References of Flow Blockchain

1. How to Create an NFT (NFTを作るには)
2. Array, Dictionary, Struct
3. Resource
4. Access Control (アクセスコントロール)
5. Resource Interface (インターフェース)
6. Private Capabilities
7. How to build a Flow Dapp on Testnet (Flow Dapps開発の流れ)
8. How to Create a Fungible Token (Fungible Tokenの作り方)
9. Send a Transaction with a Non-Custodial Wallet (ノンカストディアルウォレットでトランザクションを送信するには)
10. Query for Events (イベントを取得するには)
11. Optional Binding
12. How to Multi-Sign (マルチサインを行うには)
13. Optional References
14. Configure Cadence VSCode Extension (VSCode Extensionセットアップ方法)
15. How to Build an NFT Marketplace (NFTのマーケットプレイスを作るには)
16. How to Verify Account Ownership (account-proof)
17. Account Storage Iteration (アカウントが所有するNFTを一覧で取得するには)

最終改訂日: 2022年12月03日

1. How to Create an NFT (NFTを作るには)

·To save NFTs directly to one's account storage, do the following
(アカウントのストレージにNFTを直接保存するには以下のようにします)

```
prepare(acct: AuthAccount) {
  acct.save(<-SampleCrypto.createNFT(), to: /storage/MySampleCrypto)
}
```

·NFTs can be referenced. It is used when you want to do something to the recipient's storage. Do the following
(NFTは参照することが出来ます。受取者のストレージに対して何か行いたい時に使用します。以下のようにします)

```
prepare(acct: AuthAccount) {
  let aReference = acct.borrow<&SampleCrypto.NFT>(from: /storage/MySampleCrypto)
    ?? panic("Nothing exists here! You don't have an NFT")
  log(aReference.id)
}
```

·Capability is used to store NFTs in the recipient's storage, and differs from the two above codes in that it stores a collection of NFTs.
(NFTを受取者のストレージに保存する為にはCapabilityを使います。2つ上のコードと異なる所はNFTのコレクションを保存している点です)

```
prepare(acct: AuthAccount) {
  acct.save(<-SampleCrypto.createCollection(), to: /storage/Collection)
  acct.link<&SampleCrypto.CollectionPublic>(/public/Collection, target: /storage/
Collection)
}
```

·To save an NFT to a recipient using Capability, do the following
(Capabilityを使用してrecipient(受取者)にNFTを保存するには以下のようにします)

```
prepare(acct: AuthAccount) {
  let nftMinter = acct.borrow<&SampleCrypto.NFTMinter>(from: /storage/Minter)
  let publicReference =
getAccount(recipient).borrow<&SampleCrypto.Collection{SampleCrypto.CollectionPublic}
>()
    ?? panic("This account does not have a Collection")
  publicReference.deposit(token <- nftMinter.createNFT())
}
```

·To transfer an NFT using Capability, do the following
(Capabilityを使用してNFTを転送するには以下のようにします)

```
prepare(acct: AuthAccount) {
  let collection = acct.borrow<&SampleCrypto.Collection>(from: /storage/Collection)!
  let publicReference =
getAccount(recipient).borrow<&SampleCrypto.Collection{SampleCrypto.CollectionPublic}
>(from: /storage/Collection)
    ?? panic("This account does not have a Collection")
  publicReference.deposit(token <- collection.withdraw(id: id))
}
```

·To downcast from the NFT standard to the NFT itself, do the following
(NFTスタンダードから、NFT本体にダウンキャストするには以下のように行います)

```
pub fun downcastSample(id: UInt64): &NFT {
  let refNFT = &self.ownedNFTs[id] as auth &NonFungibleToken.NFT
  return refNFT as! &NFT
}
```

·When downcasting, place an `auth` directive on the interface. Also, place an `as!` directive on the variable when performing a downcast.
(ダウンキャストを行う時はインターフェースに対して`auth`指示子を置きます。また、ダウンキャスト実行時は`as!`指示子を置きます。)

·Use getViews to return all information held by the NFT
(NFTが持つ全情報を返すにはgetViewsを使います)

```
pub fun getViews(): [Type] {
  return [
    Type<MetadataViews.Display>(),
    Type<MetadataViews.Royalties()
  ]
}
```

getViews is generic and allows you to retrieve NFTs from all over the world without importing contracts
(getViewsはジェネリックであり、世界中のNFTをコントラクトのインポートなしに取得することができます)

·Use resolveView to return each piece of information held by the NFT
(NFTが持つ各情報を返すにはresolveViewを使います)

```
pub fun resolveView(_ view: Type): AnyStruct? {
  switch view {
    case Type<MetadataViews.Display>():
     return MetadataViews.Display(
       name: self.name,
       description: self.description,
       thumbnail: MetadataViews.HTTPFile(url: self.thumbnail)
     }
    case Type<MetadataViews.Royalties>():
     return MetadataViews.Royalties(
       self.royalties
     }
  }
  return nil
}
```

