Amplify CLI Sandbox DevTools Project Offboarding Knowledge Transfer

Author: Megha Narayanan
Summer 2025

Basic Structure and Critical Links

Amplify DevTools provides a browser-based developer console for the Amplify Sandbox environment. It enables developers to:

- Monitor and control Amplify sandbox lifecycle (start, stop, delete)
- View deployed AWS resources and their configurations
- Stream and analyze CloudWatch logs in real-time
- Track deployment progress through CloudFormation events
- Test Lambda functions directly from the UI

This tool serves as a crucial developer experience enhancement for the Amplify CLI ecosystem by providing visual insights into sandbox environments that would otherwise require complex AWS console navigation or CLI commands.

The devtools project lives in the aws-amplify/amplify-backend repo, on the feature/dev-tools branch (note this is distinct from the feature_dev_tools branch, which is dead).

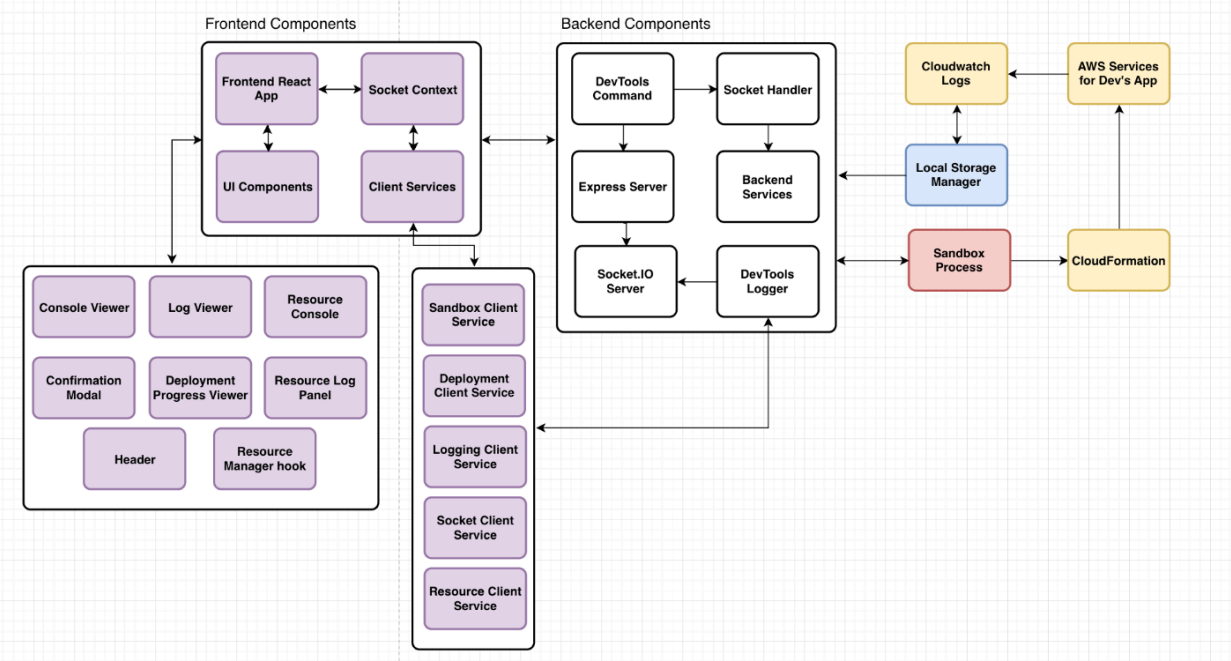
For more details of motivation, design decisions, and potential direction for future features see the original design doc here: CLI Sandbox DevOps Tool Design Doc: Megha Intern Project

GIT PULL REQUEST LINKS

DevTools was merged in this series of pull requests: (note the 3.x PRs are each off PR2, not eahcother)
PR1: Dependencies and General Structure : https://github.com/aws-amplify/amplify-backend/pull/2868
PR2: Backend Services, Console Viewer, and Resources: https://github.com/aws-amplify/amplify-backend/pull/2879
PR3.1: Frontend Testing for [Previously] Existing Components: https://github.com/aws-amplify/amplify-backend/pull/2906
PR3.2: General Structure Updates and Error Handling https://github.com/aws-amplify/amplify-backend/pull/2909
PR3.3 Local Storage https://github.com/aws-amplify/amplify-backend/pull/2911
PR3.4 Logs https://github.com/aws-amplify/amplify-backend/pull/2914
PR3.5 Deployment Tracking https://github.com/aws-amplify/amplify-backend/pull/2917
PR4: Testing https://github.com/aws-amplify/amplify-backend/pull/2918

