Nautilus Trusted Compute

**System Architecture Document - 8 Jul 2022**

**Abstract**

Nautilus Trusted Compute is a data storage and code execution platform that allows groups of data creators to form pools of sealed data to which they can jointly grant third parties rights to execute authorized code without loss of privacy. The Nautilus platform uses Trusted Execution Environments (TEEs) for authorized code execution which can attest to data creators and other users which code has been executed on a sealed data pool. Data creators can issue digital rights tokens (DRTs) as fungible tokens recorded on a distributed ledger which encode rights for code execution to whomever presents them to the Nautilus platform. Digital rights and royalties generated from digital assets are managed using smart contracts which govern the data pools.

[**1 Actors**](#_skm0wsmwcrna) **2**

[1.1 Data creators](#_q66l0uyum24a) 2

[1.2 Code creators](#_ap5lm0s88re9) 2

[1.3 Third parties](#_of2abj7u5g1w) 2

[**2 System Overview**](#_fr3iwsbcz1p) **3**

[**3 Digital Rights Tokens (DRTs) [WIP]**](#_l4i4vyona2zl) **4**

[**4 Nautilus Tokens (NTLs)**](#_is9aosyswnqk) **7**

[**5 Technology Overview**](#_824sxsv2nlmy) **7**

[5.1 Trusted Execution Environments (TEEs)](#_g8g3echwex4q) 7

[5.2 Distributed Ledger Technology (DLT)](#_b3o2wtyx64lr) 7

[5.3 WebAssembly (Wasm)](#_yvdm8t1r0nr2) 8

[**6 System Components**](#_emr7nf6bk9o0) **8**

[6.1 Data Management System](#_1sj426bxphnw) 8

[6.1.1 Vault App](#_g5zywv8c84sv) 9

[6.1.1.1 Creating and Uploading Data Packages](#_p441dm18as2a) 9

[6.1.1.2 Pooling Data](#_bz9fw68gtddm) 12

[6.1.2 Issuing Digital Rights Tokens](#_des8x7noy3wi) 13

[6.1.3 Data Service](#_m41zu4w0guch) 13

[6.2 Code Execution System](#_qzimd852fjph) 13

[6.2.1 Execution Service](#_806no4pccpdo) 14

[6.2.2 Execution Enclave](#_wbb0numx24v0) 15

[6.3 Authorization System](#_om3tr5xenqum) 16

[6.3.1 Authorization Service](#_1srti3yscdd7) 16

[6.3.2 Oracle Nodes](#_viku0nuzw9rw) 17

[**7 External Systems**](#_depfp1frube3) **18**

[7.1 Object Storage](#_84fog6543ern) 18

[7.2 Code and Executable Repository](#_n20jk8rjydcy) 18

[7.3 Billing System](#_tuna3oo3t6ht) 18

[**Glossary**](#_cakm5mj57rcq) **19**

# 1 Actors

## 1.1 Data creators

Data creators are users who have data they wish to license to third parties without loss of privacy. Data creators upload data packages in order to issue Digital Rights Tokens (DRTs) that authorize third-party code executions on their data. A data creator can issue a DRT for a data package to allow other data creators to pool their data, as long as both datasets conform to a common data schema.

## 1.2 Code creators

Code creators write code compatible with data schemas defined in data packages. This code can then be referenced by a Digital Rights Token, thereby authorizing execution on a specific data package.

## 1.3 Third parties

Third parties want to execute code on sealed data packages. For this, they need to acquire appropriate DRTs from data creators which they provide to the Nautilus platform in turn. Third parties also need to acquire Nautilus tokens to pay data and code providers as well as the Nautilus platform for code execution.

# 2 System Overview

The Nautilus Trusted Compute (NTC) platform consists of four logical components:

1. Data Management System: The Data Management System provides functionality for uploading, pooling, and managing access to confidential data. This includes a client-side application for data creators that manages their keypairs, the data packages uploaded by the data creator and the issuance of Digital Rights Tokens.
2. Code Execution System: The Code Execution System is responsible for verifying code execution requests, code execution against data pools, and delivering results to authorized third parties.
3. Authorization System: The Authorization System is responsible for relaying information about the distributed ledger that records digital rights tokens to the rest of the system. The information from the authorization layer is used by the execution service to verify code execution requests by checking that the digital right held by the third party is valid and has been redeemed correctly.