An example of obtaining NFT information is as follows
(NFTの情報を取得する一例は以下です)

```
pub fun getMetadata(address: Address, id: UInt64): NFTResult {
  let collection = getAccount(address)
    .getCapability(ExampleNFT.CollectionPublicPath)
    .borrow<&{MetadataViews.ResolverCollection}>()
    ?? (could not borrow.)

  let nft = collection.borrowViewResolver(id: id)

  var data = NFTResult()

  if let view = nft.resolveView(Type<MetadataViewsDisplay>()) {
    let display = view as! MetadataViews.Display
    data.name = display.name
    data.description = display.description
  }

  return data
}
```

For more information on the NFT Standard, please watch the community video below (NFT Standardの詳細は下記コミュニティビデオをご覧ください)

Community's video: https://lnkd.in/gPCgWx2N & https://www.youtube.com/watch?v=bINELogSODk
& https://www.youtube.com/watch?v=rgJRTozG3cw

2. Array, Dictionary, Struct

How to declare an Array (Arrayの宣言の仕方)

```
pub contract ADS {
  pub var allNames: [String]
  init() {
    self.allNames = []
  }
}
```

If you don't want to define a type, (型を定めたくない場合は)

```
  pub var allNames: [AnyStruct]
```

AnyStruct can contain all types. (とすることもできます。AnyStructには全ての型を入れることができます。)

To insert values, do the following (値を入れるには以下のようにします)

```
  self.allNames.append("Taro")
```

To combine arrays, do the following (配列を結合するには以下のようにします)

```
  self.allNames.concat(allNames2)
```

To put a value in a specified position, do the following (値を指定の位置に入れるには以下のようにします)

```
  self.allNames.insert(at: 1, "Jiro")
```

How to declare a Dictionary (Dictionaryの宣言の仕方)

```
  pub var luckeyNumbers: {String: Int}
```

To insert values, do the following (値を入れるには以下のようにします)

```
  self.luckeyNumbers["Taro"] = 7
```

Before doing this, if you want to generate an error for a transaction if "Taro" does not exist in allNames, you can do so by using the pre
(これを実施する前に、allNamesに”Taro"が存在しない場合にトランザクションに対しエラーを出したい場合、preを使用することで可能です)

```
pre {
  self.allNames.contains("Taro"): "This is not in allNames"
}
```

To get all the Dictionary keys, do the following (Dictionaryのキーを全て取得するには以下のようにします)

```
  self.luckeyNumbers.keys // returns an array of all the keys
```

To get all Dictionary values, do the following (Dictionaryの値を全て取得するには以下のようにします)

```
  self.luckeyNumbers.values // returns an array of allthe values
```

To remove a value, do the following (値を取り除くには以下のようにします)

```
  self.luckeyNumbers.remove(key: "Taro")
```

How to declare a Struct (Structの宣言の仕方)

```
pub struct Identity {
  pub let luckeyNumber: Int
  pub let favoriteFruit: String
  init (_luckeyNumber: Int, _favoriteFruit: String) {
    self.luckeyNumber = _luckeyNumber
    self.favoriteFruit = _favoriteFruit
  }
}
```

To initialize a Struct and put it into a Dictionary, do the following
(Structを初期化してDictionaryに入れるには以下のようにします)

```
pub fun addIdentities(name: String, luckeyNumber: Int, favoriteFruit: String) {
  self.identities[name] = Identity(_luckeyNumber: luckeyNumber, _favoriteFruit:
favoriteFruit)
```

```
}
```

To give a method to a Struct, do the following (Structにメソッドを持たせるには以下のようにします)

```
pub fun changeFavoriteFruit(newFruit: String) {
  self.favoriteFruit = newFruit
}
```

To call a Struct method from within a transaction, do the following (トランザクション内からStructのメソッドを呼び出すには以下のようにします)

```
  ADS.identities[name]!.changeFavoriteFruit(newFruit: favoriteFruit)
```

[NOTE] ([注意])
Since blockchain values are basically Public, when storing values that you do not want to make public, such as user information, it is recommended to encrypt them once in the backend, and then return the encrypted values and store them in Struct in a transaction.
(ブロックチェーンの値は基本的にPublicであるため、ユーザー情報など公にしたくない値を格納する時は一度バックエンドで暗号化し、その暗号化された値を返してトランザクションでStructに格納することをお勧めします)

Community's video: https://lnkd.in/gTp4tiVj

3. Resource

How to declare a Resource (Resourceを宣言の仕方)

```
pub resource Hello {
  pub let greeting: String
  init(_greeting: String) {
    self.greeting = _greeting
  }
}
```

To create a Resource, do the following
(Resourceを作成するには以下のようにします)

```
  var testResource: @Hello <- create Hello(_greeting: greeting)
  return <- testResource
```

`create` keyword cannot be used outside of Contract (`create`キーワードはContract外では使えません)

To declare a collection of Resource, do the following (Resourceのcollectionを宣言するには以下のようにします)

```
pub resource HelloBucket {
  pub var hellos: @{UInt64: Hello}
  init() {
    self.hellos <- {}
  }
  destroy() {
    destroy self.hellos
  }
}
```

·Resource always needs to be with <- or destroy, otherwise an error will occur. Destroying a collection of Resource can be done as above to destroy all Resource in the collection, otherwise an error will occur.
(Resourceは常に<-かdestroyで行き先を指示しないとエラーになります。ResourceのcollectionのdestroyはResourceをdestroyすることが出来、これがないとエラーとなります。)

Here is an example of putting a Resource into a collection of Resource
(ResourceのcollectionにResourceを入れる一例は以下です)

```
pub fun depositHello(hello: @Hello) {
  self.hellos[hello.uuid] <-! hello
}
```

·`<-! ` will cause panic if the resource already exists
(`<-!`はもし既にリソースが存在する場合はpanicが発生します)

·`.values` cannot be used in Resource's Dictionary because it would be meaningless if the Resource itself could be moved.

