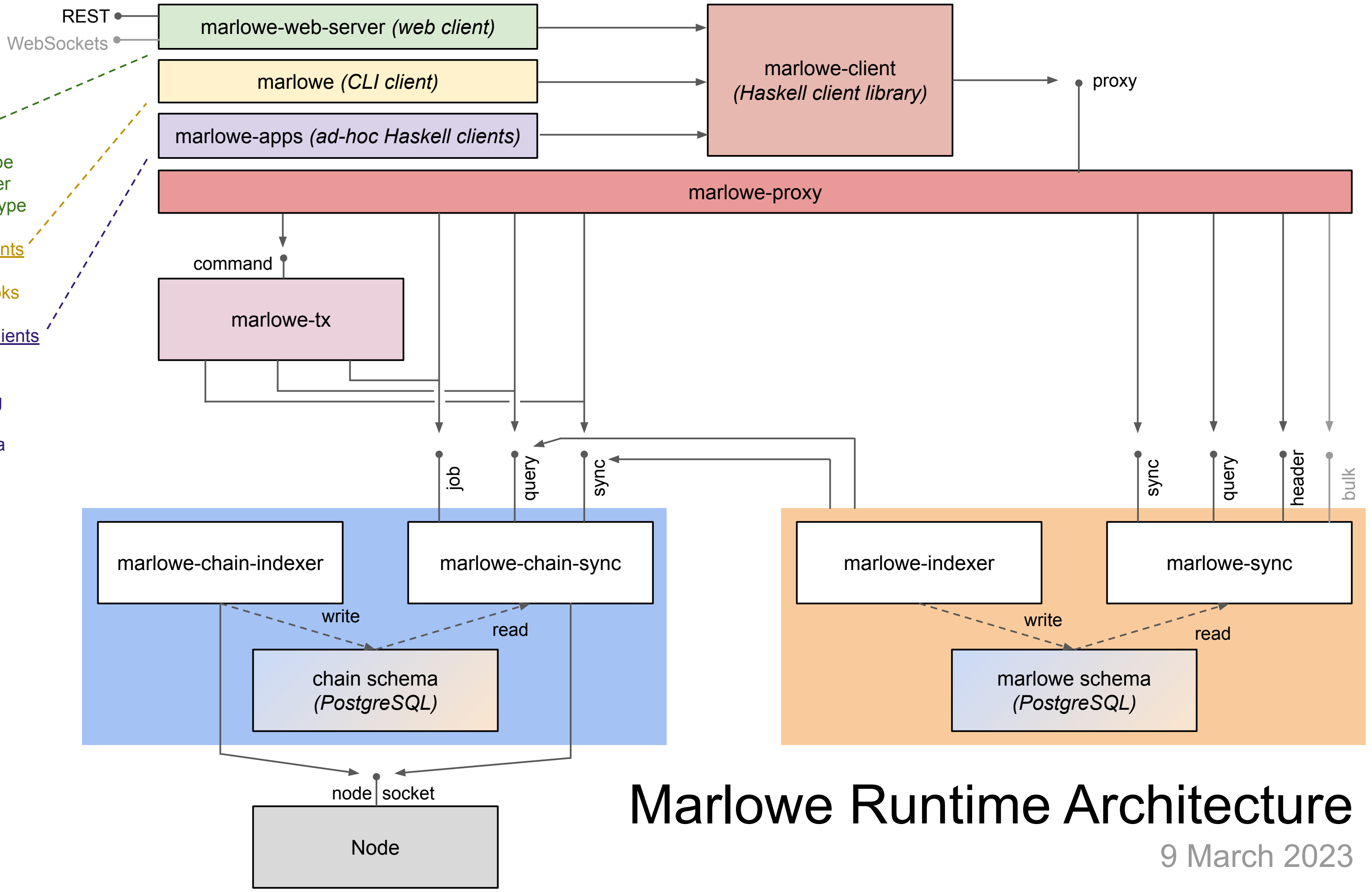
Scaling Strategies for Marlowe Runtime

21 March 2023

Abstract

The Marlowe Runtime backend consists of stateless services that index the blockchain for Marlowe transactions, that respond to clients' queries of that index, and that build new Marlowe transactions. It also includes several client-facing tools such as an optionally stateful REST server, a command-line interface, and a client library. The near statelessness of core Marlowe Runtime services enables a variety of deployment strategies where services are replicated to handle high demand. Additionally, services may utilize a single PostgreSQL database instance or replicated cluster.

