CLI Sandbox DevOps Tool Design Doc: Megha Intern Project

Problem Statement

The Amplify Sandbox interface currently lacks key developer tools within the local per-developer environment that the sandbox provides. The issues with the current terminal-based experience include:

- **Current sandbox tools do not allow dynamically changing configuration/inputs:** enabling/disabling lambda function logs, and selecting which functions to stream logs require restarting sandbox
    - There is no way on the command line to text individual lambda functions with specific inputs
- **Function log multiplexing:** when you stream logs from multiple lambda functions they are multiplexed, making troubleshooting less streamlined
- **Seed is hard to use**. Customers need to understand how to author their seed script based on their data model and user attributes.
- **Outputs in the terminal are not easy to navigate**: Because the terminal is text-based, this prevents us from using better visualizations which could lead to a more efficient user experience
    - This also poses accessibility concerns
- **Log streaming is limited:** In the existing experience, there is no way to stream the CloudWatch logs of many resources, and only Lambda logs are visible via the terminal (Feedback received from survey).

Some of these issues are solved with the Amplify console, which allows individual testing of functions and provides a list of resources. However, this provides its own issues:

- **Disjoint developer experience:** viewing resources and testing functions on the console requires the developer to go back and forth between tools to test functions and view outputs
- **Poor resource visibility:** Finding the corresponding resources in AWS Console is difficult as the names of resources don't align 1:1 with what customers see on AWS Console

This brings us to the key issue:

- **Amplify console is NOT per-developer and does not track real-time changes:** it requires app to be deployed, rather than a sandbox tool which tracks local changes in real time
    - Re-deploying the whole app for every small change can be slow and expensive
    - Inhibits functionality of developers having separate sandboxes to begin with if all changes need to be deployed to see information

Overall conclusion: **sandbox lacks a local analog for Amplify console.** Thus, our goal is to create a web based DevTools which improves on the functionality of existing command-line developer tools and mimics some of the missing functionality of the Amplify console, but for the local sandbox.

Goals

1. **Unified Debugging Interface**: Create a centralized dashboard that aggregates and organizes logs from all Amplify services (Functions, Auth, Data) with advanced filtering, search, and visualization capabilities.
2. **Interactive Resource Manager**: Develop a visual representation of backend resources that clearly maps local resource configurations and deployments to AWS services and provides direct links to relevant AWS Console pages. This should show an intutive hierarchy of resources
3. **Per-resource logging:** Cloudwatch logs from all resources should be accessible and developers should be able to

dynamically toggle streaming these logs

4. **Streamlined Lambda Function Testing:** Provide a clean interface to test lambda functions with inputs, including selective log streaming with real-time updates to the functions
5. **[Out of Scope] Data Seeding Script Generation:** Allow developers to quickly add data to their application as well as download a seedable script OR record their seed operations that can be replayed again.
6. **[Out of Scope] AI Assistance**: Provide an AI-powered assistant to suggest fixes for errors in the console

Functional and Nonfunctional Requirements

- [P0] Running as local server
- [P0] Toggle sandbox on/off — UP FOR DISCUSSION
- [P0] Stream console output: developers should still be able to see anything that would print to the sandbox console in terminal
- [P0] Resource management and linking: all resources should be organized in a logical hierarchy, and should link back to their pages in the AWS console
- [P0] Lambda function logging: developers should be able to stream logs of any lambda function, with ability to dynamically toggle logs on/off, and organize logs by function
- [P0] Cloudwatch logs for all other resources: similarly, developers should be able to dynamically toggle logs on/off and view logs of different resources in different tabs/windows
- [P1] Look and feel of the UI should be either according to Amplify colors OR AWS Console.
- [P1] Lambda testing with free form input: developers should be able to test lambda functions in an isolated environment, and enter inputs to these functions as a free-form text box
- [P2] Listing all sandboxes within this project, and giving option to toggle each ON/OFF (Note only 1 sandbox can be running at once)
- [P3] Stream Cloudtrail logs
- [P3] Data seeding script generation: Given data input, tool should detect model and schema and create a script that developers can use to seed their app with that data

