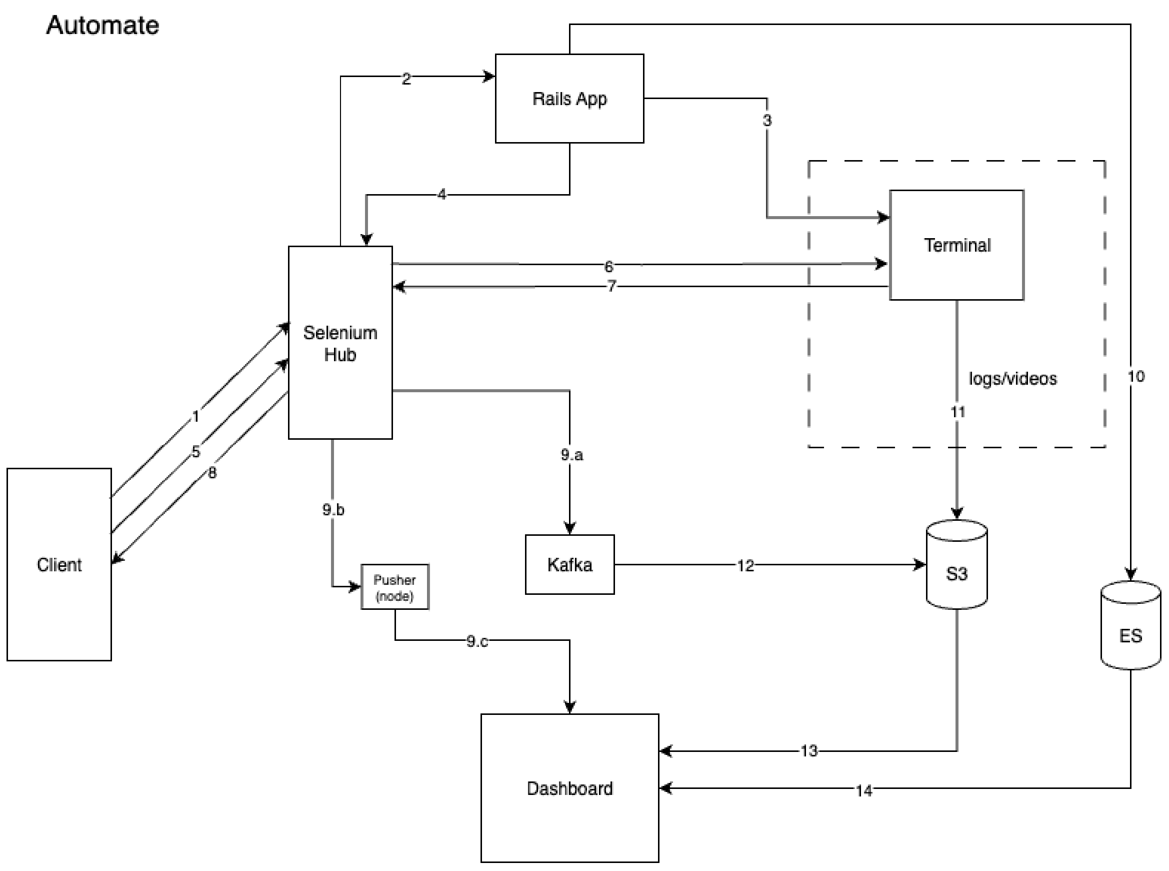
Automate & App Automate Architecture

▼ Web Automate

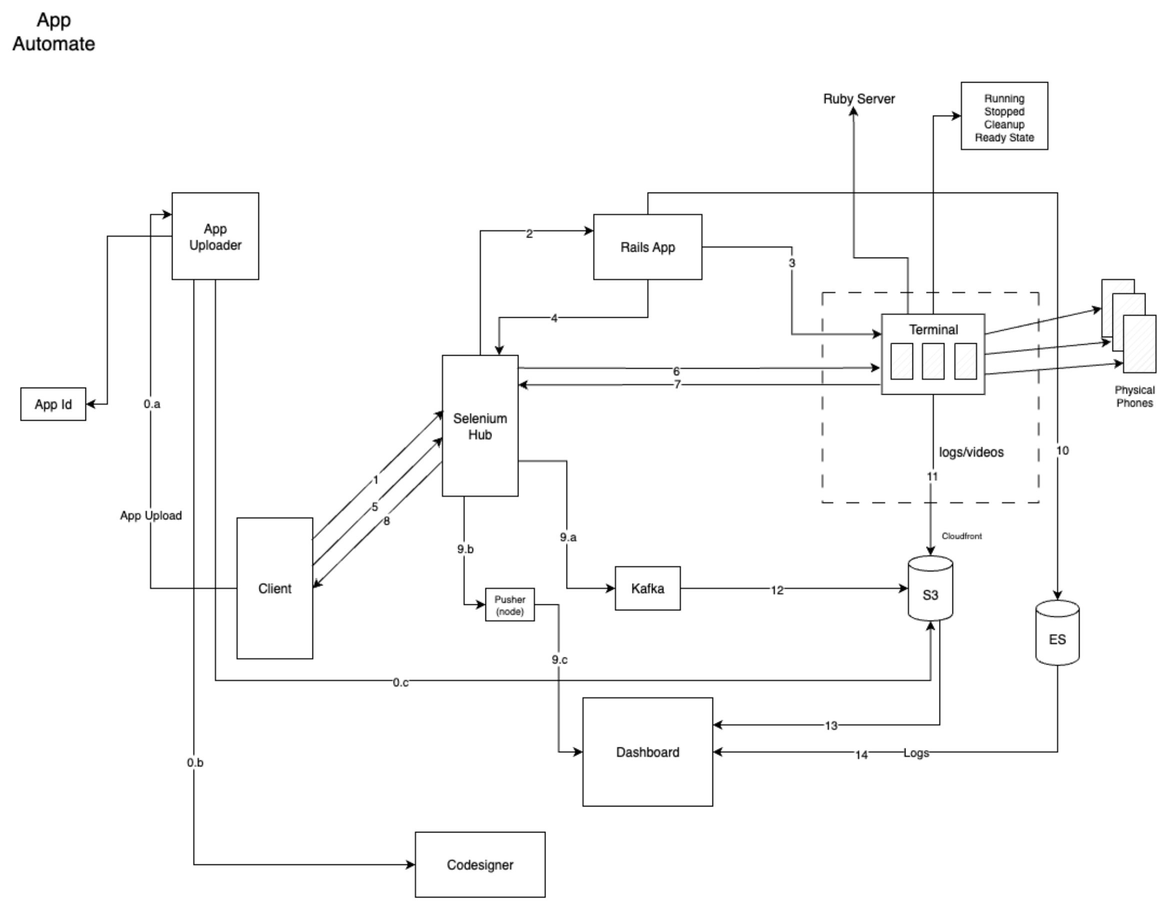


1. Client requests a terminal on server to be allocated for current session.
2. Selenium Hub routes the request to rails app which looks for available terminal on server.

3. Rails app sends signal to prepare the terminal for the session. Terminal prepares the new session for current execution, and reserves the hardware for the current session.
4. Rails app allocate a terminal, create a session id and sends it to Selenium Hub, indicating hardware is reserved and referenced session id is created.
5. Client can send command to be executed on app installed on terminal, all the commands are in standard format specified by Selenium. All commands are routed through Selenium Hub.
6. Based on session id from user, Selenium Hub sends the signal to appropriate terminal in Selenium specified APIs.
7. Command is received by selenium server running on terminal, it executes the command in app and return the response in specified format to Selenium Hub.
8. Selenium Hub routes the received response back to client indicating the command is executed on server.
9. -
    a. Logs are pushed from Selenium Hub to Kafka stream.
    b. Logs are pushed from Selenium Hub to Dashboard through Pusher service for live updates on tests using WebSockets.
    c. Pusher establishes WebSocket connection with Dashboard and sends the live data from Selenium Hub to Dashboard.
10. Rails app gets the log stream and pushes it to elastic search with corresponding session id.
11. All debug logs generated by app are recorded, on completion of session these logs and videos/pictures recorded by ffmpeg on terminal are sent to S3 for permanent storage.
12. Selenium logs from Kafka stream is also pushed to same S3 location for permanent storage.
13. User can log in to dashboard which can query S3 to receive complete formatted data related to the current session.

14. Data from ElasticSearch is also sent to Dashboard for users to see the session data.

▼ App Automate



- Codesigner:
  - This component is responsible for signing .ipa (iOS applications) with developer certifications so that it can be installed on iOS devices on servers (terminals).
  - This component is also responsible for patching applications both Android and iOS (.apk and .ipa) with images, etc.
- App Uploader:
