FireFly Updates

Community Call

Date: Wednesday 14th Dec 2022

HYPERLEDGER FIREFLY

Antitrust Policy Notice

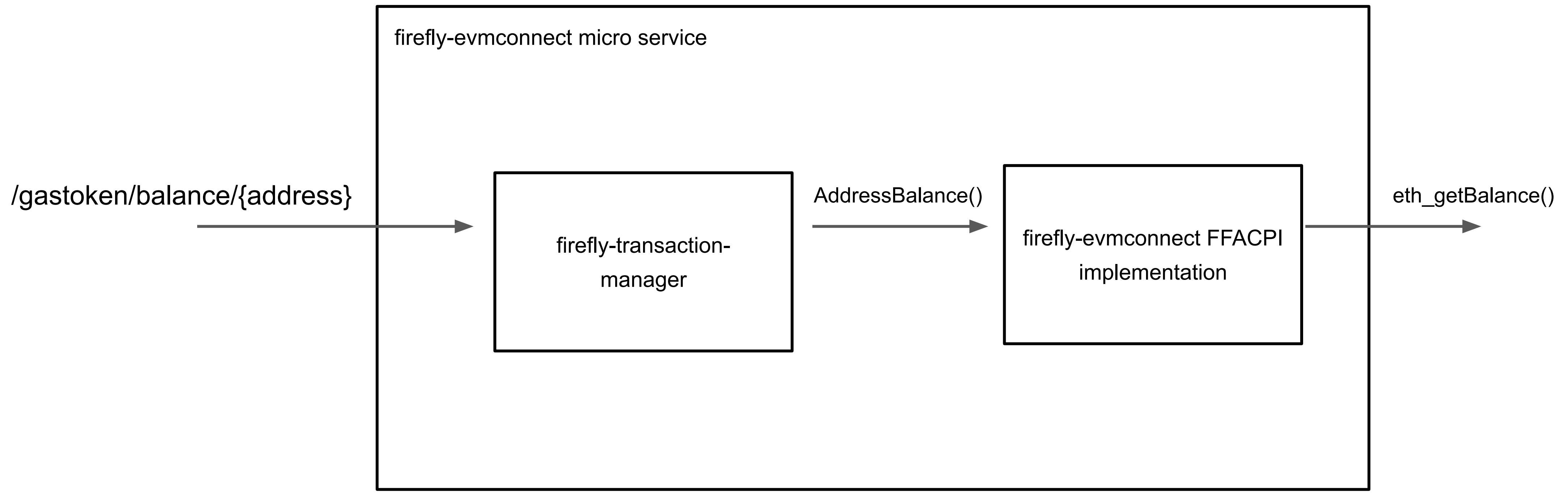
Linux Foundation meetings involve participation by industry competitors, and it is the intention of the Linux Foundation to conduct all of its activities in accordance with applicable antitrust and competition laws. It is therefore extremely important that attendees adhere to meeting agendas, and be aware of, and not participate in, any activities that are prohibited under applicable US state, federal or foreign antitrust and competition laws.

Examples of types of actions that are prohibited at Linux Foundation meetings and in connection with Linux Foundation activities are described in the Linux Foundation Antitrust Policy available at http://www.linuxfoundation.org/antitrust-policy. If you have questions about these matters, please contact your company counsel, or if you are a member of the Linux Foundation, feel free to contact Andrew Updegrove of the firm of Gesmer Updegrove LLP, which provides legal counsel to the Linux Foundation.

Agenda

1. New FFCAPI for querying blockchain balances
2. Improving efficiency of blockchain receipt handling
3. Proposal for Policy Engine changes

1. FFCAPI - New **AddressBalance()** function


firefly-evmconnect micro service
/gastoken/balance/{address}
firefly-transaction-manager
AddressBalance()
firefly-evmconnect FFACPI implementation
eth_getBalance()

1. FFCAPI - New **AddressBalance()** function

FFCAPI Interface (pkg/ffcapi/api.go in hyperledger/firefly-transaction-manager)

```
type API interface {

       // AddressBalance gets the balance of the specified address
       AddressBalance(ctx context.Context, req *AddressBalanceRequest) (*AddressBalanceResponse, ErrorReason, error)

       // BlockInfoByHash gets block information using the hash of the block
       BlockInfoByHash(ctx context.Context, req *BlockInfoByHashRequest) (*BlockInfoByHashResponse, ErrorReason, error)

       // BlockInfoByNumber gets block information from the specified position (block number/index) in the canonical chain currently known to the local node
       BlockInfoByNumber(ctx context.Context, req *BlockInfoByNumberRequest) (*BlockInfoByNumberResponse, ErrorReason, error)

       // NextNonceForSigner is used when there are no outstanding transactions for a given signing identity, to determine the next nonce to use
       NextNonceForSigner(ctx context.Context, req *NextNonceForSignerRequest) (*NextNonceForSignerResponse, ErrorReason, error)

       ...
}
```

1. FFCAPI - New **/gastoken/balances/{address}** route

FFTM Route (pkg/fftm/route_get_address_balance.go in hyperledger/firefly-transaction-manager)

```
return &ffapi.Route{
        Name:    "getBalance",
        Path:    "/gastoken/balances/{address}",
        Method: http.MethodGet,
        PathParams: []*ffapi.PathParam{
                {Name: "address", Description: tmmsgs.APIParamSignerAddress},
        },
        ...
        JSONHandler: func(r *ffapi.APIRequest) (output interface{}, err error) {
                return m.getLiveBalance(r.Req.Context(), r.PP["address"])
        },
}
```

API path calls out that the balance is related to the gas currency for the chain, nothing to do with tokens/contracts