Assumptions

- DevTools will be used primarily by developers who are already familiar with Amplify and its sandbox environment
- Users have a modern web browser that supports WebSockets and modern JavaScript features
- The tool will be used on machines with sufficient resources to run both the Amplify sandbox and a local web server
- DevTools will not replace the existing terminal-based sandbox but will complement it
- DevTools will not require any changes to the user's Amplify project structure or configuration

High Level Design

Infrastructure

The key design decision is to move these tools **out of the terminal** and into a **local browser-based experience**. This makes sense over the terminal because:

- Allows us to implement more intuitive UI with a shallower learning curve for reaching different tools
- Allows developers to maintain “tabs” of different information (i.e. to test multiple lambda functions at once) without

switching between windows

Since the sandbox is local, local hosting makes sense over a web-based tool since we don't want to take on any more infrastructure than necessary, and mitigates any potential security concerns.

Processes
There are two key processes here: the sandbox, and the DevTools console.

The **sandbox** process is a collection of your backend resources in a local context, which maintains isolation between different developers working on the same project.

- When the sandbox process is "stopped," resources are NOT deleted, and if it is started again, it will return to those same cloud resources

The **DevTools console** is an interface that provides tools for interacting with the locally deployed backend, providing a visual representation of all the resources in your Amplify project, improved log viewing capabilities by resource and isolated lambda function testing.

Web Server: It runs an Express server that serves a web application interface accessible through a browser.

Real-time Communication: It uses Socket.IO to establish WebSocket connections for real-time updates between the UI and backend.

Resource Visualization: It provides a visual representation of all the resources in your Amplify project, organized in a logical hierarchy.

Design Question: DevTools/Sandbox relationship — Should DevTools be a standalone parent of sandbox, or a child of sandbox?

Option 1: DevTools as a Standalone Process (Parent of Sandbox)
In this model, DevTools is an independent process that can start, stop, and manage the Sandbox.
Pros:

- Clear separation of concerns: DevTools manages the UI and user experience, while Sandbox manages AWS resources
- More flexible lifecycle management: DevTools can run without Sandbox for certain features
- Allows for future expansion where DevTools might manage multiple sandboxes or other processes

Cons:

- More complex implementation: Requires DevTools to understand how to start/stop Sandbox with all possible options
- Credential management complexity: DevTools may need to handle AWS credentials for starting Sandboxes
- Potential process management issues: DevTools must reliably track and manage the Sandbox child process
- Potential redundancy: DevTools is MOST useful when sandbox is running— if there is no sandbox launched, much of the functionality is already provided by AWS console

Option 2: DevTools as a Child Process (Started by Sandbox)
In this model, DevTools is started as an optional feature of Sandbox (suggests starting with `npx ampx sandbox --devtools`).
Pros:

- Simple implementation: Sandbox already handles configuration, credentials, and resource management
- Guaranteed resource availability: DevTools always has access to Sandbox resources

Cons:

- Limited flexibility: DevTools cannot run independently of Sandbox
- Reduced feature set: Cannot provide any functionality when Sandbox is not running

In both cases, we need to consider if the sandbox auto-starts, or if starting the sandbox needs to be done through the DevTools UI.

Auto-start Sandbox
Pros:

- Clearer mental model: DevTools and Sandbox are perceived as a unified tool
- Potential confusion: If we don't auto-start, some features will be disabled until Sandbox is started
- Cuts an additional step for users: otherwise, they must manually start Sandbox to access all features

Cons:

- If the sandbox is auto-started, there needs to be a way to pass in all the flags and options, which would need to be done from the command line. Having to use the command line for these options inhibits the original functionality of a unified UX in DevTools.
- Resource usage: Starts Sandbox even when users might only need DevTools features

Based on the analysis, I recommend:

Implement DevTools as a standalone process that can manage Sandbox, do not auto-start Sandbox but make it easy to start from the UI, and provide buttons for all possible sandbox options/settings on the UI.