These slides provide an overview of strategies for scaling Marlowe deployments and provide rules of thumb for making such deployments robust.



Web Clients
- ACTUS prototype
- Marlowe explorer
- Ada/Djed prototype

Command-line Clients
- Starter kit
- Jupyter notebooks

Special-Purpose Clients
- marlowe-pipe
- marlowe-finder
- marlowe-scaling
- marlowe-oracle
- marlowe-lambda

Marlowe Runtime Architecture

9 March 2023

Principles of Marlowe Runtime Scaling

- A single Marlowe runtime instance can handle building and submitting as many Marlowe transactions as the network can sustain on average.
  - The Plutus execution-cost budget for a block is the ultimate limit to Marlowe throughput.
  - A single Marlowe Runtime deployment can handle generating two or three times this demand, but may be overwhelmed by excessive peak demand.
  - Infrequently, a retry may be necessary.
  - Peak unsustainable demand may exceed the local node’s memory pool.
  - Marlowe Runtime applies backpressure to requests, blocking them while resources are not available.
  - The file-handle limit for each Marlowe Runtime service should be set to twice the maximum number of simultaneous requests that it should handle.
- Backend services are nearly stateless with respect to clients.
  - Any upstream service may call any other downstream service.
  - Arbitrary composition of downstream services is feasible.
  - For page-returning protocols, the components maintain some amount of state for individual connections: when a client connects to one of the protocol ports, it establishes a persistent TCP connection, and the client and server both run a state machine to coordinate the exchange.

Tuning Considerations

- Use-case specific questions:
  - What is the peak simultaneous number of transactions that will be built by Marlowe Runtime?
  - What is the peak number of transactions that will need to be in the memory pool?
  - Will transactions be submitted to the Runtime’s node or to the wallet’s node?
  - What are the underlying computing resources?
- Client-facing “knobs” for tuning performance
  - Delay blocks or seconds between submitting a transaction and using Marlowe Runtime to build a new transaction based on its output.
  - Delay blocks or seconds between Marlowe Runtime’s building of a transaction and submitting it.
  - Retry a building then submitting transaction, with exponential back-off, up to a maximum number of retries.
  - Limit the number of simultaneous transactions being built by each Marlowe Runtime instance.