  - Uploads the application files and asynchronously triggers code signing workflow for applications. After signing and patching, it uploads the

updated image on S3 for terminals to download the app during the test suites.

0. -
    a. Client uploads the app build file (.apk or .ipa) to the app uploader service.
    b. App Uploader service asynchronously sends the build file for patching and code signing to Codesigner.
    c. On completion of step Codesigning and Patching, the updated build file is uploaded to S3 which can be later used by Terminals to download and install the applications to execute the tests.
1. Client requests a terminal on server to be allocated for current session.
2. Selenium Hub routes the request to rails app which looks for available terminal on server.
3. Rails app sends signal to prepare the terminal for the session. Terminal prepares the new session for current execution, and reserves the hardware for the current session.
4. Rails app allocate a terminal, create a session id and sends it to Selenium Hub, indicating hardware is reserved and referenced session id is created.
5. Client can send command to be executed on app installed on terminal, all the commands are in standard format specified by Selenium. All commands are routed through Selenium Hub.
6. Based on session id from user, Selenium Hub sends the signal to appropriate terminal in Selenium specified APIs.
7. Command is received by selenium server running on terminal, it executes the command in app and return the response in specified format to Selenium Hub.
8. Selenium Hub routes the received response back to client indicating the command is executed on server.
9. -

    a. Logs are pushed from Selenium Hub to Kafka stream.
    b. Logs are pushed from Selenium Hub to Dashboard through Pusher service for live updates on tests using WebSockets.
    c. Pusher establishes WebSocket connection with Dashboard and sends the live data from Selenium Hub to Dashboard.

10. Rails app gets the log stream and pushes it to elastic search with corresponding session id.
11. All debug logs generated by app are recorded, on completion of session these logs and videos/pictures recorded by ffmpeg on terminal are sent to S3 for permanent storage.
12. Selenium logs from Kafka stream is also pushed to same S3 location for permanent storage.
13. User can log in to dashboard which can query S3 to receive complete formatted data related to the current session.
14. Data from ElasticSearch is also sent to Dashboard for users to see the session data.