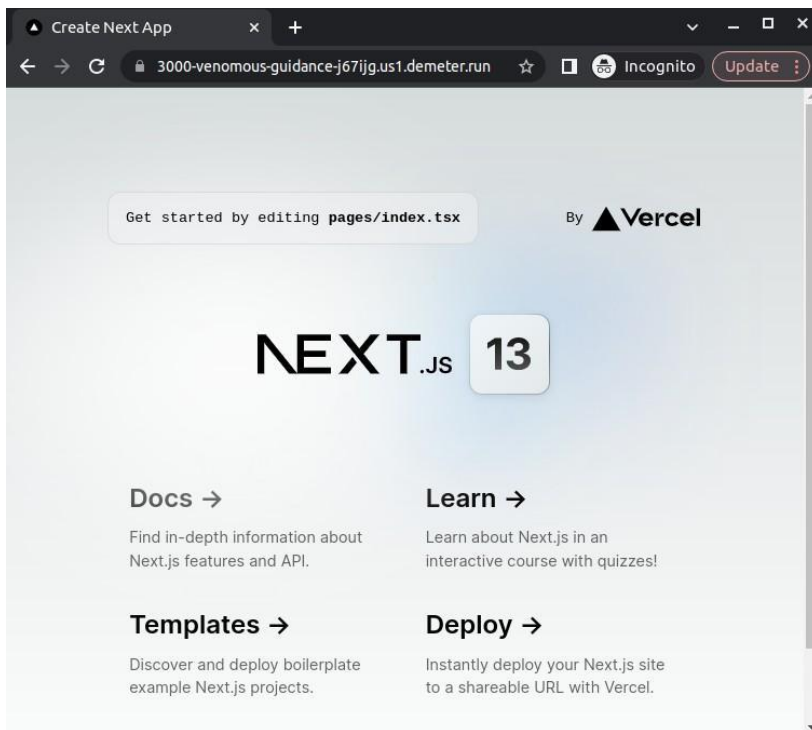
Go to your workspace using the terminal window in the Visual Studio Web.

```
$ npx create-next-app@latest --typescript .
Need to install the following packages:
create-next-app@13.1.2
Ok to proceed? (y) y
✓ Would you like to use ESLint with this project? ... Yes
✓ Would you like to use `src/` directory with this project? ... No
✓ Would you like to use experimental `app/` directory with this project? ... No
Creating a new Next.js app in /config/workspace/repo/helios-builder/app.
Using npm.
Installing dependencies:
- react
- react-dom
- next
- @next/font
- typescript
- @types/react
- @types/node
- @types/react-dom
- eslint
- eslint-config-next
added 271 packages, and audited 272 packages in 28s
102 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
Initializing project with template: default
Success! Created app at /config/workspace/repo/helios-builder/app
npm notice
npm notice New major version of npm available! 8.19.3 -> 9.3.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.3.0
npm notice Run npm install -g npm@9.3.0 to update!
npm notice
abc@venomous-guidance-j67ijg-0:~/workspace/repo/helios-builder/app$
```

Startup Next.js

```
$ npm run dev
```

Then select the URL on the Exposed Ports tab for your workstation to launch the application. You should see the default Next.js page like this.



To use Helios, you will need to install the Helios npm module

```
$ npm install @hyperionbt/helios
```

Also install the express-validator module to sanitize inputs when required.

```
$ npm install express-validator
```

You will also need to replace the next.config.js file with the following

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  webpack: function (config, options) {
    config.experiments = {
      asyncWebAssembly: true,
      topLevelAwait: true,
    };
    return config;
  },
};
module.exports = nextConfig;
```

IPFS (NFT Images)

IPFS(InterPlanetary File System)는 일반적으로 **NFT** 이미지 및 기타 미디어 파일을 저장하는 데 사용됩니다. **IPFS**는 업로드된 콘텐츠에 고유 키를 부여하여 액세스할 수 있도록 하는 **P2P** 파일 공유 네트워크입니다.

이미지를 **IPFS**에 올리는 데 도움을 줄 수 있는 서비스 제공업체가 많이 있습니다. 쉽고 무료인 옵션으로는 **thirdweb**이 있습니다.

<https://blog.thirdweb.com/guides/how-to-upload-and-pin-files-to-ipfs-using-storage/>