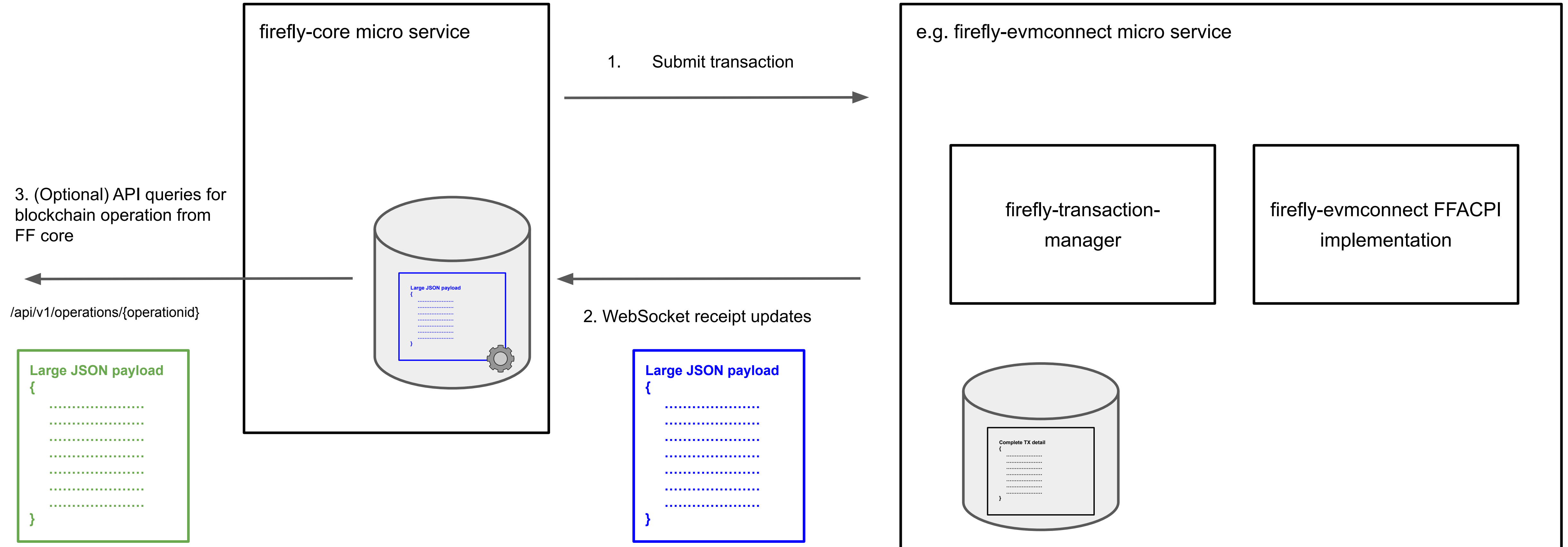
1. FFCAPI - **AddressBalance()** implementation

evmconnect implementation (internal/get_address_balance.go in hyperledger/firefly-evmconnect)

```go
func (c *ethConnector) AddressBalance(ctx context.Context, req *ffcapi.AddressBalanceRequest) (*ffcapi.AddressBalanceResponse, ffcapi.ErrorReason, error) {

	var addressBalance ethtypes.HexInteger
	var blockTag = req.BlockTag
	if blockTag == "" {
		blockTag = "latest"
	}
	err := c.backend.CallRPC(ctx, &addressBalance, "eth_getBalance", req.Address, blockTag)
	if err != nil {
		return nil, "", err.Error()
	}

	return &ffcapi.AddressBalanceResponse{
		Balance: (*fftypes.FFBigInt)(&addressBalance),
	}, "", nil

}
```