Starting Developer Tools

The next design decision is whether this tool should launch with its own command or as a modifier to the existing `npx ampx sandbox` command. While having a completely new command would minimize risk of breaking backwards compatibility, simply adding a new option, such as --devtools to the existing command would also preserve all of its functionality. This has the added benefit of maximizing visibility of the new feature to users, as well as signaling that the DevTools console contains much of the same information and serves a similar purpose to the original terminal-based sandbox console. An expansion of the tradeoffs are below:

Option 1: Add as Flag to Existing Command (`npx ampx sandbox --devtools`) (used in Megha's POC)
Pros:

- Maintains intuitive connection between sandbox and its development tools, as DevTools console contains much of the same information and serves a similar purpose to the original terminal-based sandbox console
- Simpler discovery for users already familiar with sandbox command
- Aligns with philosophy of extending existing tools rather than creating new ones

Cons:

- Forces DevTools to be dependent on sandbox starting and couples lifecycle of DevTools with sandbox process
- Creates a confusing set of dependencies IF we implement start/stop functionality as devtools is a option of starting sandbox, but DevTools has ability to start and stop sandbox, making it both a parent and child process of the sandbox
- May confuse users about whether DevTools requires sandbox for all features
- Less flexibility for future features that could work without sandbox

Option 2: Create New Command (npx ampx devtools) (used in Praveen's POC)
Pros:

- Allows DevTools to operate independently of sandbox state
- Clearer separation of concerns between sandbox and development tools
- More flexible for future expansion (e.g., viewing amplify_outputs without sandbox)
- Easier to implement features that don't require sandbox

Cons:

- May seem disconnected from sandbox functionality initially, and/or could lead to confusion about when to use sandbox vs devtools
- Users need to open a new window to run **`npx ampx devtools`**

