Design

Introduction

This file gives the general design of Alexandria.

Client Commands

The flow starts on the client side. In the example of bootstrapping, the UI should produce a struct BootstrapRequest. This BootstrapRequest should be transformed to a Command that can be used to produce a Hyperledger Sawtooth transaction. To make this possible, a Command should have the following properties:

- A Google Protocol Buffers object as specified in the data model.
- A list of input Sawtooth addresses.
- A list of output Sawtooth addresses.
- The keypair to sign with.

Introducing the Command interface allows testing without a real blockchain. We can apply Command objects to the state. To do this, we need a Golang interface that is implemented by the Hyperledger Sawtooth processor context. This means that we have an interface BlockchainAccess with methods GetState, SetState and DeleteState.

Server Command Processing

The transaction handler's Apply method should wrap the transaction payload into a regenerated Command object and apply that to the BlockchainAccess instance that is the Sawtooth context. Applying the command should not update the state directly, but should result in a slice (Golang's version of an array) of StateUpdate objects. The StateUpdate objects are both used to do the updates of the state, and they are used to generate the Sawtooth events to emit.

When working with Sawtooth state, it is useful to hide unmarshaling and marshaling in a BlockchainAccessAdapter object. Applying a Command to the BlockchainAccess thus constitutes the following steps:

- Create a BlockchainAccessAdapter and fill it by reading the relevant addresses.
- Check the validity of the command.
- Generate the StateUpdate objects.
- Apply the StateUpdate objects to the BlockchainAccessAdapter.
- Flush the BlockchainAccessAdapter by marshaling the modified Google Protocol Buffers objects and applying SetState and DeleteState on the BlockchainAccess.
- Transform the StateUpdate objects into (eventType, attributes, payload) tuples.

- Emit these tuples through the Hyperledger Sawtooth context.

The client should be able to detect missed events. This is achieved by adding some extra data as is explained in the data model.

Maintaining Local Data

The Client and the Major tool should access a common Data Access Object (DAO) component that maintains the local database. It should expose a function to receive Sawtooth events. The function should translate each incoming event to an Event object. An Event should be able to apply itself to a database/sql transaction object. Finally, the DAO component should provide methods to query the database resulting in structs holding data elements that are relevant within Alexandria. For example, the DAO should have methods like SearchPersonByKey(), IsBootstrapped() and GetPrices().

Subscribing to Sawtooth events should not be in the DAO component to make the DAO component easier to test.

Packages

We have the following packages:

- model
- command
- dao
- blockchain
- processor
- major
- client
- portal

Package **model** holds Google Protocol Buffers and Data Definition Language code to describe the data. It also holds helper functions to transform data. Bundling all these in the model package makes the data format and the data transformations visible.

Package **command** holds all code about creating commands and applying them to the state. It also implements BlockchainAccess, BlockchainAccessAdapter and StateUpdate as described earlier. It exposes structs to be used by the user interface to prepare commands.

Package **dao** exposes structs that can be read from the local database. It exposes an init method that initializes the local database. It also exposes a function that processes incoming Sawtooth events, but it does not register to them.

Package **blockchain** holds code to send transactions based on Command instances and maybe to poll the status of transactions. It also subscribes to Sawtooth events.

Package **processor** holds the executable that is registered to Hyperledger Sawtooth as the transaction processor.

Packages **major**, **client** and **portal** hold the executables for the Major Tool, the Client Tool and the Portal.