In addition to the logical system components we use these external systems:

* Object Storage (or Azure Cosmos NoSQL database) for storing the sealed data and metadata.
* The Algorand network (or another distributed ledger) for creating and managing digital rights tokens.
* Code and executable repository (most likely github).
* A billing system used to pay for code execution. This billing system uses a token issued by Nautilus as an internal accounting mechanism.

Figure 1 (below) contains a visual overview of the system.

### 

### Figure 1: System Landscape Diagram

|  |
| --- |
| **Figure 1:** [System landscape diagram](https://c4model.com/#SystemLandscapeDiagram) of Nautilus Trusted Compute and the systems it interacts with. Shapes with a gray background are external systems that we use or integrate with. The dark blue shapes represent the systems that are part of Nautilus Trusted Compute, and navy blue represent people or actors that use the system. The dashed line represents a relationship. |

# 3 Digital Rights Tokens (DRTs) [WIP]

Digital Rights Tokens (DRTs) represent the right to perform a *code execution* on the data it references. DRTs are represented on the Algorand blockchain as Algorand Standard Assets (ASAs). A DRT represents the right for one user to execute a particular piece of code on a data pool. Each DRT references the ID of the smart contract that governs the data pool, the hash of the binary code, and the URL of the binary code. These values are immutable upon creation of the DRT and are referenced in the *creator, url,* and *metaDataHash* fields of the corresponding [*asset configuration transaction*](https://developer.algorand.org/docs/get-details/asa/). [include figure reference for DRT configuration]

Smart Contracts govern the data pools uploaded by data creators and manage the issuance and trade of DRTs. A smart contract has a one-to-one relationship with a data pool and is created from the data creator’s wallet. To represent the data pool, the smart contract stores a hash of the data pool in its global variable. The logic regarding the policies of how a data pool is governed and how DRTs are issued for the data pool reside in the smart contract code. For data pools that consist of multiple contributors, the smart contract keeps track of the contributors by issuing them a contributor token in the form of an ASA, unique to the data pool. Stored local variables unique to the relationship between the contributors’ wallet and the smart contract ensure the correct royalty amounts are paid out.

Data creators can instruct a smart contract to create DRTs for a specific data pool, whereby the smart contract will make use of [*inner transactions*](https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/#inner-transactions) to construct an asset configuration transaction, registering the smart contract address as the creator of the DRT.

From here, the DRT will reside in the account of the smart contract until purchased from a buyers account. The exchange price of the DRT is stored in the global variables of the smart contract.

Trading of DRTs takes place on the Algorand blockchain. In our implementation, a third party initiates the trade by requesting the creation of a [*group transaction*](https://developer.algorand.org/docs/get-details/atomic_transfers/#group-transactions) which consists of two transactions to the smart contract that holds the DRTs.

1. Instruction to purchase the DRT, referencing the Asset ID of the DRT.
2. Transfer of an agreed amount of Algos as payment for the DRT asset.

Group transactions resolve [atomically](https://www.google.com/url?q=https://en.wikipedia.org/wiki/Atomicity_(database_systems)&sa=D&source=docs&ust=1650434799043723&usg=AOvVaw0rWGwD9ts3_l38dkSVJFAf) on the Algorand blockchain and result in both of the substituent transactions being resolved simultaneously. The smart contract ensures the necessary checks before constructing a transfer asset transaction to the buyer’s account via an inner transaction.

Figure 2 contains a visual representation of the DRT issuance and trading process.