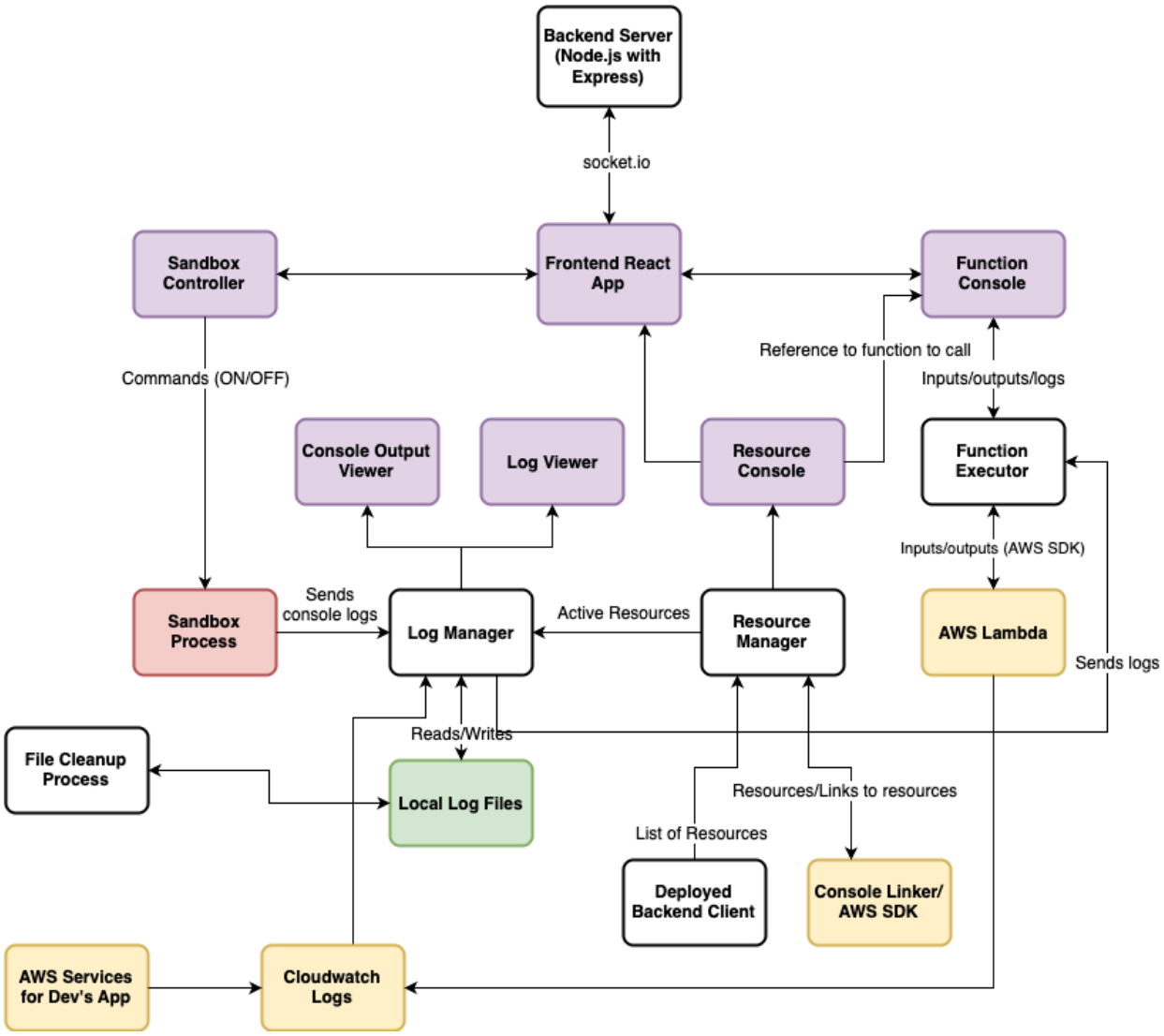
Given the feedback and long-term considerations, I recommend implementing DevTools as its own command `npx ampx devtools`

System Architecture

The Amplify DevTools consists of these main components:

- **Web Frontend (React with AWS Cloudscape)**
- **Backend server (Node.js with Express):**
  - Express server instead of http for build-in error handling and better routing as this project will likely have more complex routing needs
  - Configuration can be managed in a json file at this level
- **Resource Manager: provides hierarchy and list of resources to app, links to AWS console**
- **Log Manager: Captures logs from Cloudwatch, linked to specific resources from resource manager**
  - Will need to deal with local file cleanup and maintenance (likely using the temp folder)
- **Function Executor: interfaces with AWS Lambda and Cloudwatch to pass along inputs/outputs and logs**

Note: Function Console may be combined with Log Viewer for functions, pending UX design



Log Storage

One key design decision is whether keep a buffer of logs, or stream them directly to the front end. The main options are direct streaming or keeping buffers in local files. In each design, we assume that we record logs only when logs have been toggled ON in the DevTools console. This allows us to avoid taking on more infrastructure to buffer all logs, as for long-running processes this could amount to a lot of data. Generally, logs are used in real-time— in most development tools, logs can't be viewed retroactively. This design simply echoes that quality: logs can only be viewed from after the point when you toggled logging on. By default, all logging should be off. Either way, we should provide an option which would turn all on/off (a button on the frontend and `--start-logging=ALL/NONE` on command line).