2. Blockchain connector receipt handling improvements

Current behaviour

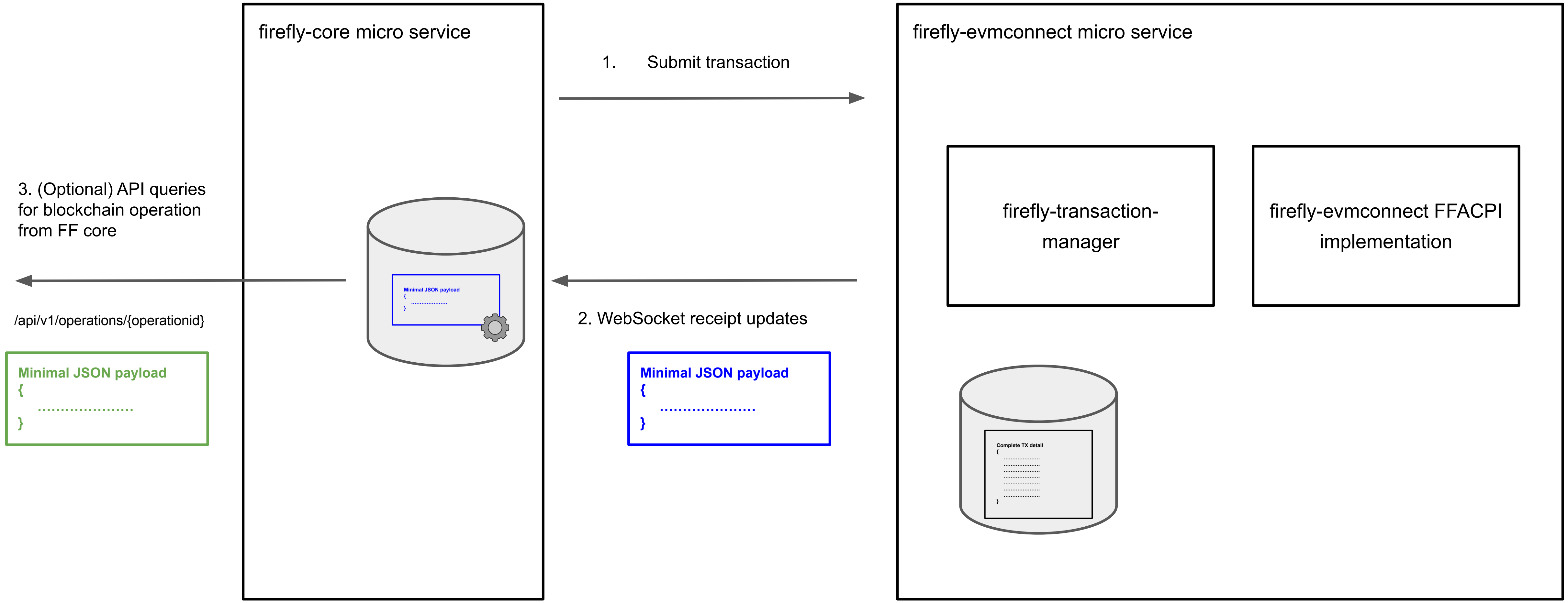

firefly-core micro service
1. Submit transaction
e.g. firefly-evmconnect micro service
firefly-transaction-manager
firefly-evmconnect FFACPI implementation
3. (Optional) API queries for blockchain operation from FF core
/api/v1/operations/{operationid}
2. WebSocket receipt updates
Large JSON payload
Large JSON payload
Large JSON payload
Complete TX detail

Downsides

1. Unnecessarily chatty between connector and FireFly core
2. FireFly core stores a lot of extra information that might never be needed
3. Data stored by the connector is duplicated

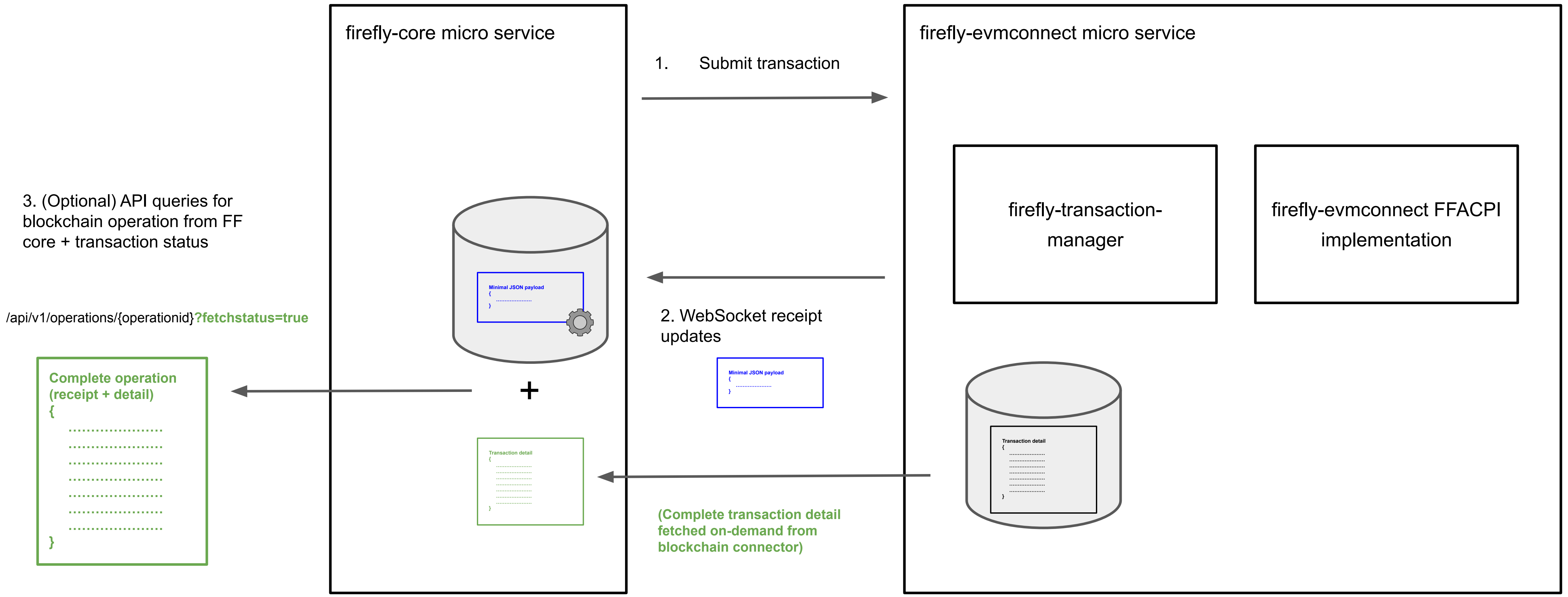
2. Blockchain connector receipt handling improvements

Proposed behaviour


firefly-core micro service
1. Submit transaction
firefly-evmconnect micro service
firefly-transaction-manager
firefly-evmconnect FFACPI implementation
3. (Optional) API queries for blockchain operation from FF core
Minimal JSON payload { .... }
2. WebSocket receipt updates
/api/v1/operations/{operationid}
Minimal JSON payload { .... }
Minimal JSON payload { .... }
Complete TX detail { .... }