### 

### 

### Figure 2: Sequence diagram for DRT issuance

|  |
| --- |
| **Figure 2**: [Sequence diagram](https://www.uml-diagrams.org/sequence-diagrams.html) for issuing Digital Right Tokens. This diagram describes how a data creator can issue a DRT for their data, and the process of trading the DRT with a third party. The different colors in the diagram do not carry any meaning. |

The lifecycle of the DRT is completed upon its redemption. This is accomplished by transferring the DRT, via an [*asset transfer transaction*](https://developer.algorand.org/docs/get-details/asa/#transferring-an-asset)*,* to the creator address of the DRT, i.e. the smart contract address.

The necessary data, for the redemption of the DRT, is contained in the immutable fields of the ASA metadata. The following metadata is recorded in this manner:

* A URL pointing to the location of the relevant WASM executable;
* A cryptographic hash of each of the WASM executables.
* The address of the smart contract that created the ASA.

Nautilus provides a dashboard that data creators use to create and set prices for DRTs that they issue. The dashboard also allows third parties to purchase DRTs for data pools listed by the data creators.

Nautilus provides automated creation of smart contracts and the issuance and trading of their respective DRTs. Through these transactions, a small platform is charged for the transactions made through the dashboard.

# 4 Nautilus Tokens (NTLs)

The life cycle of the native Nautilus Tokens (NTLs) is similar to those of the DRTs. Nautilus Tokens are issued as an Algorand Standard Asset and all NTL tokens are transferred to the reserve address. Once the token is listed on exchanges, the price will be determined by supply and demand. Since the Nautilus platform will only accept NTL tokens, there will be demand for it. The only supply is the Nautilus Reserve which manages the ASA reserve wallet. To execute code on sealed data, third parties are billed based on the computing resources they require as well as the time it takes to execute code. Data creators are billed NTL tokens on a subscription basis for their data storage needs.

Once a token is transferred from the reserve wallet to a third-party or other wallet, it can be traded freely on an exchange. Whenever a third party redeems a certain amount of NTL tokens, these tokens are transferred to the reserve wallet again.

# 5 Technology Overview

Nautilus Trusted Compute is built using three key technologies:

1. Trusted Execution Environments,
2. Distributed Ledger Technology and
3. WebAssembly.

## 5.1 Trusted Execution Environments (TEEs)

Nautilus Trusted Compute uses TEEs to execute computations on sealed data inside an enclave. Using [Intel SGX](https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html) enclave technology, neither the system operator nor the cloud provider has access to the sealed data at any point during the code execution. Data accessed from outside the enclave always remains encrypted.

Our software is built for Intel SGX, which is a hardware-based enclave technology that uses a special instruction set built into Intel CPUs. By using TEEs, we are able to design a system that operates on sensitive data, within limitations defined by the data creators, and without risking exposure of the unsealed data.

Additionally, the Intel SGX attestation process allows us to prove to data creators that only authorized code ever executes on their data packages.

## 5.2 Distributed Ledger Technology (DLT)

Nautilus Trusted Compute (NTC) uses DLT to keep a transparent and publicly visible record of the history of smart contracts created to govern data pools and every DRT ever issued. Distributed ledgers serve as existing infrastructure for use in the encoding and trading of digital rights.

The initial implementation of NTC uses the Algorand blockchain as DRT ledger. Smart contracts are created and deployed on the blockchain and are recognised as Applications. DRTs issued from the smart contract are encoded on the blockchain as Algorand Standard Assets (ASA) and transactions involving issued DRTs are recorded on the blockchain.

Additionally, Nautilus issues a separate utility token used to pay for computations. This token serves as an internal accounting mechanism. The token is issued using Algorand Standard Assets.

## 5.3 WebAssembly (Wasm)

[Wasm](https://webassembly.org/) is a binary format for executables that focuses on portability. Nautilus Trusted Compute uses Wasm to facilitate general purpose computation by providing [a sandboxed Wasm runtime](https://docs.rs/wasmi/latest/wasmi/) that runs inside an enclave.

Several common languages like Typescript and Python have support for compiling to WebAssembly. This makes the technology approachable for a large number of developers to develop code targeting the NTC platform.