(ResourceのDictionaryでは`.values`が使用できません。Resource自体の移動ができてしまっては意味がないためです。)

Community's video: https://lnkd.in/gwUNUGrx

4. Access Control (アクセスコントロール)

Access Control for variables is as follows (変数のAccess Controlは以下です)

pub(set): Read Scope: All, Write Scope: All
pub/access(all): Read Scope: All, Write Scope: Current and inner
access(contract): Read Scope: Current, inner, and containing contract (変数を含有するコントラクト内からアクセスできます), Write Scope: Current and inner
access(self): Read Scope: Current and inner, Write Scope: Current and inner

Access Control for the function is as follows (functionのAccess Controlは以下です)

pub/access(all): ALL (どこからでも呼べます)
access(contract): Current, inner, and containing contract (メソッドを含有するコントラクト内からアクセスできます)
access(self): Current and inner

·`access(contract)` will not be able to access from the transaction
(`access(contract)`にするとトランザクションからアクセスすることが出来なくなります)

Community's video: https://lnkd.in/gbd2qRCp

5. Resource Interface (インターフェース)

·An interface describes behavior that must be implemented like a Java interface. It is implemented as follows
(インターフェースはJavaのインターフェースのように実装しなければならない動作を記述します。以下のように実装します。)

```
pub resource interface IGreeting {
   access(contract) let greeting: String
}
```

·Interface can be used to efficiently restrict access
(インターフェースを使用すると効率的にアクセス制限ができます)

```
pub fun createRestrictedGreeting(): @Greeting{IGreeting} {
   return <- create Greeting()
}
```

·Resource returned by this method will only be accessible to methods and variables declared in the IGreeting interface, not to variables and methods declared in the Greeting resource
(このメソッドで返されたリソースはIGreetingインターフェースで宣言されたメソッドや変数のみがアクセス可能となり、Greetingリソース内で宣言されIGreetingインターフェースにはない変数やメソッドにはアクセスができなくなります)

·If the variable is set to `access(contract)`, the value of the variable can only be returned by the method, making the access control more visually obvious
(変数を`access(contract)`としておくと、変数の値がメソッドからしか返せない為、アクセスコントロールがより視覚的に分かりやすくなります)

```
pub resource Greeting: IGreeting {
   pub let greeting: String
   init() {
      self.greeting = "Hello, World!"
   }
}
```

·It is also possible to attach an access control different from the interface to a variable (greeting) in the Resource. In this case, when creating a resource, if there is an Interface as @Greeting{IGreeting}, the access permission is for access (contract), and if there is no Interface as @Greeting, the access permission is for pub.
(インターフェースと異なるアクセスコントロールをResource内で変数(greeting)につけることもできます。この場合、リソースをcreateする時、@Greeting{IGreeting}としてInterfaceがある場合access(contract)のアクセス権限となり、@GreetingとしてInterfaceが無い場合pubのアクセス権限となります。)

Please watch the community video below for more information on this
(詳細は下記コミュニティビデオをご覧ください)

Community's video: https://lnkd.in/grDRrKHA

6. Private Capabilities

To create a Contract that allows the privileges held by the Contract holder to pass to another account, do the following
(Contractの保持者が持つ特権を他のアカウントに渡すことができるContractを作成するには以下のようにします)

```
pub resource Admin {
  pub let profile: Capabilities<&Profile>
  init(_profile: Capability<&Profile>),
    self.profile = _profile
  ]
}
pub fun createAdmin(profile: Capabilities<&Profile>): @Admin {
  return <- create Admin(_profile: profile)
}
```

Here we are passing the Private access rights to the Profile resource to another account (ここではProfileリソースのPrivateなアクセス権限を他のアカウントに渡しています)

To create Private Capabilities that allow the privileges held by the Contract holder to be passed on to other accounts, do the following
(Contractの保持者が持つ特権を他のアカウントに渡すことができるPrivate Capabilitiesを作成するには以下のようにします)

```
prepare(acct: AuthAccount) {
  acct.save(<- Example.createProfile(name: "Taro", email: "abc@example.com"), to: /storage/Profile)
  acct.link<&Example.Profile{Example.ProfilePublic}>(/public/ProfilePublic, target: /storage/Profile)
  acct.link<&Example.Profile>(/private/ProfilePrivate, target: /storage/Profile)
  let privateCapability = acct.getCapability<&Example.Profile>(/private/ProfilePrivate)
  acct.save(<- Example.createAdmin(profile: privateCapability), to: /storage/Admin)
}
```

At the very end, we are storing Capability. Here it is stored in the same account, but it can be saved in other accounts.