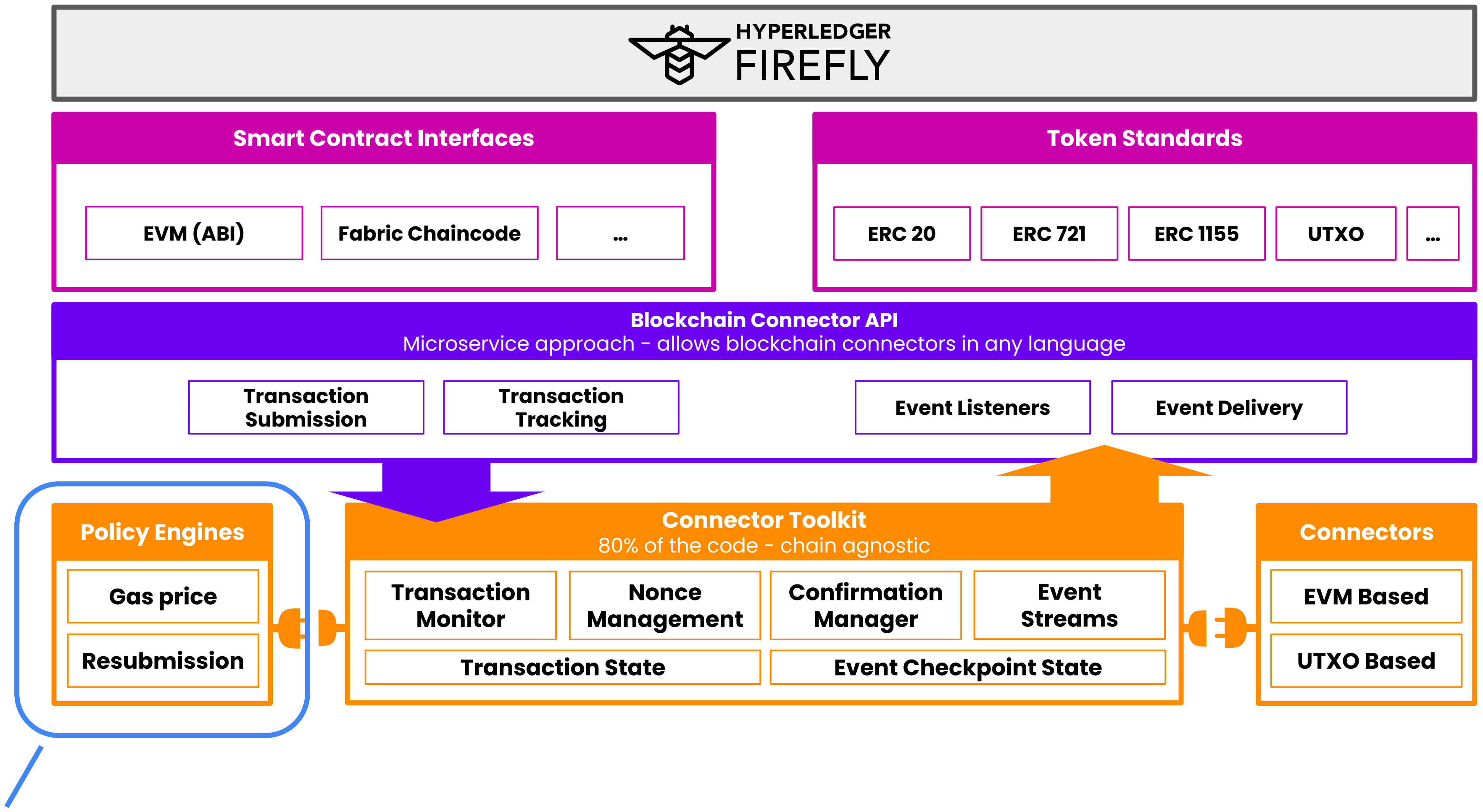
2. Blockchain connector receipt handling improvements

Proposed behaviour



firefly-core micro service
firefly-evmconnect micro service
1. Submit transaction
firefly-transaction-manager
firefly-evmconnect FFACPI implementation
3. (Optional) API queries for blockchain operation from FF core + transaction status
/api/v1/operations/{operationid}?fetchstatus=true
Minimal JSON payload
2. WebSocket receipt updates
Complete operation (receipt + detail)
Transaction detail
(Complete transaction detail fetched on-demand from blockchain connector)

3. FireFly Transaction Managed Policy Engine Changes



Pluggable policy engine

3. FireFly Transaction Managed Policy Engine Changes - Aims:

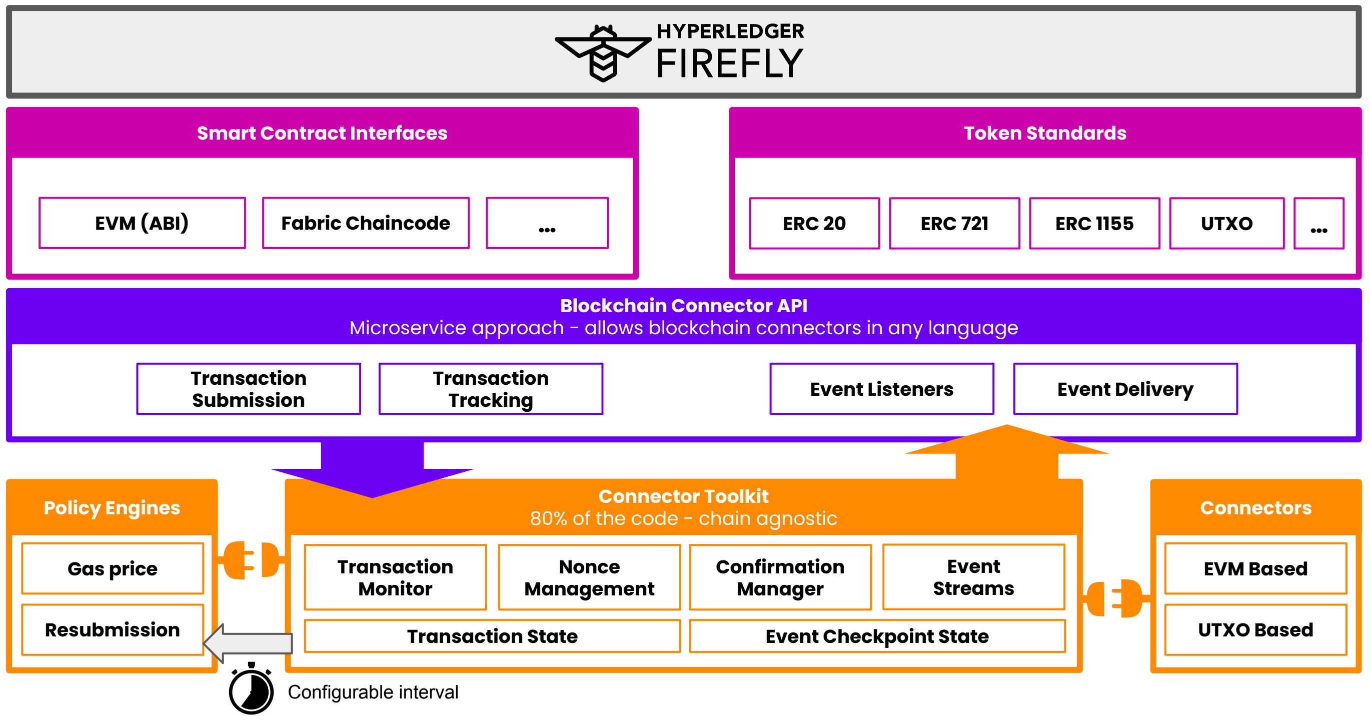
- Continue to perform common required tasks outside of policy engine:
    - Provide API endpoints
    - Receipt and confirmation tracking
    - Event stream management
        - Creation/deletion
        - Checkpointing