Trusted Compute uses [WebAssembly modules](https://github.com/WebAssembly/design/blob/main/Modules.md) to encode actions permitted by Digital Rights Tokens.

# 6 **System Components**

## 6.1 Data Management System

The Data Management System provides functionality for uploading, pooling, and managing access to confidential data. In addition, these functions require interactions with the blockchain for deploying smart contracts and issuance of DRTs.

Figure 3 contains a visual overview of the Data Management System and the components that it consists of.

### Figure 3: Container Diagram for the Data Management System

|  |
| --- |
| **Figure 3:** [Container diagram](https://c4model.com/#ContainerDiagram) of the Data Management System that shows the different containers and how they interact with each other. Shapes with a gray background are external systems that we use or integrate with. The dark blue shapes represent the systems that are part of Nautilus Trusted Compute, navy blue represents people or actors that use the system, and the light blue represents the logical containers of the Data Management System. The dashed line represents a relationship. |

### 6.1.1 Vault App

The Vault app is a client-side application used by data creators to manage their data packages, keypairs and to issue digital rights tokens for their data packages. The vault app uses Javascript and Angular and uses the Alogrand Javascript SDK to interact with additional GUI based implementations that are being planned.

Data creators use the Vault app to create and manage the keys used for sealing data for the enclave and signing transactions for issuing and transacting DRTs. Everything is stored client side on the device of the data creator or in a key manager offered by Nautilus (e.g. the Nautilus Wallet).

#### 6.1.1.1 Creating and Uploading Data Packages

Data creators use the Vault app to: (i) manage their private keys used to encrypt data; (ii) create, upload, and pool data packages; (iii) request attestations, and (iv) manage their digital rights tokens. A data package is a sealed dataset along with public and private metadata. All of the metadata is signed and can be verified to belong to the data package by means of a cryptographic signature provided alongside the sealed dataset.

The following metadata is included in the data package:

* The distributed ledger address of the data creator;
* The schema that the dataset follows;
* A list of oracle nodes trusted by the data creator.

The data package is encrypted with the public key of the execution enclave, so that the data can only be read inside of the execution enclave. All signing and sealing operations are performed client-side, using locally stored keys, and the data package is only ever transferred in a sealed state.

The enclave’s public key is shared via the attestation token as part of the attestation process represented in Figure 4. [Azure attestation](https://docs.microsoft.com/en-us/azure/attestation/workflow) generates the attestation token in step 9 of the sequence diagram to confirm trustworthiness of the execution enclave, thereby establishing a secure channel of communication between the vault app and the enclave. By sharing the key as part of the attestation process, the vault application can ensure that the data is encrypted for an execution enclave with a precisely known, predetermined state.

### Figure 4: Sequence diagram for attestation process

|  |
| --- |
| **Figure 4**: [Sequence diagram](https://www.uml-diagrams.org/sequence-diagrams.html) for attestation process. Interaction between the user, Nautilus system and Microsoft Azure system for the attestation process. The attestation process verifies the trustworthiness of the Nautilus Trusted Compute platform and the integrity of the binaries running inside it. [Microsoft Azure attestation](https://docs.microsoft.com/en-us/azure/attestation/workflow) generates an attestation token to confirm the trustworthiness of the enclave. The different colors in the diagram do not carry any meaning. |

Once the data package is encrypted, it will be saved to [object storage](#_84fog6543ern) as depicted in the last step of the data provisioning flow sequence diagram in Figure 5.

### Figure 5: Sequence diagram for the data provisioning flow

|  |
| --- |
| **Figure 5**: [Sequence diagram](https://www.uml-diagrams.org/sequence-diagrams.html) for the data provisioning flow, refer to [Figure 4](#_gv7zjzg9xt54) for the full attestation process. After the data creator has verified the trustworthiness of the platform through the attestation process, the user can start the process of uploading their data to the platform.The process includes verifying the data schema used, creating a data package, and encrypting it with public enclave key from the attestation process. The different colors in the diagram do not carry any meaning. |

#### 6.1.1.2 Pooling Data

A data pool is a specific instance of a data package that represents a collection of smaller data packages. A data creator can start the creation of a data pool by specifying the schema of the data pool along with the policies that are used to determine revenue sharing and how digital rights are issued for the data pool. The policies are encoded using smart contracts.