(一番最後のところでCapabilityを保存しています。ここでは同一アカウント内に保存していますが、他のアカウントに保存することが出来ます)

To use the privilege with the passed Capability, do the following (渡されたCapabilityを使用して特権を利用するには以下のようにします)

```
prepare(acct: AuthAccount) {
  let admin = acct.borrow<&Example.Admin>(from: /storage/Admin) ?? panic("Could not borrow the Admin.")
  log(admin.profile.borrow()!.name)
}
```

·By putting the Capability that another account has in a variable of a resource, you can store that resource and exercise its authority by borrowing the Capability in that variable.
(他のアカウントが持つCapabilityをリソースの変数に入れることで、そのリソースを保存し、その変数のcapabilityをborrowすることでその権限を行使できます。)

Community's video: https://lnkd.in/gTwGEZ_g

7. How to build a Flow Dapp on Testnet

An example of a Flow Dapp development is below (Flow Dapp開発の一例は以下です)

a. Install Node.js.
b. Install Flow CLI.
c. Create Contract template on Play Ground.
(Play Ground上でContractの雛形を作成)
d. Deploy Contract to testnet at flow-view-source.com.
(flow-view-source.comでContractをテストネットにデプロイ)
e. Create a script to get Contract information on Play Ground.
(Play Ground上でContractの情報を取得するscriptを作成)
f. Open the editor, create a new cadence folder, create a scripts folder under it, create a js file under it, and paste the `e` information.
(エディタを開きcadenceフォルダを新規で作成し、その下にscriptsフォルダを作成、その下にjsファイルを作成し、eの情報をペーストする)
g. Install @onflow/fcl and @onflow/types with npm.
(npmで@onflow/fclと@onflow/typesをインストールする)

h. Import fcl and set it to connect to the test net. At this time, set the Address that is displayed in the lower left corner when logging in with view-source.
(fclをimportし、テストネットに接続するように設定する。この時、view-sourceでログインしたときに左下に表示されるAddressをセットする)

i. Execute the fcl.send method to the point where you can retrieve information about the contract deployed on the testnet.
(fcl.sendメソッドでテストネットにデプロイしたコントラクトの情報を取得できるところまで確認する)

j. Create a method to log in to the Flow wallet with the fcl.authenticate method
(fcl.authenticateメソッドでFlowのウォレットにログインできるメソッドを作成する)

k. Create a transaction script to change Contract information on Play Ground (Play Ground上でContractの情報を変更するtransactionスクリプトを作成)

l. Open the editor and create a transitions folder under the cadence folder, create a js file under it, and paste the `k` information
(エディタを開きcadenceフォルダの下にtransactionsフォルダを作成、その下にjsファイルを作成し、kの情報をペーストする)

m. Create a button that allows you to change the information of a contract deployed on the testnet using the fcl.send method
(fcl.sendメソッドでテストネットにデプロイしたコントラクトの情報を変更できるボタンを作成する)

n. Execute the transaction to get the transactionHash and check it with flowscan.
(トランザクションを実行するとtransactionHashが取得できるのでflowscanで確認する)

See the community video below for more details. a~n steps will take about 40 minutes!
(詳細は下記コミュニティビデオをご覧ください。a~nまでの手順が40分ほどで終わります)

Community's video: https://lnkd.in/g-Sh-rFQ

Besides React, Nuxt.js can also implement Flow Dapp by using the following libraries
(React以外にNuxt.jsも以下のライブラリを使用することでFlow Dappを実装できます)

https://github.com/brunogonzales/fcl-nuxt-starter

8. How to Create a Fungible Token

·How to create a Fungible Token (Fungible Tokenの作り方)

1). The Fungible Token Standard interface can be found below.
(Fungible Token Standardのinterfaceは以下にあります。)

https://github.com/onflow/flow-ft/blob/master/contracts/FungibleToken.cdc

2). Create a Contract that imports this Interface
(このInterfaceをimportしたContractを作ります)

3). Implement totalSupply, deposit, withdraw, Minter
(totalSupply, deposit, withdraw, Minterを実装します)

·The deposit function should be as follows (depositは以下のようにします)

```
pub fun deposit(from: @FungibleToken.vault) {
  let vault <- from as! @MyToken.vault
  emit TokensDeposited(amount: vault.balance, to: self.owner?.address)
  self.balance = self.balance + vault.balance
  vault.balance = 0.0
  destroy vault
}
```

·The withdraw function should be as follows (withdrawは以下のようにします)

```
pub fun withdraw(amount: UFix64): @FungibleToken.vault {
  self.balance = self.balance - amount
  emit TokensWithdrawn(amount: amount, from: self.owner?.address)
  return <- create Vault(balance: amount)
}
```

·The Minter resource should be as follows (Minterは以下のようにします)

```
pub resource Minter {
  pub fun mintToken(amount: UFix64): @FungibleToken.vault {
    MyToken.totalSupply = MyToken.totalSupply + amount
    return <- create Vault(balance: amount)
  }
  init() {
  }
}
init() {
  self.totalSupply = 0.0
  self.account.save(<- create Minter(), to: /storage/Minter)
  }
}
```

4). Save the valut of this Contract to the account. Do the following
(アカウントにこのContractのvalutを保存します。以下のようにします)

```
prepare(acct: AuthAccount) {
  acct.save(<- MyToken.createEmptyVault(), to: /storage/Vault)
  acct.link<@MyToken.vault{FungibleToken.Balance, FungibleToken.Receiver}>(/public/
Vault, target: /storage/Vault)
}
```

