**Nautilus Trusted Compute MVP**

**Implementation Plan**

Nov 3, 2022

[**1. User Requirements**](#_b6hk1b11bukl) **2**

[1.1 Data pool creators](#_b93krx2e3oqx) 2

[1.2 Data contributor](#_5yvt3h6qhux0) 2

[1.3 Code contributors](#_692qn2dakgel) 2

[1.4 Data user](#_4wqj69awe06a) 3

[1.5 Fees](#_i1ac8pdx7o24) 3

[**2. Implementation Plan Overview**](#_dc9mxoalq7zz) **3**

[Authorization/Verification System](#_lic99eohlo23) 5

[Attestation](#_cq4cjodqmso1) 5

[Token Management System](#_aah7r2yzcci2) 5

[1. Smart Contract Build](#_txe4brjbaaqo) 5

[2. Smart Contract Deployment](#_oxfr72nojbw4) 5

[3. Smart Contract API Calls](#_bp3y3udrr0ab) 6

[Execution Management System](#_cerwmwtq59vq) 6

[Vault Execution Binaries](#_y09v5si6oqd) 6

[Data Management System](#_176tvrdrsfm6) 7

[Append Function](#_8hs4c5avj131) 7

[Metadata Database](#_yxf6ufer3ja3) 8

[Data Schema & Validation](#_rp97ss4wg1sk) 8

[Data APIs](#_mvjn0yn11zjo) 9

**3.** [**Technical Requirements**](#_fln2q635bqjo) **10**

[Data APIs](#_w96dblle1l7t) 10

[Vault Execution Binaries](#_8nwgvc7gb34z) 10

[Metadata Database](#_z624oo6cnmgm) 10

[Data Schema & Validation System](#_dfi7g5lftoud) 11

[Attestation Nodes](#_4pdoxyo5634w) 11

[Authenticated Encryption](#_pfoqs5ena5qd) 11

[Data and Code Authorization](#_dgj8h033l74u) 12

[Data Sealing](#_t5di823ggm3r) 12

[Smart Contracts](#_3o1p75ctllqj) 12

# User Requirements

### 1.0 Users

The term “user” is generic and includes data creators, data contributors, code contributors, and analysts. A user can register on <https://ntls.io> and create a profile. Once a user returns to the web app, she can log in. A user who is logged in sees a button to go to their “Profile”. If this option is selected, users can manage their account settings and personal details. They can set up a list of wallet addresses for their numeraire (e.g. $Algo), contributor tokens and digital rights tokens and choose a preferred wallet address.

TODO: Decide which personal details are necessary

TODO: Design key recovery

1.0.1 Personal/Private Data

* Multi Factor Authentication/2FA for key recovery
  + Refactor Wallet auth to use 2FA instead of multi factor authentication & Generation Agolorand account or link existing algorand account.
  + Find a way to send them algos
    - MVP: Link out to exchange.
    - Phase 2: Include exchange from Wallet (cross library) as in plan 2024
* Identity Verification

#### 

1.0.2 User Wallets

There are two proposed options for handling user wallets. The first issue is to require a user to create a Nautilus wallet, wherein we would

### 1.1 Data creators

A data creator is a user. Using the Nautilus web app, a data creator (she) can create a data pool. For this, she logs into her user account on the https://ntls.io website. Once the data creator is logged in, she sees a list of all her data pools below a button to create a new pool.

Once a data creator starts to create a new pool, she is guided through a four-plus-one step process. First, she defines a data schema for the pool. Second, she selects the file containing one or multiple rows of data conforming to the schema. Third, the user selects how many Contributor Tokens she wants to receive for the data. She also selects the wallet in which to receive these tokens. Fourth, the data is uploaded. And fifth, the data creator selects from a list of data rights that are available for the pool. It is also possible to skip this step and add data rights later.

For each existing pool in her list, the data creator is presented with an option to “Manage Digital Rights”. Once clicked, this option allows the data creator to “Create New Rights” and “Manage Existing Rights”. New rights are selected from a list of possible rights (including the AppendDRT) and the user selects a price for each new right. This price is fixed for all future issuances of this right. Rights are created by issuing Digital Rights Tokens via an interface with an Algorand smart contract.