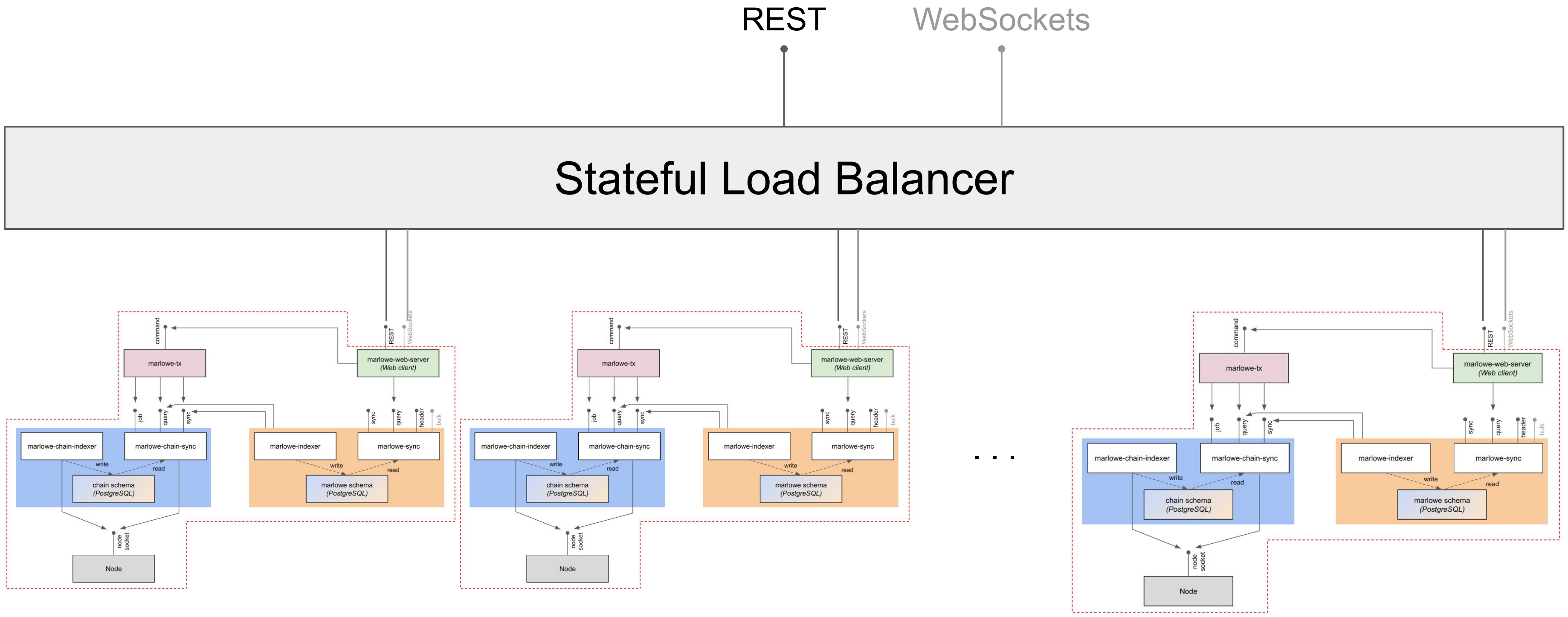
Metrics and Recommendations

- Use-case specific questions:
  - What is the peak simultaneous number of transactions that will be built by Marlowe Runtime?
    - *A single Runtime deployment can handle building ~30 transaction simultaneously.*
  - What is the peak number of transactions that will need to be in the memory pool?
    - *A node's default memory pool can hold ~60 Marlowe transactions.*
  - Will transactions be submitted to the Runtime's node or to the wallet's node?
    - *Latency between a Runtime node and a wallet's node service may be tens of seconds.*
  - What are the underlying computing resources?
    - *A single Runtime deployment can had as many simultaneous requests, provided the underlying computing resources provide sufficient is memory, sockets, and handles.*
- Client-facing "knobs" for tuning performance to reduce the likelihood of needing a retry:
  - Delay blocks or seconds between submitting a transaction and using Marlowe Runtime to build a new transaction based on its output: *~3 seconds if submitting via Runtime's node*.
  - Delay blocks or seconds between Marlowe Runtime's building of a transaction and submitting it: *~3 seconds if submitting via Runtime's node*.
  - Retry a building then submitting transaction, with exponential back-off, up to a maximum number of retries: *10 seconds to first retry with maximum of 5 retries, doubling each wait*.
  - Limit the number of simultaneous transactions being built by each Marlowe Runtime instance: throttle number of simultaneous connections to ~20.
- For page-returning protocols, the components maintain some amount of state for individual connections, so when a client connects to the load balancer, the load balancer must route all traffic for that connection to the same backing service. If the client disconnects and reconnects, it can route to a different backing service, but within the bounds of a connection, it must be the same service.