5). To issue fungible currencies and deposit it into a specific account, do the following
(独自通貨を発行して特定のアカウントに預け入れるには以下のようにします)

```
prepare(acct: AuthAccount) {
  let minter = acct.borrow<&MyToken.Minter>(from: /storage/Minter)
  let newVault <- minter.mintToken(amount: 20.0)
  let receiverVault = getAccount(receiverAccount).getCapability(/public/Vault)
      .borrow<&MyToken.Vault{FungibleToken.Receiver}>()
      ?? panic("Couldn't get the public vault.")
  receiverVault.deposit(from: <- newVault)
}
```

6). アカウント間の独自通貨の転送は以下のようにします

```
prepare(acct: AuthAccount) {
  let signerVault = acct.borrow<&MyToken.Vault>(from: /storage/Vault)
      ?? panic("Couldn't get the signer's vault.")
  let receiverVault = getAccount(receiverAccount).getCapability(/public/Vault)
      .borrow<&MyToken.Vault{FungibleToken.Receiver}>()
      ?? panic("Couldn't get the public vault.")
  receiverVault.deposit(from: <- signerVault.withdraw(amount: amount))
}
```

Community's video: https://lnkd.in/gb2GCDT5

9. Send a Transaction with a Non-Custodial Wallet

·How to send a transaction with a Non-Custodial wallet (ノンカストディアルウォレットで
トランザクションを送信するには)

An example of private key and public key generation is as follows
(private key, public keyの生成の一例は以下です)

```
flow keys generate --network=testnet --sig-algo=ECMA_secp256k1
```

The authentication function with the generated key is as follows
(生成したkeyによる認証関数は以下のようにします)

```
const EC = require('elliptic').ec
const ec = new EC('secp256k1')
const ADDRESS = '0x0c60359d10739baa'
const PRIVATE_KEY =
'64c6e9541f45676b75fb4c1c50b78d2103e1cd85004ba2074c7721f7769bc36a3'
const KEY_ID = 0

const sign = (message) => {
  const key = ec.keyFromPrivate(Buffer.from(PRIVATE_KEY, 'hex'))
  const sig = key.sign(hash(message))
  const n = 32
  const r = sig.r.toArrayLike(Buffer, 'be', n)
  const s = sig.s.toArrayLike(Buffer, 'be', n)
  return Buffer.concat([r, s]).toString('hex')
}

const hash = (message) => {
  const sha = new SHA3(256)
  sha.update(Buffer.from(message, 'hex'))
  return sha.digest()
}

const authorizationFunction = async (account) => {
  return {
    ...account,
    tempId: `${ADDRESS}-${KEY_ID}`,
    addr: fcl.sansPrefix(ADDRESS),
    keyId: number(KEY_ID),
    signingFunction: async signable => {
      return {
        addr: fcl.withPrefix(ADDRESS),
        keyId: number(KEY_ID),
        signature: sign(signable.message)
      }
    }
  }
}

module.exports = {
  authorizationFunction
}
```

To send a transaction with a Non-Custodial wallet, do the following
(ノンカストディアルウォレットでトランザクションを送信するには以下のようにします)

```
const sendTx = asnyc () => {
  console.log('Sending Tx')
  const transactionId = await fcl.send([
    fcl.transaction`
      transaction(number: Int, greeting: String) {
        prepare(signer: AuthAccount) {
        }
        execute(){}
      }
    `,
    fcl.args([
      fcl.arg(1, t.Int),
      fcl.arg('Hello', t.String)
    ]),
    fcl.proposer(authorizationFunction),
    fcl.payer(authorizationFunction),
    fcl.authorizations([authorizationFunction]),
    fcl.limit(9999)
  ]).then(fcl.decode)
  console.log(transactionId)
}
sendTx()
```

Community's video: https://lnkd.in/g5tqYRcU

10. Query for Events (イベントを取得するには)

To retrieve events periodically, do the following
(定期的にイベントを取得には以下のようにします)

```
const contractAddress = '1654653399040a61' // Take 0x from 0x1654653399040a61
const contractName = 'FlowToken'
const eventName = 'TokensDeposited'

const fetchEvents = async() {
  const latestBlock = await fcl.send([
    fcl.getBlock(true)
  ]).then(fcl.decode)

  let end = latestBlock.height - 10
```

```
  let start = end - 100
  const response = await fcl.send([
    fcl.getEvents(`A.${contractAddress}.${contractName}.${eventName}`, start, end)
  ]).then(fcl.decode)

  const {events} = response
  console.log(events)
  events.forEach((event) => {
    console.log(event.payload.value.fields, event.payload.value.fields[1].value.value)
  })
}
fetchEvents()
```

Community's video: https://lnkd.in/gJpqi3aw

11. Optional Binding

If you want to handle separately when a dictionary has a data and when it has not, you can write the code to get the state when the dictionary has a data by a variable.
(Dictionaryに値がある時とない時で処理を分けたい時、値がある状態を変数に移動して処理を書くことが出来ます)

```
let tests: {Int: Test} = { 10: Test(10), 20: Test(20) }
if let test = tests[10] {
  return test.fruites[0]
} else {
  return nil
}
```