When managing existing rights is selected, the data creator can “Issue Existing Rights”, “Stop Issuing Rights”, and “Delete Data Pool”.

When “Issue Existing Rights” is selected, the data creator is presented with a list of all existing rights and an overview of how many of each have been issued. She can then select one such right and select a number how many more she wants to issue (up to a maximum).

“Stop Issuing Rights” should be a simple checkbox that users can tick if they want to prevent the smart contract from selling any further rights.

When “Delete Data Pool” is selected, the platform checks if all contributor tokens have been returned to the platform. If that is not the case, the platform tells the data creator how many contributor tokens are still outstanding. The data creator is then asked whether she wants to redeem contributor tokens and a workflow is started that allows the data creator to send contributor tokens to the smart contract managing the data pool.

Data creators receive a proportion of the funds received from selling DRTs, including the AppendDRT. Data creators are able to trade Contributor Tokens on a secondary market.

TODO: Discuss how data schema definition looks like;

TODO: Decide whether we have a step where the data’s conformity to the specified data schema is checked;

TODO: What governance mechanism decides the price of DRTs?

### Data Schema and Validation

We must enforce a data schema to:

1. Ensure interoperability of all the datasets comprising a data pool,
2. Ensure compatibility of datasets with the binary executables corresponding to DRTs.

In the case of the MVP, the most direct way to validate columnar data is to use a JSON standard. To ensure that different datasets can be combined and run with a given binary, each JSON dataset and each binary must specify a data schema. The JSON Schema provides a well-maintained, fast and comprehensive library in [Rust](https://crates.io/crates/jsonschema) and [Javascript](https://www.npmjs.com/package/jsonschema). It provides a media type for describing JSON documents, as well as functions to compare a JSON dataset to a JSON Schema. We have implemented a prototype that use jsonschema to check a datafile against a schema file: [here](https://github.com/ntls-io/nautilus-trusted-compute/blob/main/rust-workspace/crates/ntc-data-packages/src/data_packages/json_schema.rs) and here.

Unknowns:

**Where to validate?** Even on an MVP and in internal testing, the rate of schema mismatch is likely to be high. We can use the Rust implementation referenced above to validate a data file in the enclave before sealing data. However, an error at this point would be slow and resource intensive. It is also not clear we can use the jsonschema tool in an enclave environment due to the no\_std requirement. Relying on the backend for validation then has the risk of increased development time and a clunky interface at a key point of the UI.

The alternative is to port this code to JS and check a schema on the client side, before it is encrypted and sent to our servers. This would be easier to develop and provide a better user experience.

TODO: Design selection of data rights;

TODO: Discuss what other options a data creator must have under “Manage Existing Rights”

TODO: Design workflow for sending contributor tokens to the ASA

TODO: Discuss how contributor and digital rights tokens are held by users, in the Nautilus wallet or in an external Algorand wallet?

OLD:

DRTs can be purchased from the smart contract and issued a digital token to the wallet of other users. The tokens grant rights to run code on the data managed by the data pool smart contract. One such token allows for an “append” operation. Users who redeem a DRT for an append operation are data pool contributors, and receive royalties for any funds generated by the data pool after they have joined.

### 1.2 Data contributor

A data contributor is a user. Once logged in, the data contributor can select an existing pool with available AppendDRTs and select “Append Data”. Once selected, a workflow starts that first checks whether the data to be appended is consistent with the schema of the data pool and then guides the data contributor through the purchase of an AppendDRT. Note that data contributors cannot “Manage Existing Rights” like data creators.

If an analyst purchases a DRT for the data pool, data contributors receive royalties proportional to the sample size of their data.

**Removal of data**: In order for a data contributor to withdraw their data, there must not be an unredeemed execution token that references that data. The bookkeeping for the location of a datasets in a data pool, and the corresponding DRTs which reference a dataset can be tracked using Algorand boxes. However, it may not be an MVP feature.