- Allow enough flexibility for policy engines to do:
    - More powerful nonce management
    - Efficiently dispatch granular status updates directly to consumers
    - Control the limits of new transaction buffering
    - Manage transactions at signing-key scope
        - Per-key buffer/queue
        - Sign/submit concurrently for different signing keys

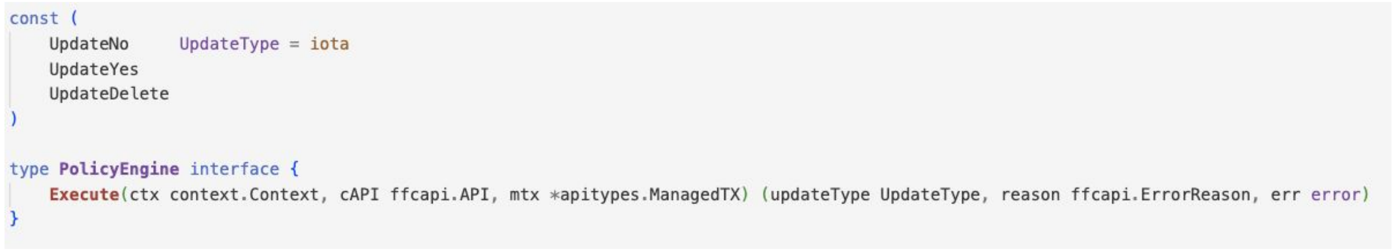
- Continue to provide a simple policy engine in the box

Characteristics of the current model

- Operates on an interval dictated by the transaction monitor
- Acts on one transaction at a time
  - Inhibits concurrent transaction processing
- Nonces can't be influenced by the policy engine
- Cannot update transaction state directly



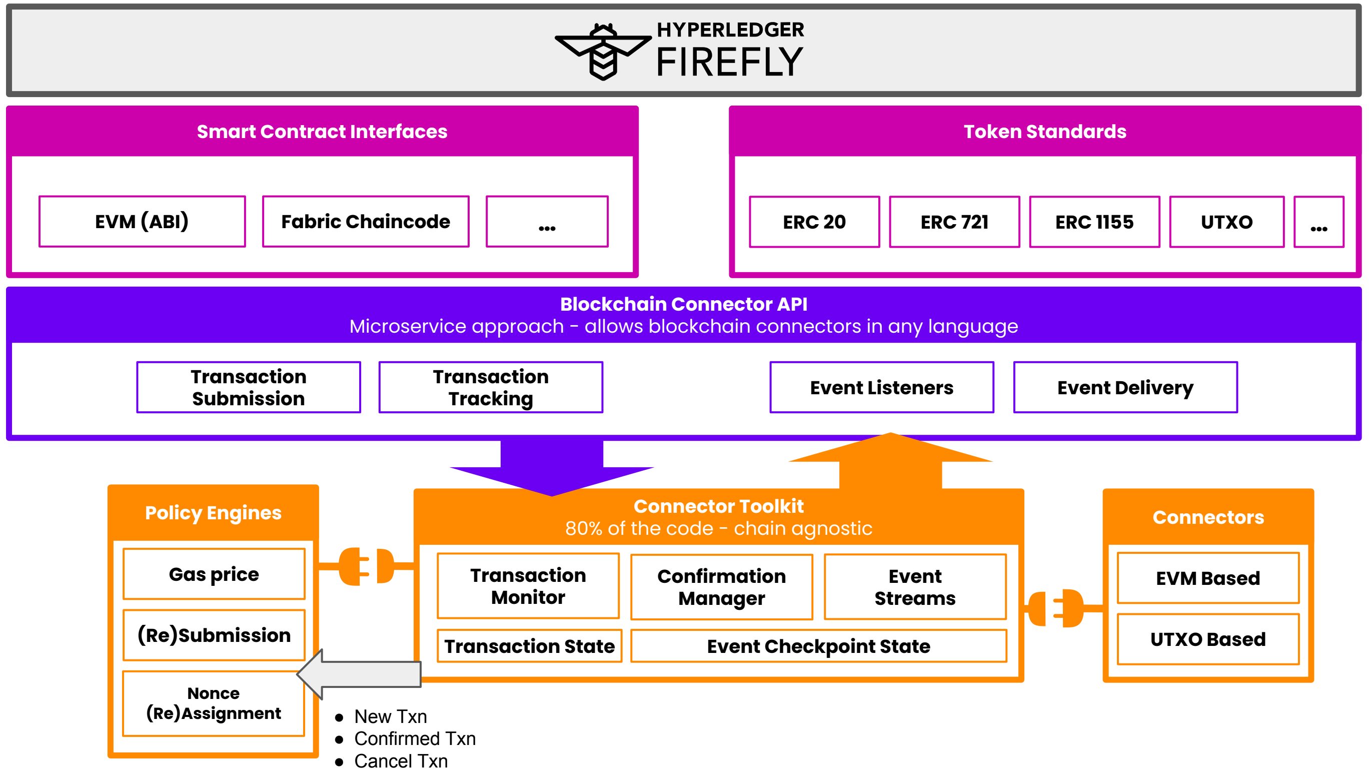
PolicyEngine Interface



```
const (
    UpdateNo     UpdateType = iota
    UpdateYes
    UpdateDelete
)

type PolicyEngine interface {
    Execute(ctx context.Context, cAPI ffcapi.API, mtx *apitypes.ManagedTX) (updateType UpdateType, reason ffcapi.ErrorReason, err error)
}
```