·Optional Binding starts with `if let`. (Optional Binding は`if let`から始まります)

Community's video: https://lnkd.in/gWZtX-M5

12. How to Multi-Sign (マルチサインを行うには)

a: (Frontend:) Post request to the backend (バックエンドにリクエストを送信します)

Private key is only needed at the signature position, so the rest of the signature is placed at FrontEnd. Do the following

(Private keyはsignatureの場所でのみ必要なので、それ以外はFrontEndに置いておきます。以下のようにします。)

```
export const serverAuthorization = asnyc (account) => {
  const addr = '0x0c60359d10739baa'
  const keyId = 0
  return {
    ...account,
    tempId: `${addr}-${keyId}`,
    addr: fcl.sansPrefix(addr),
    keyId: number(keyId),
    signingFunction: async (signable) => {
      const signature = await getSignature(signable)
      return {
        addr: fcl.withPrefix(addr),
        keyId: number(keyId),
        signature: signature
      }
    }
  }
}
```

·The getSignature(signable) part is the POST request to the backend
(getSignature(signable)の部分がバックエンドにPOST requestするところです)

b: (Backend:) Take in the signable message from the frontend, and sign it with the Admin's private key that is stored on the backend.
(フロントエンドから署名するメッセージを受け取り、バックエンドに保存されているAdminのprivate keyを使用して署名します)

```
const signature = sign(signable.message)
```

c: (Backend:) Return the signature back to the frontend. (署名メッセージをフロントエンドに返します)

The front-end code at this time is as follows (この時のフロントエンドの処理は以下です)

```
const transactionId = await fcl.send([
  fcl.transaction`
  transaction() {
    prepare(frontendUser: AuthAccount, backendAdmin: AuthAccount) {
    }
  }
  `,
  fcl.payer(serverAuthorization),
```

```
  fcl.proposer(fcl.authz),
  fcl.authorizationz([fcl.authz, serverAuthorization]),
  fcl.limit(9999)
]).then(fcl.decode)
```

·This allows the Admin to pay transaction costs on behalf of the client, for example.
(これによってトランザクション費用を肩代わりすることなどができるようになります。)

Community's video: https://lnkd.in/gjFmFNTR

13. Optional References

·This is the area modified by Secure Cadence
(Secure Cadenceによって変更された部分です)

```
pub var tests = @{UInt64: Test}
```

To get a reference to a resource in this dictionary, do the following
(このDictionaryの中にある、リソースの参照を取得するには以下のようにします)

```
pub fun getRef(): &Test? {
  return &self.tests[0] as &Test?
}
```

This is a change by Secure Cadence where the resource in the Dictionary is Optional, to allow developers to handle nil cases properly. Previously, there was no ? was not added, which could result in nil references.
(ここはSecure Cadenceによる変更で、開発者がnilケースを適切に扱えるように、Dictionaryのリソースに Optional?を付けるように変更されました。以前は?がつかなかったのでnilの参照が生じる可能性がありました。)

Community's video: https://lnkd.in/gYvGX8dU & https://lnkd.in/gk4gKcFB

14. Configure Cadence VSCode Extension (VSCode Extensionのセットアップ方法)

1). Install Flow CLI. (flow-cliをインストール)

Flow CLI: https://developers.flow.com/tools/flow-cli/install

2). Install Cadence VSCode Extension. (Cadence VSCode Extensionをインストール)

Marketplace: https://marketplace.visualstudio.com/items?itemName=onflow.cadence
Github: https://github.com/onflow/vscode-cadence

3). Run `flow init` in your terminal to create flow.json file. This will highlight the Cadence code.
(flow initのコマンドを叩き、flow.jsonファイルを作成します。こうすることでCadenceコードがハイライトされます。)

4). Restart VSCode. (VSCodeを再起動する)

Community's video: https://lnkd.in/gkDWMzaw

15. How to Build an NFT Marketplace (NFTのマーケットプレイスを作るには)

a. Create a forSale object to manage the listing, a Capability to withdraw the NFT and a Capability to deposit the Flow.
(出品を管理するforSaleオブジェクトとNFTを引き出すCapabilityとFLOWを預け入れるCapabilityを作成します)

```
pub resource SaleCollection {
  pub var forSale: {UInt64: UFix64} // id of the nft --> the price of that NFT
  pub let NFTCollectionCaps: Capability<&MyNFT.Collection>
  pub let FlowTokenVault: Capability<&FlowToken.Vault{FungibleToken.Receiver}>
}
```

b. To allow only NFT holders to list, do the following
(NFTを保持する人のみが出品できるようにするには以下のようにします)

```
pub fun listForSale(id: UInt64, price: UFix64) {
  pre{
    self.NFTCollectionCaps.borrow()!getIDs().contains(id): "This SaleCollection owner does not have this NFT"
    price > 0.0: "It doesn't make sence to list a token for less than 0.0"
  }
  self.forSale[id] = price
}
```

c. To create a purchase function, do the following
(purchaseメソッドを作成するには以下のようにします)

```
pub fun purchase(id: UInt64, recipientCollection:
&MyNFT.Collection{NonFungibleToken.CollectionPublic}, payment: @FlowToken.Vault) {
  pre{
    payment.balance == self.forSale[id]: "This payment is not equal to the price of the
NFT"
  }
  recipientCollection.deposit(token: <-
self.NFTCollectionCaps.borrow()!.withdraw(withdrawID: id))
  self.FlowTokenVault.borrow()!.deposit(from: <- payment)
}
```