TODO: Discuss whether data contributors should be able to remove their data from the pool.†

### 1.3 Code contributors

For the MVP, users will be able to execute a set of binaries provided by Nautilus. In the future, users will be able to upload their analysis code via WASM binaries and issue tokens for the use of the code, and data pool creators will be able to issue DRTs for code created by users.

### 1.4 Analyst

A user with a Nautilus account can browse existing data pools and purchase DRTs that are offered by the data pool creator. The analysis code will run inside the TEE enclave and store the sealed result. The user will be able to redeem the result via the NTC client at any time after the computation is completed.

### 1.5 Fees

Nautilus will charge fees as a percentage of the value of each redeemed DRT.

# Implementation Plan Overview

The following components are required to satisfy the above user requirements:

1. Data Management System
   1. Seal, store and validate the format of user data
   2. One or more backend APIs:
      1. Allow the frontend to transfer sealed data to storage
      2. Allow the backend to retrieve data from storage and initiate enclave execution
      3. Store relevant metadata on users, datasets,
2. Code Execution System

Handles and verifies code execution requests and delivering results

* 1. The execution enclave, implemented with Apache Teaclave and the Intel SGX SDK.

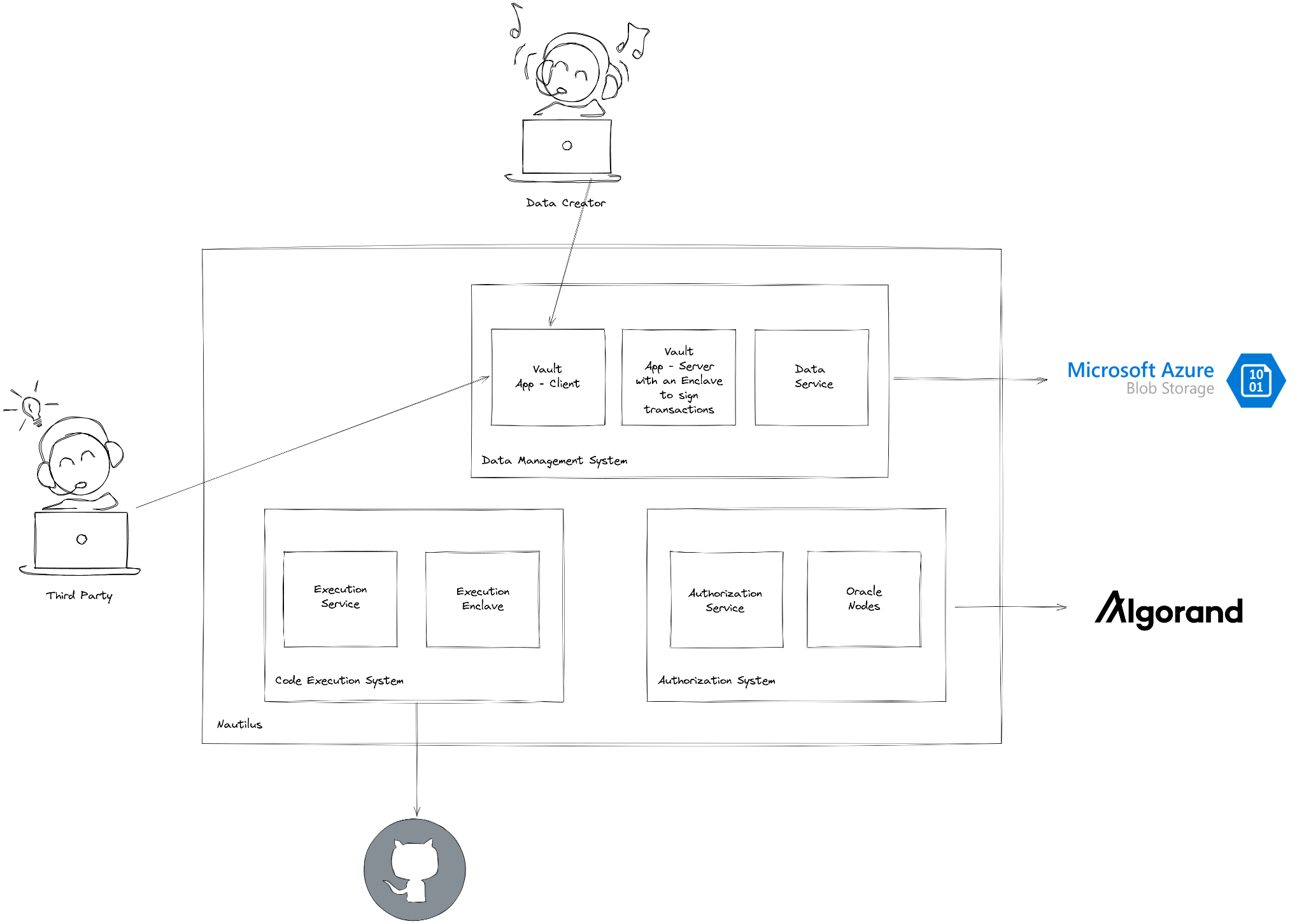
1. Authorization/Verification System

Used to verify the execution of code on data in the enclave.

* 1. Attestation using Oracle nodes
  2. Verifying data packages and compute code
  3. End-to-end encryption

We also implement smart contracts and DRT functionality

1. Digital Rights Tokens (DRTs)
   1. Smart Contracts
      1. A smart contract built using the Pyteal library deployed on the Algorand blockchain.
      2. Frontend interface written with Angular and Typescript to interact with the smart contract and with the backend



## 

## 2. 1 Authorization/Verification System

### 2.1.1 Attestation

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.*

Work required:

* The attestation is handled in simulation mode in the Nautilus Wallet. Reproduce and check the simulated validation.
* Research the requirements for accessing and using oracle nodes.

2.1.2

## 2.2 Token Management System

The token management system encompasses interactions with the blockchain such as the smart contract and the issuing of digital right tokens.

#### 2.2.1 Smart Contract Build

The first is the development of the smart contract code using the Pyteal library. This includes all the subroutines and functions contained and explained in the following [wiki](https://github.com/ntls-io/ntc-smart-contract-mvp/wiki). The output of this part are approval.teal and clear.teal files.