When a new data pool is created, the smart contract can be used to issue DRTs for the new data pool. Any DRTs issued for the data pool will be issued by the address of the smart contract.

When another data creator wants to join the data pool, they will issue a DRT that allows their data to be appended to the existing data pool. In addition to the data creator’s DRT, the existing data pool needs to issue a DRT that allows it to be used in the creation of a new data pool. The two DRTs are then used to authorize the creation of a new data pool on the system. The new data pool will have a new blockchain address and will inherit the policies from the existing data pool.

### 6.1.2 Issuing Digital Rights Tokens

Data creators can use the Vault app to issue DRTs for any of the data packages that they create. The information encoded in the DRT is described in the section on [Digital Rights Tokens](#_l4i4vyona2zl).

Additionally, the token needs to be created using credentials corresponding to the distributed ledger address included in the data package.

### 6.1.3 Data Service

The Data Service is responsible for storing and retrieving the sealed data packages uploaded by data creators using the Vault app. Data packages are sealed by the vault app for the execution enclave using [crypto\_box from NaCl](https://nacl.cr.yp.to/box.html). The sealed data packages are stored using NoSQL storage (Azure CosmosDB).

## 6.2 Code Execution System

The Code Execution System is responsible for verifying code execution requests, code execution against data pools, and delivering results to authorized third parties. The code execution system consists of the execution service and execution enclave.

Figure 6 contains a visual overview of the Code Execution System and its two components, the Execution Service and the Execution Enclave.

### Figure 6: Container Diagram for the Code Execution System

|  |
| --- |
| **Figure 6:** [Container diagram](https://c4model.com/#ContainerDiagram) of the Code Execution System that shows the different containers and how they interact with each other. Shapes with a gray background are external systems that we use or integrate with. The dark blue shapes represent the systems that are part of Nautilus Trusted Compute, navy blue represents people or actors that use the system, and the light blue represents the logical containers of the Code Execution System. The dashed line represents a relationship. |

### 6.2.1 Execution Service

The Execution Service is responsible for processing code execution requests and passing the data package, binary and authorization data to the execution enclave. After the code execution is complete, it returns the sealed result to the third-party.

### 6.2.2 Execution Enclave

The execution enclave is responsible for validating the code execution request. The following conditions are checked to ensure that the code execution request is valid:

1. The issuer of the DRT and the data creator that uploaded the dataset are the same party. For data pools, the application account of the data pool acts as the data creator.
2. The signatures of all the oracle nodes are validated using the public keys from the dataset.
3. The K responses from the different oracle nodes N match.
4. The binary executable corresponds to the hash included in the DRT.

After the code execution is complete, the execution enclave seals the result using the public key of the party that redeemed the DRT. This public key is included in the authorization results obtained from the oracle nodes. This process is depicted in Figure 8.

### Figure 8: Sequence diagram for code execution request

|  |
| --- |
| **Figure 8**: [Sequence diagram](https://www.uml-diagrams.org/sequence-diagrams.html) for code execution request. This diagram describes the code execution flow that starts when the third-party redeems a DRT and requests execution. This includes the process for requesting and compiling the authorization results from the oracle nodes, and the verification of the results inside of the execution enclave. The different colors in the diagram do not carry any meaning. |

## 6.3 Authorization System

The Authorization System is used to verify the authorization for a specific code execution request allowed by a DRT. We achieve this by calling a number of designated nodes on the distributed ledger, called *oracle nodes*, each of which has a unique keypair that is used to sign (and verify the validity of) any code execution requests.

An oracle node can be hosted by any organization, or individual, with the intention that data creators must choose to trust some (nonempty) subset of them. Additionally, the data creator must submit the addresses and verification keys (of the nodes) as part of the sealed and signed data package they wish to publish.