```
$ npx thirdweb@latest upload path/to/file.extension
```

Shopify Ada Payments

작성 당시 **Ada Shopify** 결제 플러그인이 없습니다. 다음 **Shopify** 통합은 기본 플랜을 가진 상인들과 체크아웃 페이지를 사용자 정의할 수 없는 경우에 사용할 수 있습니다. 더 높은 가격의 플랜을 가진 상인들은 프론트엔드 및 체크아웃 사용자 정의를 더 할 수 있으며 이는 **Ada**로 결제하는 프로세스를 간소화할 수 있습니다.

Shopify Configuration

We need to update the Shopify store so when an order is placed, there is a link to pay with Ada.

1. In Settings, select Apps and sales channels
2. Select Develop apps for your store
3. Select Allow custom app development
4. Select Allow customer app development again
5. Create an app
6. Enter the name of the app, eg. "Pay With Ada"
7. Select Create app
8. Select Configure Admin API scopes
9. Scroll down and enable in the Order section, write_orders and read_orders
10. Scroll down and enable in the Products section, read_products
11. Go to the bottom of the page and press Save
12. Now select the API credentials tab
13. In the Access tokens box, select Install app
14. A prompt will ask you if you want to install your app, select Install
15. Go back to the main settings menu and select Payments
16. Select Add manual payment method
17. In the dialog box, enter the name of the payment method. eg "Pay With Ada"
18. Add additional details if required. eg "Paying with Ada using your Cardano Wallet"
19. Add payment instructions. eg. "Please select to the Pay Now With Ada link below to pay using your Cardano Wallet"
20. Go back to the main settings menu and Select Checkout
21. Scroll to the bottom of the page and add the following to the Additional scripts window

```
<script>
  Shopify.Checkout.OrderStatus.addContentBox(
    '<h2 style="color:red;">Pay To Complete Your Order</h2>',
    '<a href="#" id="paynow">Pay Now In Ada</a>'
  );
  var urlStr =
"https://3000-venomous-decision-kd0ey1.us1.demeter.run/"
  ; var url = new URL(urlStr);
  var params = url.searchParams;
  var orderId = Shopify.checkout.order_id
  params.append("id", orderId);

  function updatePayNow () {
    document.getElementById("paynow").href=url
  }
</script>
```

```

function updateOrderNum () {
    document.querySelector("body > div > div > div > main >
div.step > div.step_sections > div:nth-child(1) > div > div >
span").innerHTML=orderId
}

document.addEventListener("DOMContentLoaded", function() {
    updatePayNow()
});
document.addEventListener("DOMContentLoaded", function() {
    updateOrderNum()
});
</script>

```

22. Select Save
23. In the Setting menu, select Apps and sales channels
24. Select Develop Apps
25. Select Pay With Ada App
26. Select Manage Credentials
27. Show the Access Token and copy it to your clipboard
28. Paste the Access Token into your ~/.bashrc file
for NEXT_PUBLIC_ACCESS_TOKEN value

About Helios



A DSL for writing smart contracts on the Cardano
blockchain. Twitter: [@helios_lang](https://twitter.com/helios_lang)

Github:

<https://github.com/Hyperion-BT/helios>

Discord: <http://discord.gg/XTwPrvB25q>

Web: <https://hyperion-bt.org>

About The Author



Lawrence Ley is passionate about learning new technologies and how they can make a positive impact. His Cardano journey started out in the Plutus pioneers program 2nd cohort and has worked on a number open source impact projects. He discovered Helios and was thoroughly impressed how easy it was to pick up due to its familiar language, syntax and structure. Since onboarding new developers to Cardano is not easy, he decided to write this book.

Twitter: [@lley154](https://twitter.com/lley154)

YouTube: <https://www.youtube.com/channel/UC5R8HrDUms8XqZVWszqbA7A>

GitHub: <https://github.com/lley154>