#### 2.2.2 Smart Contract Deployment

The automatic deployment of the smart contract to the Algorand Testnet every time a user uploads data and creates a data pool. The smart contract requires a sequence of transactions (instructions) to successfully deploy itself. The Algorand Javascript SDK is used for this deployment and the approval.teal and clear.teal files are required.

The smart contract is designed so that all the steps in creating the smart contract are successfully completed before it can start operating as a data pool.

The successful creation of a smart contract requires the following transactions in the following sequence sent to the Algorand Testnet.

1. An application creation transaction sent to the testnet blockchain to deploy the teal files, signed by the data pool creators nautilus wallet.
2. An “optin” transaction to the newly deployed application from the smart contract creator.
3. A transaction to fund the smart contract, requiring the smart contracts minimum balance.
4. An application call transaction to the smart contract from the creator to create the contributor token.
5. An “optin” transaction to the contributor token from the smart contract creator.
6. An Application call transaction to the smart contract from the creator to add the smart contract creator as a data contributor

A UML diagram of the above process can be found [here](https://drive.google.com/file/d/13E-krmtFSBKlUx4TIjJcp5oWPL3ylN_2/view?usp=share_link).

#### 2.2.3 Smart Contract API Calls

The Algorand Javascript SDK will be used to develop API calls to interact with the smart contract. UML diagrams are used to explain the method to interact with the smart contract.

API calls need to interact with the smart contract are:

* An “optin” transaction to the smart contract from an account
* An “optin” transaction to an asset from an account
* A send payment transaction to the smart contract
* Application calls from an accounts for each method
  + Create DRT method
  + Update DRT Price method
  + Add New Contributor Token method
  + Buy DRT Method in the form of a group transaction (1. App call, 2. Payment)
  + Create Contributor Token Method
  + Claim Fees Method
  + Update Data Package Method

## 2.3 Execution Management System

### 2.3.1 Vault Execution Binaries

We currently have a [basic median function](https://github.com/ntls-io/wasm-exec-sgx/tree/main/get-median-wasm) written in Rust and compiled to WASM binary. The addition of other functions is not a current priority. We also have *standard deviation* and *mean* on a branch waiting to be merged - <https://github.com/ntls-io/wasm-exec-sgx/pull/9> - Two binaries are created for each function - one for INT and one for Float.