Simple Replication

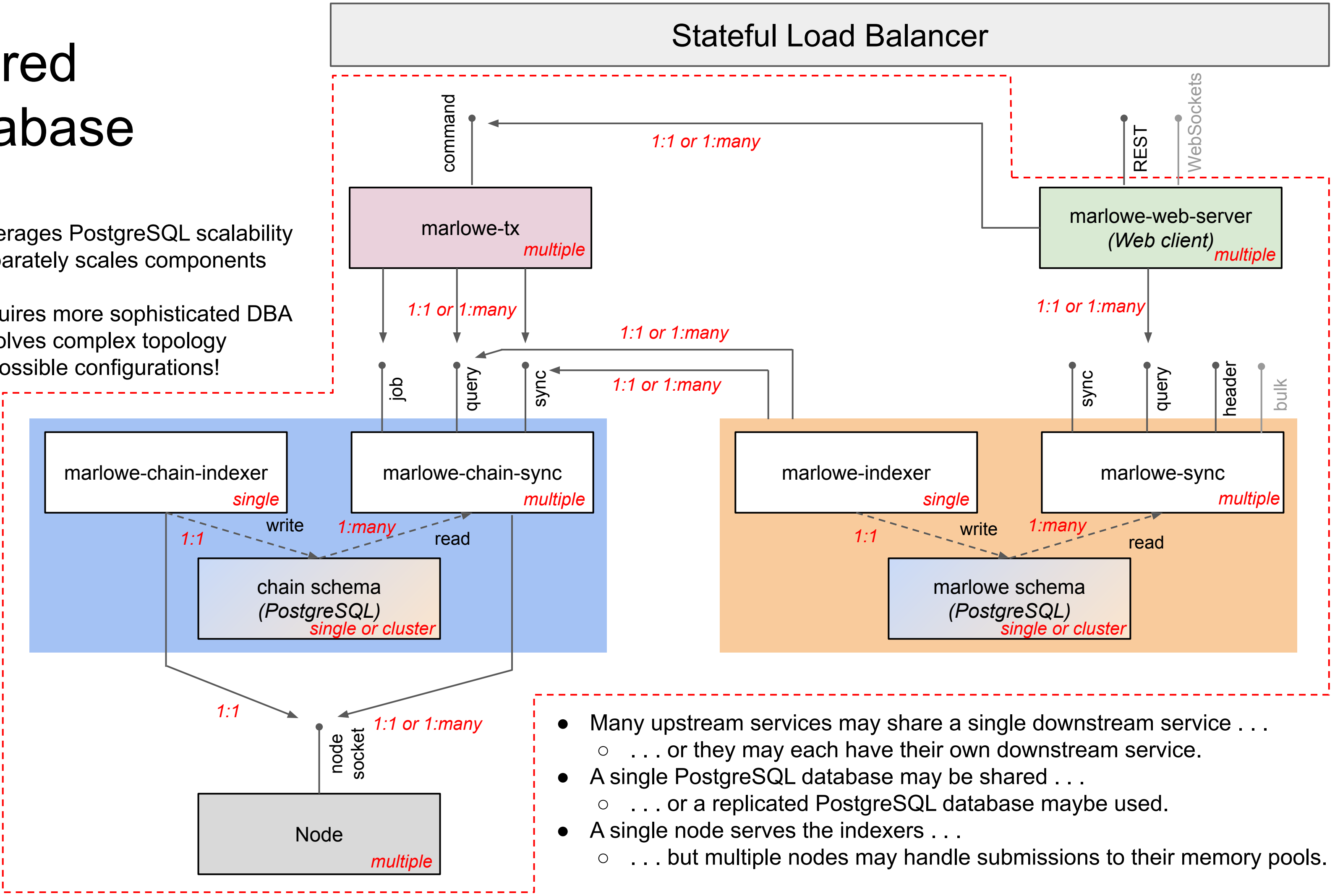
- *Pro:* straightforward to deploy
- *Con:* uses more PostgreSQL than strictly necessary

- Marlowe services are stateless with respect to clients, except for `marlowe-web-server`, where the PUT request to submit a transaction references the POST request that built it.
- If transactions are submitted outside of Marlowe Runtime, then Marlowe Runtime can be considered stateless with respect to clients.



Shared Database

- *Pros:*
  - leverages PostgreSQL scalability
  - separately scales components
- *Cons:*
  - requires more sophisticated DBA
  - involves complex topology
- Many possible configurations!



- Many upstream services may share a single downstream service . . .
  - . . . or they may each have their own downstream service.
- A single PostgreSQL database may be shared . . .
  - . . . or a replicated PostgreSQL database maybe used.
- A single node serves the indexers . . .
  - . . . but multiple nodes may handle submissions to their memory pools.

Conclusions

- A single Marlowe Runtime deployment can support as many simultaneous requests/transactions as the computing platform’s memory, sockets, and handle resources allow.
  - The file-handle limit for each Marlowe Runtime service should be set to twice the maximum number of simultaneous requests that it should handle.
- Several scaling strategies are available for handling high Marlowe demand:
  - Simple replication of Marlowe Runtime instances, supervised by an optionally stateful load balancer.
  - Pools of Marlowe Runtime services sharing a common PostgreSQL database or replicated cluster.
  - The topology for the pools of Marlowe Runtime services may be static or dynamic.
- Given the current protocol limits, the Cardano mainnet can support very roughly ~45 Marlowe transactions per minute, depending upon the type of transaction (i.e., creation, application of input, or withdrawal).