Proposals for a new approach

- Policy engine notified when a transaction arrives, not on an interval
- Can make decisions based on multiple transactions
- Engine is responsible for updating transaction persisted state via toolkit API
  - Transaction state no longer read only to policy engine
  - E.g. could set state=SUBMITTING before calling RPC endpoint, state=SUBMITTED after
- Can make more complex decisions about how nonaces are used
  - E.g. TXN3 hasn't been mined, cancel it and re-use its nonce for TXN4

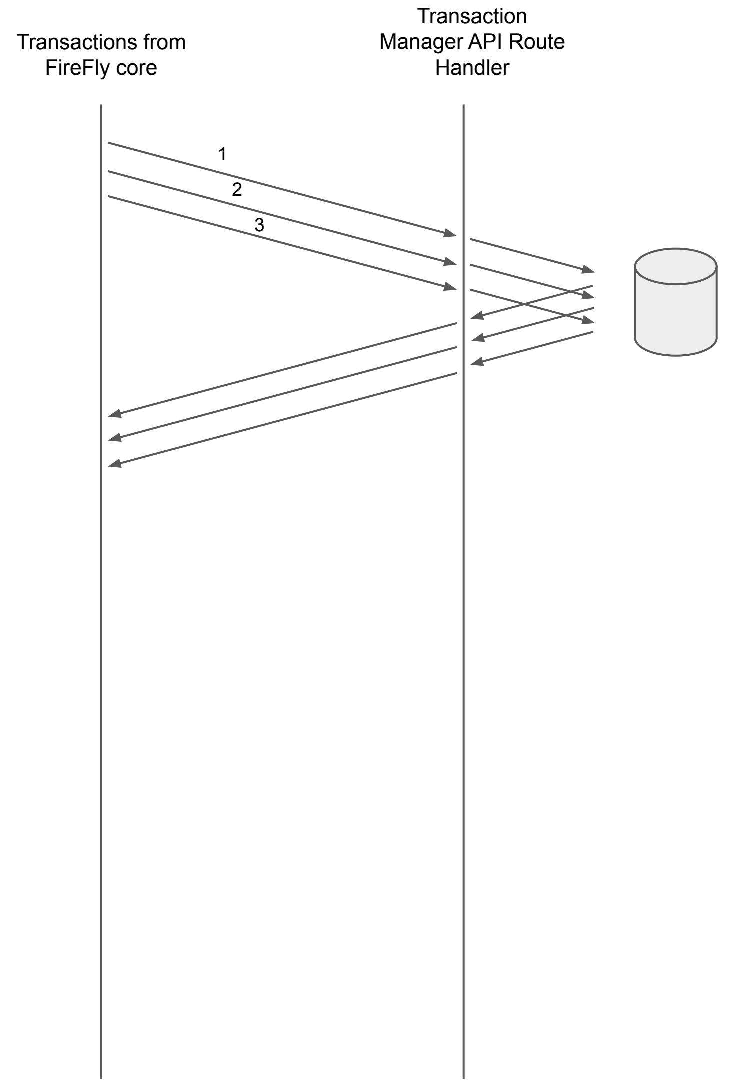


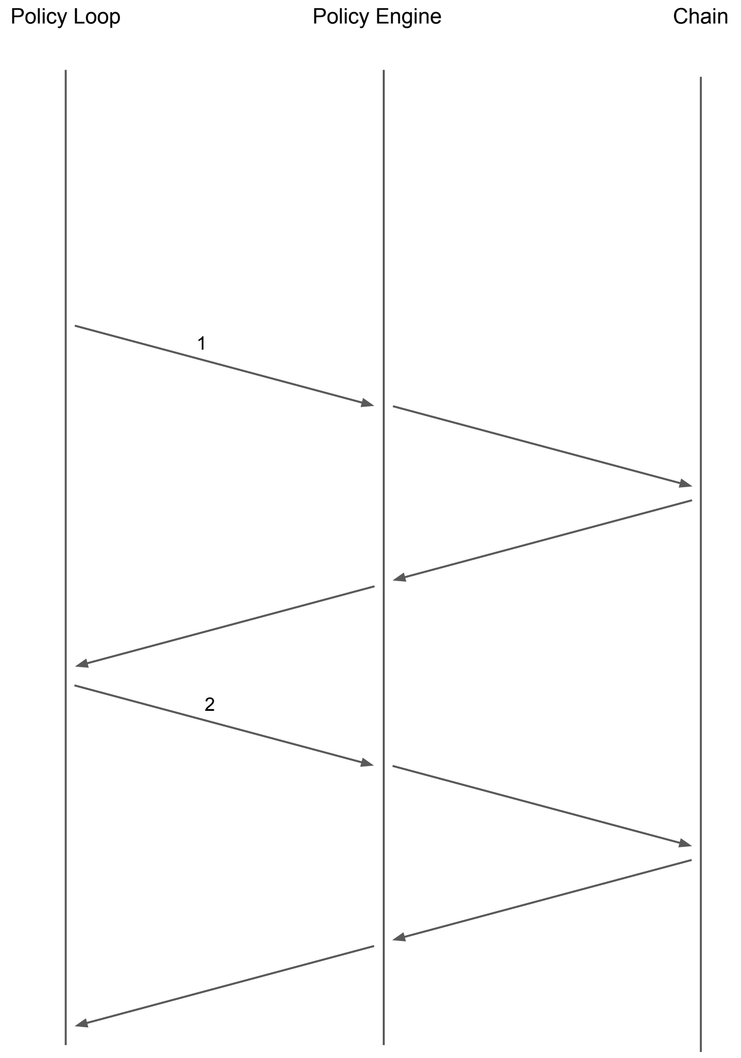
Example PolicyEngine Interface

```
type PolicyEngine interface {
        Init(toolkit *PolicyEngine.Toolkit)
        NewTransaction(mtx *ManagedTx)
        CancelTransaction(mtx *ManagedTx)
        TransactionConfirmed(mtx *ManagedTx)
}
```

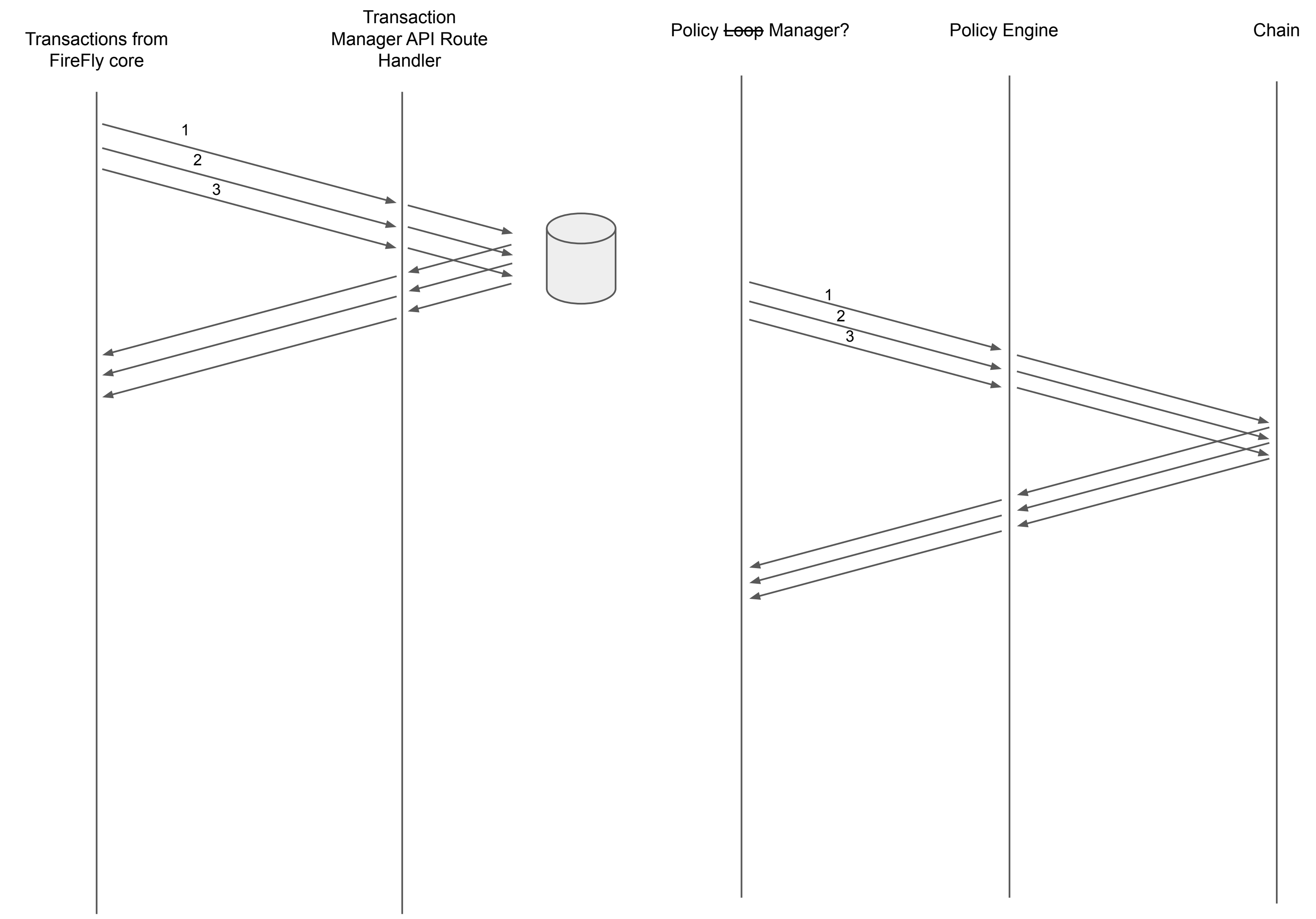
```
type Toolkit interface {
        persistence     persistence.TxnPersistenceAPI
        metrics         metrics.MetricsAPI
        statusUpdate    chan struct{}
}
```