d. Create SaleCollectionPublic Interface
(SaleCollectionPublicインターフェースを作成します)

```
pub resource interface SaleCollectionPublic {
  pub fun getIDs(): [UInt64]
  pub fun getPrice(id: UInt64): UFix64
  pub fun purchase(id: UInt64, recipientCollection:
&MyNFT.Collection{NonFungibleToken.CollectionPublic}, payment: @FlowToken.Vault)
}
```

e. To store the marketplace resource in an account, do the following
(アカウントにマーケットプレイスのリソースを保存するには以下のようにします)

```
prepare(acct: AuthAccount) {
  let MyNFTCollection = acct.getCapability<&MyNFT.Collection>(/public/
MyNFTCollection)
  let FlowTokenVault = acct.getCapability<&FlowToken.Vault{FungibleToken.Receiver}
>(/public/flowTokenReceiver)
  }
  acct.save(<- NFTMarketplace.createSaleCollection(MyNFTCollection:
MyNFTCollection, FlowTokenVault: FlowTokenVault), to: /storage/MySaleCollection)
  acct.link<&NFTMarketplace.saleCollection{NFTMarketplace.SaleCollectionPublic}>(/
public/MySaleCollection, target: /storage/MySaleCollection)
}
```

f. To list a NFT on the marketplace, do the following
(マーケットプレイスに出品するには以下のようにします)

```
prepare(acct: AuthAccount) {
  let saleCollection =
acct.getCapability<&NFTMarketplace.saleCollection{NFTMarketplace.SaleCollectionPubli
c}>(from: /public/MySaleCollection)
    ?? panic("This SaleCollectiondoes not exist.")
  }
```

```
    saleCollection.listForSale(id: id, price: price)
}
```

g. To retrieve a marketplace listing, do the following
(マーケットプレイスの出品リストを取得するには以下のようにします)

```
pub fun main(account: Account): {UInt64: NFTMarketplace.SaleItem} {
    let saleCollection = acct.getCapability(/public/
MySaleCollection).borrow<&NFTMarketplace.SaleCollection{NFTMarketplace.SaleCollect
ionPublic}>()
        ?? panic("Could not borrow the user's SaleCollection.")
    let collection = acct.getCapability(/public/
MyNFTCollection).borrow<&MyNFT.Collection{NonFungibleToken.CollectionPublic,
MyNFT.CollectionPublic}>()
        ?? panic("Can't get the User's Collection.")
    }
    let saleIds = saleCollection.getIDs()
    let returnVals: {UInt64: NFTMarketplace.SaleItem} = {}
    for saleId in saleIds {
        let price = saleColletion.getPrice(id: saleID)
        let nftRef = colletion.borrowEntireNFT(id: saleID)
        returnVals.insert(key: nftRef.id, NFTMarketplace.SaleItem(_price: price, _ref: nftRef))
    }
    return returnVals
}
```

h. To cancel a marketplace listing, do the following
(マーケットプレイスの出品を取り消すには以下のようにします)

```
prepare(acct: AuthAccount) {
    let saleCollection = acct.borrow<&NFTMarketplace.saleCollection>(from: /storage/
MySaleCollection)
        ?? panic("This SaleCollectiondoes not exist.")
    }
    saleCollection.unlistFromSale(id: id)
}
```

i. To purchase NFTs from the marketplace using $FLOW, do the following
($FLOWを使用してマーケットプレイスからNFTを購入するには以下のようにします)

```
prepare(acct: AuthAccount) {
    let saleCollection = getAccount(account).getCapability(/public/
MySaleCollection).borrow<&NFTMarketplace.SaleCollection{NFTMarketplace.SaleCollect
ionPublic}>()
        ?? panic("Could not borrow the user's SaleCollection.")
```

```
    let recipientCollection = acct.getCapability(/public/
MyNFTCollection).borrow<&MyNFT.Collection{NonFungibleToken.CollectionPublic}>()
        ?? panic("Can't get the User's Collection.")
    }
    let price = saleCollection.getPrice(id: id)
    let payment <- acct.borrow<&FlowToken.Vault>(from: /storage/
flowTokenVault).withdraw(amount: price) as! @FlowToken.Vault
    saleCollection.purchase(id: id, recipientCollection: recipientCollection, payment: <-
payment)
}
```

For more information on the NFT Marketplace, please watch the community video below
(NFT Marketplaceの詳細は下記コミュニティビデオをご覧ください)

Community's video(33:40~): https://lnkd.in/gh8sC2g9

16. How to Verify Account Ownership (account-proof)

[Improved application reliability and server-side authentication (アプリケーションの信頼性向上とサーバーサイド認証)]

Flow CLI Github Reference (Flow CLI Githubリファレンス):
https://github.com/onflow/fcl-js/blob/master/docs/reference/proving-authentication.mdx

·The range of applications is wide because it goes to the server side to obtain nonce and authenticates it on the server side. Ex: Establish validity period, etc.
(サーバーサイドにnonceを取りに行き、且つ、サーバー側で認証する為、アプリケーションにさまざまな応用が出来ます。例:有効期限を設けるなど)

a. Retrieve account proof data from your application server when connecting to the wallet.
(ウォレット接続時にアプリケーションサーバーからアカウント証明データを取得してきます)