Eventually we also want binaries for linear regression and other basic analysis and machine learning algorithms. We ultimately envision a system where a user can contribute custom code as a WASM binary and issue rights to run code.

Currently all the binaries use a byte vector - we need to investigate the use of arrays in future - this is especially important for any machine learning binaries.

Work required:

## 2.4 Data Management System

### 2.4.1 Append Function

The function to append data to an existing data pool is distinct from the other functions in that it is only used internally. For that reason it may not be necessary to execute it as a WASM binary, and instead run it as Rust code directly inside the enclave. This would have performance and maintenance advantages. The mean, median, linear regression functions, and other functions would be developed as WASM binaries for use as DRTs and as examples for external code providers.

*Currently we have two options for appending the data:*

* WASM Binary (Append Vector): <https://github.com/ntls-io/wasm-exec-sgx/pull/10>
* Append JSON (Inside enclave using Rust): <https://github.com/ntls-io/nautilus-trusted-compute/pull/70>

Append JSON can join any two JSON data sets as long as it follows the predefined format:

| {"pool": [ ]} |
| --- |

Work required:

* Update append to download the two JSON files from CosmosDB (waiting on sealing task).
  + Data downloaded in sealed format - unseal and validate schemas.
  + Update ECAL to reflect this change.
* Data validation needs to happen before joining two JSON files - need to ensure the schemas are the same.
  + It would be done when uploading, but some check should also be done on the combined dataset to insure data integrity.
* New data pool needs to be sealed and then uploaded to CosmosDB after each append task.

### 2.4.1 Metadata Database

Store data that is not available in the blockchain, HPKE, or is too slow with those methods. Use a CosmosDB deployed on Azure.

Connection details

Database schema

Collections

* Append collection
* Smart contract
* Data package
* Wallet [maybe just use the NW metadata database]
* DRT collection
* Data schema types
  + schema\_desc\_short (128 bytes) to smart contract maybe

### 2.4.2 Data Schema & Validation

Previous research on data schema definition was done by Pi on the [Data Schema Options](https://docs.google.com/document/d/1ZR25tgpHCkr3HXdLfMeo0D41GXOeFW6C_Sp3Y8xJ7co) document. An option for schema definition is the [cue dev package](https://docs.dagger.io/1010/dev-cue-package/) from Daggar. JSON validation, etc.

We currently have a custom crate for validating JSON schemas - [GitHub](https://github.com/ntls-io/nautilus-trusted-compute/tree/main/rust-workspace/crates/ntc-data-packages)

Planned user flow:

* A user initiates the data pool creation process (or the data contributor process if they are adding to an existing dataset).
* Before they upload (call an add\_data api endpoint), they should be able to search for a schema, (list\_schemas api endpoint),
  + Choose schema
* Compare new data set to selected schema
  + Ensure all fields from schema are contained in dataset
  + Check data type in each column

Work required:

* Add oracle node information to Metadata - [Line 20](https://github.com/ntls-io/nautilus-trusted-compute/blob/main/rust-workspace/crates/ntc-data-packages/src/data_packages/common.rs)
* Update crate once sealing task is completed - [Link to GitHub](https://github.com/ntls-io/nautilus-trusted-compute/blob/main/rust-workspace/crates/ntc-data-packages/src/data_packages/sealing.rs)
* add\_data api endpoint (Use the Data POST Api endpoint - [PR](https://github.com/ntls-io/nautilus-trusted-compute/pull/69))
* list\_schemas api endpoint (Can clone functionality from Data API endpoint - [PR](https://github.com/ntls-io/nautilus-trusted-compute/pull/69))

## 

### 2.4.3 Data APIs

We currently have a PR for two APIs: <https://github.com/ntls-io/nautilus-trusted-compute/pull/69>