After a third-party redeems a DRT, they request the code execution allowed by the DRT using the Execution Service. The Execution Service then requests a signed authorization from all of the oracle nodes specified in the data package, using the Authorization Service.

The Authorization Service is responsible for compiling the responses from all the oracle nodes into a single result that is returned to the Execution Service. The Execution Service will then retrieve the executable listed in the DRT and pass it to the Execution Enclave, alongside the authorization results.

The Execution Enclave checks that the results from the oracle nodes, as well as all the other details returned during the authorization process, are consistent. Before proceeding with the code execution, the Execution Enclave ensures that there is no mismatch between the results of the authorization process and the metadata contained in the DRT.

Figure 8 contains a detailed view of the authorization process and how it interacts with the Code Execution System.

### 6.3.1 Authorization Service

The Authorization Service will query all of the trusted oracle nodes. Once a code execution request has been verified, each requested node will return a response containing:

1. Confirmation that the DRT has been redeemed correctly
2. The Address or other cryptographic identifier that indicates who created the DRT
3. The public key of the party that redeemed the DRT
4. Information encoded in the DRT:
   1. What right has been granted, and a hash of the executable that corresponds to that right.
   2. A hash of the data-package that corresponds to the DRT.

### 6.3.2 Oracle Nodes

When a data creator uploads a data package to a data repository, the trusted *oracle nodes* must be declared as part of this process. The oracle nodes are predesignated nodes on the distributed ledger, tasked with the verification and signing of code execution requests.

Figure 9 contains an overview of the Authorization System.

### Figure 9: Container Diagram for the Authorization System

|  |
| --- |
| **Figure 9:** [Container diagram](https://c4model.com/#ContainerDiagram) of the Authorization System that shows the different containers and how they interact with each other. Shapes with a gray background are external systems that we use or integrate with. The dark blue shapes represent the systems that are part of Nautilus Trusted Compute, navy blue represents people or actors that use the system, and the light blue represents the logical containers of the Authorization System. The dashed line represents a relationship. |

# 7 **External Systems**

## 7.1 Object Storage

Nautilus Trusted Compute uses object storage as a mechanism to store sealed data packages. The initial implementation will likely use Azure Blob Storage, but the system design does not depend on any specific storage service or mechanism.

## 7.2 Code and Executable Repository

Nautilus Trusted Compute integrates with existing code and executable repositories to retrieve executables linked to issued DRTs. The system design does not depend on a specific implementation, but our initial implementation uses GitHub.

## 7.3 Billing System

Third parties pay for code executions using NTLs that are deposited, before execution, into an escrow account. The escrow account is managed by a smart contract instantiated on the Algorand Blockchain as an application.

A single smart contract, written in PyTeal, on the Algorand blockchain network, is deployed as an escrow and is used by third parties in order to deposit NTLs. This escrow account has a minimum holding value required to perform any code execution. Each code execution has an associated cost and, after every code execution request, this amount is deducted from the third party's balance held in the smart contract escrow. The deduction is then transferred to Nautilus.

# 

# 

# Glossary

| **Term** | **Explanation** |
| --- | --- |
| Attestation | The mechanism by which a third entity can verify that a software entity is running within an enclave on an SGX enabled platform. |
| Data Creator | Any organization, individual or entity that creates data packages using the NTC platform. |
| Data Package | A sealed dataset along with public and private metadata. |
| Data Pool | A data package that consists of data provided by two or more data creators. |
| Digital Rights Token (DRT) | A digital token issued on a distributed ledger that represents the right to execute authorized code on a data package. |
| Distributed Ledger Technology (DLT) | The infrastructure and protocols that allow the recording and validation of transactions on an immutable ledger. |
| Enclave | A computing context which uses hardware mechanisms to provide an isolated execution environment for trusted computation. |
| Oracle node | The oracle nodes are predesignated nodes on the distributed ledger, tasked with the verification and signing of code execution requests. |
| Vault | The client application for NTC that is used for key management, issuance of DRTs and the sealing and unsealing of data. |

## 