Option 1: Pure Streaming
Pros:

- Minimal resource usage
- Simple implementation
- No cleanup required

- Zero risk of memory leaks

Cons:

- Logs lost on page refresh or connection drop
- No access or ability to search historical logs
- Poor developer experience when debugging requires log history

Option 2: Local File Storage (Recommended)
Pros:

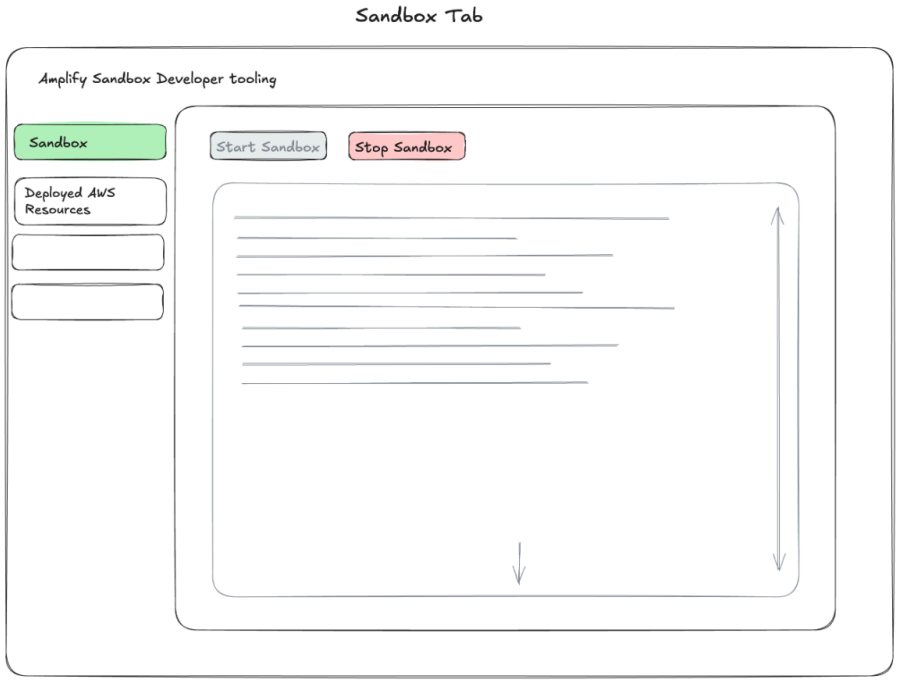
- Logs persist through page refreshes
- No additional backend infrastructure needed
- Allows searching through session history
- Memory efficient (stream from disk as needed)
- Enables additional features like:
    - Log filtering by time ranges
    - Save/share/export logs and debug sessions

Cons:

- Potential security issue of storing Cloudwatch Logs
- Requires file system management (more complex implementation)
- Need cleanup strategy for old log files
- Potential disk space concerns for very long sessions, but can be mitigated by max buffer size and good cleanup

User Experience

Mockup of what UX looks like:



Security

By design, this tool should have minimal security concerns, as all communication happens locally, mitigating most traditional web security concerns. Furthermore, Lambda function testing will execute in isolated environments, and we can use the existing Amplify credential chain without additional exposure. This leaves minimal opportunity for threats. However, regardless, we will make sure that all inputs are validated to prevent injection attacks. Will still require AppSec review.

Timeline

Week 1:

- Design UX for each screen
- Basic frontend/backend communication
- Stream console logs
- Add ability to start/stop sandbox
- Add tab for deployed AWS resources

Weeks 2-3:

- List resources in hierarchy
- Link resources back to their console pages
- Write console logs to local file
- Read console logs to frontend
- Implement file cleanup
- Build log manager — write cloudwatch logs for each resource to a local file and read to front end

Week 4-5:

- Implement function console
- Interface with AWS Lambda
- Stream logs for functions directly to testing interface
- Add documentation/cleanup

NOTES FROM DESIGN REVIEW

Problem Statement/Goals

Requirements

for logging might be worth focusing on appsync/resolvers.

- Should be up to accessibilty standards of console/existing tools
- Look and feel should match AWS Console

Design

- Security: need to sanitize customer data

Starting sandbox:
print out url on default
npx ampx sandbox devtools (subcommand) to just launch devtools