The APIs connect to CosmosDB and can upload data and query data using specific fields. Currently this is the format for CosmosDB:

| {  "\_id" : ObjectId("63612d200d0cc3d50ecda3b5"),  "user\_id" : "Test Client 01",  "pool\_name" : "NTLS first pool",  "sealed\_data" : "This is some sealed data",  "created\_on" : {  "$date" : 1667312928083  } } |
| --- |

API endpoint: <http://127.0.0.1:8000/api/data>

**POST - upload/add data**

| curl --header "Content-Type: application/json" \  --request POST \  --data '{"id":"Test Client 01","pool\_name":"NTLS third pool","sealed\_data":"This is some sealed data"}' \  http://127.0.0.1:8000/api/data |
| --- |

**Get - View/get data**

| curl --header "Content-Type: application/json" \  --request GET \  --data '{"id":"Test Client 02"}' \  http://127.0.0.1:8000/api/data |
| --- |

Create Data Package Endpoint

* Nautilus address
* Data package hash
* Number of rows
* Smart Contract ID ( Application ID )
* Smart Contract Address
* Contributor Token ID ( Asset ID )
* DRT ID ( Asset ID )
* DRT Name
* DRT Exchange Rate

Work required for Data APIs:

* Set-up proper CosmosDB for NTC (CosmosDB used for testing - “ntc-data”)
* Update environment variable to include connection string to CosmosDB
* Waiting on sealing functionality - currently we are just using a placeholder String, this needs to be updated - “*This is some sealed data*”
* SSL/TLS for connecting to CosmosDB?
* API keys for specific users?

Data Storage API

* http://127.0.0.1:8000/api/data/store\_dataset
* http://127.0.0.1:8000/api/data/retreive\_dataset

## 

# 3. Technical Planning

Technical components and their requirements and time estimates for task planning. Where necessary, refer to separate technical documents for components. Fill in details for required work and implementation.

## 3.1 Data Service

In progress Create 2 APIs (upload data pool & query data pool for creator) - 1 weeks

Metadata and other functions, *See section 2.4.3* - [PR](https://github.com/ntls-io/nautilus-trusted-compute/pull/69)

In progress TEE configuration and deployment -

Set up teaclave environment - *1-2 weeks* - 70%

Migrate from sandbox to SGX hardware - *2-3 days* - 30%

Launched SGX Benchmarking

run a set of computations in SGXv1 and SGXv2 hardware to establish the performance of our technical stack. - *1-2 weeks*

In progress WASM Binary development

develop a set of basic analysis functions - mean, median, linear regression, basic clustering, etc. to use as a demonstration and an example for code providers - *2-3 weeks* - 70%

## 3.2 Execution Service

In progress Create 2 APIs (send oracle node responses to enclave & gather code and data to send to enclave) - *3 weeks*

In progress WASM runtime (research) - test with new binaries, test new hardware (link) - *3 weeks* - 70%

Testing Append Wasm binary - *1 week* - 50%

Not started Create HTTP server

Not started Validate Code Execution Request

Not started Seal result in enclave and retrieve it , Display result in file, display in interface, R2

## 3.3 Data Storage [Bingle Kruger](mailto:bingle@ntls.io)

In progress Design Cosmos Database - 2 days

In progress Create Cosmos Database - 2 days

## 3.4 Data Schema & Validation System [Bingle Kruger](mailto:bingle@ntls.io)[Joseph Bochenek](mailto:joe@ntls.io)

Not started Build a system for generating data schemas to ensure compatibility with combined data pools and to determine how code is run on data (i.e. which columns for which binaries, etc)

Not started A system to validate new files based on data schemas

In progress Search functionality and API for schema

Notes:

* JSONSchema code (link) written in the context,
* This can be done on the front end or in the enclave, probably client-side for the MVP.
* Verifying the data in very large datasets can require a lot of resources on the clients local machine.

