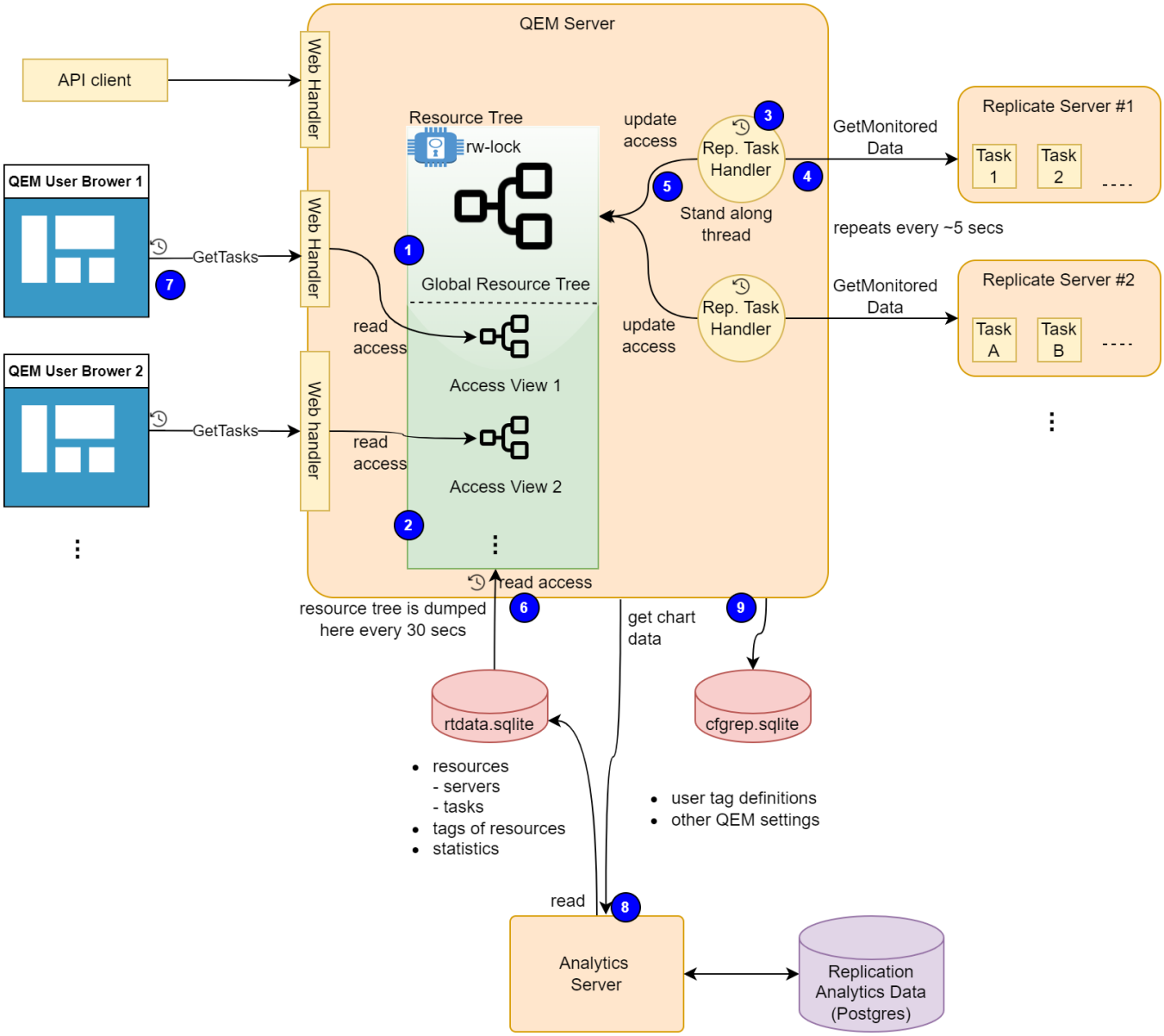
High level operation

The following QEM architecture diagram provides high-level description of how QEM operates.