Current threading model


Transactions from FireFly core
Transaction Manager API Route Handler
1
2
3



Threading model options with proposed changes

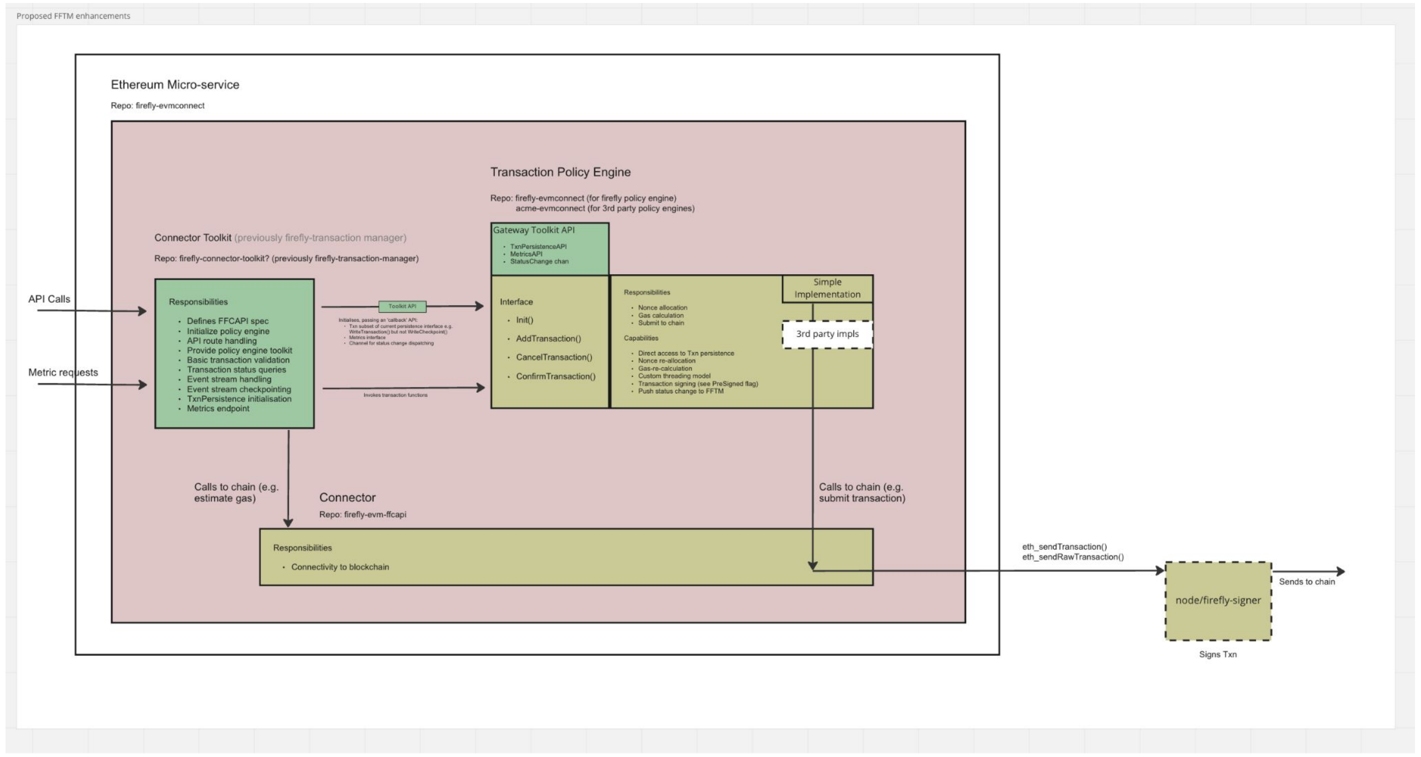

Transactions from FireFly core
Transaction Manager API Route Handler
Policy ~~Loop~~ Manager?
Policy Engine
Chain
1
2
3
1
2
3

Possibly do some refactoring as part of the same work

- Rename firefly-transaction-manager?
  - firefly-connector-toolkit?

- Move evmconnect FFCAPI implementation into its own repo?

- See FireFly contributor Miro board


Proposed FFTM enhancements
Ethereum Micro-service
Repo: firefly-evmconnect
Transaction Policy Engine
Repo: firefly-evmconnect (for firefly policy engine)
acme-evmconnect (for 3rd party policy engines)
Connector Toolkit (previously firefly-transaction manager)
Repo: firefly-connector-toolkit? (previously firefly-transaction-manager)
API Calls
Metric requests
Responsibilities
• Defines FFCAPI spec
• Initialize policy engine
• API route handling
• Provide policy engine toolkit
• Basic transaction validation
• Transaction status queries
• Event stream handling
• Event stream checkpointing
• TxnPersistence initialisation
• Metrics endpoint
Toolkit API
Initialises, passing an 'callback' API:
• Txn subset of current persistence interface e.g. WriteTransaction() but not WriteCheckpoint()
• Metrics interface
• Channel for status change dispatching
Invokes transaction functions
Gateway Toolkit API
• TxnPersistenceAPI
• MetricsAPI
• StatusChange chan
Interface
• Init()
• AddTransaction()
• CancelTransaction()
• ConfirmTransaction()
Responsibilities
• Nonce allocation
• Gas calculation
• Submit to chain
Capabilities
• Direct access to Txn persistence
• Nonce re-allocation
• Gas-re-calculation
• Custom threading model
• Transaction signing (see PreSigned flag)
• Push status change to FFTM
Simple Implementation
3rd party impls
Calls to chain (e.g. estimate gas)
Calls to chain (e.g. submit transaction)
Connector
Repo: firefly-evm-ffcapi
Responsibilities
• Connectivity to blockchain
eth_sendTransaction()
eth_sendRawTransaction()
node/firefly-signer
Signs Txn
Sends to chain

Open Discussion