## 3.5 Attestation [Bill Guo](mailto:bill@ntls.io)[Jean-Pierre De Villiers](mailto:jean-pierre@ntls.io)

Not started Create API to query oracle nodes, validate responses and send responses to Execution Service

Not started Change Oracle Nodes DRT structure

Not started Create SGX DCAP Attestation

Notes:

* We have mock attestation at the moment, fetch enclave report rather than the full DCAP, needed for both. Herman created a PR on the NTC repo ([SGX DCAP Attestation](https://github.com/ntls-io/sgx-dcap-attestation-rust)).
* Oracle nodes verify if DRTs are valid, check details, so they must be modified to use smart contract addresses rather than DRT tokens.

## 3.6 Authenticated Encryption

Not started HPKE Research - 1 week

Require JP input

Not started Data Sealing

Not started Data and Code Authorization

## 3.8 Smart Contracts

Testing Smart Contract Prototype build - 5 weeks

Research Smart Contract DRT append Process (Part of build but pending discussion with Co)

* Time estimate if there's a clear path - ~ 3 days includes documentation and testing
* For data to be appended we need to 1st validate the incoming data and 2nd instruct the smart contract to add the contributor so that they can receive royalties, so we can't just let anyone join unless their data has been validated.. Therefore we need some input from the enclave validation.
* Option 1 is for contributors to search for data pools to contribute towards, and after approval from the creator and validation from the enclave, the creator instructs the smart contract to add a data contributor
* Option 2 is for contributors to purchase the “Append” DRT (Could be free of charge). Upon redemption of the DRT, the users data is first validated by enclave, then if successful a transaction is sent from the redeemers wallet to add themselves as a contributor and a transaction is sent from the enclave to approve the data validation. The smart contract will only accept the new data contributor provided that there is the approval transaction from the enclave.

Not started Build the Create Smart Contract Deployment Method from SDK, Section 2.2.2 (~1 week)

Not started Smart Contract Javascript/Python SDK for APIs, Section 2.2.3 (~2 weeks)

Not started Integrate smart contract deployment with nautilus backend (~ 2 weeks)

[Smart contract requires inputs and transactions signed by enclave]

## 3.9 Vault App

Not started Create Upload Page for Vault App Frontend and link it to Data Service API

Not started Create Schema Selection Page

Not started DRT “Marketplace”

Marketplace

Not started Sign up & Register

Not started Front end of the marketplace

Not started Query the smarcontracts to retrieve DRTs

Not started Purchase process. Send algos to smartcontract

## 3.10 Documentation

In progress Revise architecture doc - 2 days - 60%

In progress Revise Implementation Plan - 3 days - 40%

In progress Revise technical documentation

## Research Items

Questions:

* Append DRT logic - meeting with Co Nov 23, 2022
* ~~Why do we need HKPE which is scheduled to consume five weeks of developer time.~~ 
  + ~~Why don’t we use the primitive sealing methods for MVP and focus on HKPE after that?~~
  + ~~Answer: we use HPKE for better encryption and metadata~~
* ~~How will the append function be created and does it need to be a binary?~~
  + ~~Why don’t we just use Rust for MVP?~~
* Restrict extra columns in the datasets or control them with schema verification - data projection or storing columns separately
* Do we need an updated SDK for memory and multi-threading
* Attestation methods - DCAP vs internal driver for EPC management

References to merge with this document:

[SystemsArchitecture\_NTC](https://docs.google.com/document/d/1veN4eiil2Lbrl23Z9OQ63Gidkd0_yw-_1zWwJ_I_ess/)

[Product Roadmap](https://docs.google.com/spreadsheets/d/1LR08J3sbHAYVz3AdkokcrFfqEoejJpS_ZX1wnumQdck/edit?pli=1#gid=225711394)

[NTC Decisions](https://docs.google.com/document/d/1sIYKeYgh8oz_ZZFQtWgPhUpEuquG-Ve7OHQAb8nL7Wk/edit#heading=h.fgm3fue14tg5)

[MVP Vault Data Processing](https://docs.google.com/document/d/1lWJMMaVluHj-5qmjkGjyzDDaUSOlbwGekgdOtfxEJUM/edit#heading=h.99dr7usurpz2)

[Trusted Compute](https://drive.google.com/drive/u/0/folders/1X5JE39hGk42ws6XtC0TQqjjtmyfrSTDg)