(Example) (一例)

[Frontend]
```
const resolver = async () => {
  const response = await fetch('/api/generate')
  const { nonce } = await response.json()
  return {
```

```
      appIdentifier: "YourAppIdentifier",
      nonce: nonce
    }
}
fcl.config()
    .put('accessNode.api', 'https://rest-testnet.onflow.org')
    .put('discovery.wallet', 'https://fcl-discovery.onflow.org/testnet/authn')
    .put('fcl.accountProof.resolver', resolver)
    .put('env', 'testnet')
```

[Backend]

```
import crypto from "crypto"
import { addNonce } from "../../storage"
export default function handler(req, res) {
    const nonce = crypto.randomBytes(32).toString('hex')
    addNonce(nonce)
    res.status(200).json({ nonce })
}
```

b. The data retrieved from the server should include a random nonce. The specific application identifier is displayed by wallets when users are asked to approve a signing request.
(サーバーから取得したデータにランダムなnonceが含まれる必要があります。特定のアプリケーション識別子は、ユーザーが署名要求を承認するよう求められたときにウォレットによって表示されます。)
(Example) (一例)

[Authentication on the Frontend side only (Frontend側でのみ認証)]

```
export async function login() {
    let response = await fcl.authenticate()
    const accountProofService = response.services.find(service => service.type === 
'account-proof')
    if (accountProofService) {
        const verified = await fcl.AppUtils.verifyAccountProof(
          "YourAppIdentifier",
          accountProofService.data
        )
        if (verified) { ... do something }
    }
}
```

[Authentication on Backend side (Backend側で認証)]

```
export async function login() {
    let response = await fcl.authenticate()
```

```
  const accountProofService = response.services.find(service => service.type ===
'account-proof')
  if (accountProofService) {
    const response = await fetch('/api/verify', {
     method: 'POST',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify(accountProofService.data)
    )
    const verified = response.json()
    if (verified) { ... do something }
  }
}

import * as fcl from "@onflow/fcl"
import { checkNonce, removeNonce } from "../../storage"
export default function handler(req, res) {
  const data = req.body
  const nonce = data.nonce
  if (!checkNonce(nonce)) {
    return res.status(200).json({ verified: false })
  }
  removeNonce(nonce)
  const verified = await fcl.AppUtils.verifyAccountProof(
   "YourAppIdentifier",
   data
  )
  res.status(200).json({ verified })
}
```

Community's video(00:16~): https://lnkd.in/ghi4GJTA

17. Account Storage Iteration

[To get a list of NFTs owned by an account (アカウントが所有しているNFTを一覧で取得するには)]

To retrieve all the information set in the PublicPath of an account, do the following (アカウントのPublicPathに設定されている情報を全て取得するには以下のようにします)

```
pub fun main(user: Address): [PublicPath] {
  let publicAccount: PublicAccount = getAccount(user)
  return publicAccount.publicPaths
}
```

To get a list of all NFT IDs owned by an account, do the following
(アカウントが所有する全てのNFTのID一覧を取得するには以下のようにします)

```
pub fun main(user: Address): [[UInt64]] {
  let publicAccount: PublicAccount = getAccount(user)
  let authAccount: AuthAccount = getAuthAccount(user)
  let ids: [[UInt64]] = []
  let iterationFunc: ((StoragePath, Type): Bool) = fun (path: storagePath, type: Type):
Bool {
    if type.isSubtype(of: Type<@NonFungibleToken.Collection>()) {
      let collection = authAccount.borrow<@NonFungibleToken.Collection>(from: path)
      let collectionIds: [UInt64] = collection.getIDs()
      ids.append(collectionIds)
    }
    return true
  }
  authAccount.forEachStored(iterationFunc)
}
```

·If false is returned in fun in iterationFunc, the loop stops there; if true, the loop continues.
(iterationFuncの中のfunの中でfalseを返すとループの処理がそこで止まります。trueを返すとループが続きます。)

To get a list of all NFT references owned by an account, do the following
(アカウントが所有する全てのNFTの参照一覧を取得するには以下のようにします)

```
pub fun main(user: Address): {Type: [&NonFungibleToken.NFT]} {
  let publicAccount: PublicAccount = getAccount(user)
  let authAccount: AuthAccount = getAuthAccount(user)
  let displays: {Type: [&NonFungibleToken.NFT]} = {}
  let iterationFunc: ((StoragePath, Type): Bool) = fun (path: storagePath, type: Type):
Bool {
    if type.isSubtype(of: Type<@NonFungibleToken.Collection>()) {
      let temp: [&NonFungibleToken.NFT] = []
      let collection = authAccount.borrow<@NonFungibleToken.Collection>(from: path)
      for id in collection.getIDs() {
        temp.append(collection.borrowNFT(id: id))
      }
      displays[type] = temp
    }
    return true
```

```
    }
    authAccount.forEachStored(iterationFunc)
}
```

Documentation:
https://developers.flow.com/cadence/language/accounts#storage-iteration

Community's video: https://lnkd.in/gZFbQxJb