The numbered markers in the diagram are:

1. The Resource Tree is QEM's main working structure. It is a sort of an in-memory database capturing the customers Replicate environment - the Replicate and Compose servers and the tasks they run ("resources"). The Resource Tree stores runtime information and statistics about the those resources as well as the tags applied to them.
2. The Access Views are subsets of the Resource Tree that contain only the resources that are viewable by currently logged-on users. The Access Views help accelerate responses to the QEM web app queries as well as to the QEM public API. All users who have the same set of access control permissions across all servers and tasks use the same Access View - the more different subsets of users exist with different access control settings, the more Access View instances will be maintained while those users are connected. Those views are updated along with the entire Resource Tree when changes from Replicate servers are applied.
3. The Replication Task handler is a worker thread that is created for every Replicate server that is currently being monitored by QEM. This thread wakes up at a given interval (5 seconds by default), (4) queries the Replicate server it monitors about the status of its active replication task, and then (5) it updates the Resource Tree with that up-to-date information.
4. Get information about active tasks from Replicate
5. Update Resource Tree with information about active replication tasks.
6. The Resource Tree is backed up regularly (every 30 seconds by default) to a SQLite database file named rtdata.repo in the QEM data folder. When QEM starts, it restores its state from that rtdata.repo database. In addition to dynamic runtime data, this database also stores the tags assigned to different resources (as that is also part of the data maintained in the Resource Tree). This on-disk backup of the Resource Tree is also used by the Analytics server as explain in (8) below.

7. QEM users run the QEM web application that retrieves, on regular intervals (currently every 5 seconds), information about all the replication tasks that are part of the Access View associated with the logged-on user as explained in (2) above. The QEM web application updates its internal structures based on the data returned from QEM and offers low latency view of changes in the system.
8. The Analytics Server keeps a watch on the rtdata.sqlite file in the QEM data folder and every time it changes, the Analytics Server reads it and update data stored in the Postgres database. When users want to see analytics charts about the activity in the monitored servers and tasks, QEM forwards the request for data to the Analytics Server which runs the required queries against the Postgres database and return the data to QEM which returns it to the QEM web application in the browser for rendering.
9. QEM also maintains a SQLite configuration database named cfgrepo.sqlite in the QEM data folder. This database maintains all the other settings of QEM that are not dynamic (including, for example, the definition of user tags, network settings, email settings, audit settings, logging settings, etc.)

Performance considerations

QEM is a highly interactive and user friendly design, monitoring and control system for Replicate and Compose servers. It uses an in memory Resource Tree structure to provide information in an efficient manner to connected QEM web application users.

Still, every system has its limits and QEM is no exception. The following points should be considered when troubleshooting performance issues with QEM.

1. The QEM Resource Tree is a shared structure protected by read-wrote locks. The moment there's a need to update this structure, a write-lock request is made and from that point, all new read requests will be blocked, when all pending read requests complete, the write-lock will be made and the structure will be updated (e.g. based on new task information from Replicate). When the update completes, the write-lock is released and then all of the other threads that were trying to read the structure will be able to do so.
2. As the number of Replicate servers managed by QEM increase, there will be more and more threads trying to update the Resource Tree using write-locks. When the time it takes to update data from a single Replicate server times the number of monitored servers exceed the GetMonitoredData calls (see (4) above), the system will be updating the Resource Tree all the time, starving threads that read the Resource Tree and resulting in exceeding long delays for the GetTasks requests. When this happends, it is recommended to increase the interval between successive GetMonitoredData calls.
3. The number of QEM web app clients (and API clients, to a much lesser extent) also has impact on the time it takes to return data from GetTasks requests but the sensitivity here generally low as long as time it takes to create the response data for the GetTasks requests, times the number of QEM user windows, divided by the number of available CPU cores is shorter than the QEM web application GetTasks polling interval minus the combined ReourceTree write time during that interval.