BASIC ARCHITECTURE:

DevTools follows a client-server architecture:
- Backend: Node.js/Express server with Socket.IO for real-time communication
- Frontend: React application with CloudScape Design System components
- Communication WebSockets via Socket.IO for real-time bidirectional events
- Persistence: Local storage for settings and session data



DevTools uses a hardcoded port (3333) for its server, which creates a fundamental limitation: only one DevTools instance can run per machine, preventing users from working with multiple projects simultaneously and potentially conflicting with other applications using the same port. While this design choice restricts flexibility, it does provide the benefit of a consistent, predictable access URL (http://localhost:3333) for all DevTools sessions.

Run `npm run build` inside the `cli` directory to build the frontend and backend of DevTools.

KEY DEPENDENCIES & INTEGRATIONS

AWS Service Integrations

- **CloudWatch Logs**: For resource log streaming
- **Lambda**: For function invocation testing
- **CloudFormation**: For stack status and deployment tracking
- **Amplify**: For backend metadata and resource discovery

Frontend Libraries

- **Socket.IO Client**: Real-time communication
- **CloudScape Design System**: UI component library
- **React**: UI framework

Backend Libraries

- **Express**: HTTP server framework
- **Socket.IO Server**: WebSocket communication
- **AWS SDK v3**: AWS service clients

File Structure and Organization

These are some of the key files:

```
packages/cli/src/commands/sandbox/sandbox-devtools/
├── react-app/                  # Frontend React application
│   ├── src/
│   │   ├── components/
│   │   │   ├── ConsoleViewer.tsx     # Console log display
│   │   │   ├── DeploymentProgress.tsx # Deployment tracking
│   │   │   ├── Header.tsx            # Navigation header
│   │   │   └── ResourceConsole.tsx   # Resource management UI
│   │   ├── contexts/
│   │   │   └── socket_client_context.tsx # Socket service provider
│   │   ├── services/           # Client services
│   │   │   ├── deployment_client_service.ts # Deployment tracking
│   │   │   ├── logging_client_service.ts # Log management
│   │   │   ├── sandbox_client_service.ts # Sandbox operations
│   │   │   └── socket_client_service.ts # Base Socket.IO client
│   │   ├── integration-tests.tsx        #Frontend integration tests
│   │   └── App.tsx             #Main app component
├── services/                   # Backend services
│   ├── socket_handlers.ts      # Main socket event handler
│   ├── resource_service.ts     # Resource management
│   ├── shutdown_service.ts
│   ├── socket_handlers_logging.ts # Log-related handlers
│   └── socket_handlers_resources.ts # Resource-related handlers
├── shared/                     # Shared between client and server
│   ├── socket_events.ts        # Socket event constants
│   └── socket_types.ts
├── logging/
│   ├── cloudformation_format.ts
│   └── log_group_extractor.ts
├── local_storage_manager.ts    # Persistent storage management
├── sandbox_devtools_command.ts # CLI command implementation
└── integration-tests/          # Integration tests (for backend)
```

Custom Logging Infrastructure

The system implements a custom logging infrastructure:

- **DevToolsLogger**: Extends CLI-core printer to forward logs to clients
- **Log Persistence**: Stores logs with configurable size limits
- **Log Rotation**: Implements intelligent log pruning for older entries
- **Log Filtering**: Client-side filtering by log level and content

Key Implementation Detail: The logger is injected into the sandbox instance to capture all sandbox-related logs:

```
// In sandbox_devtools_command.ts
// Create custom logger that forwards to Socket.IO clients
const devToolsLogger = new DevToolsLogger(this.printer, io, minimumLogLevel);
this.printer = devToolsLogger; // Replace standard printer
// Get sandbox with custom logger
```

```
const sandbox = await new SandboxSingletonFactory(
  this.sandboxBackendIdResolver.resolve,
  new SDKProfileResolverProvider().resolve,
  this.printer, // Original printer for initialization
  this.format,
).getInstance(devToolsLogger); // Pass custom logger to capture all sandbox logs
```

Sandbox State Management

DevTools implements a sophisticated state management system for tracking and controlling the Amplify sandbox environment. The sandbox can exist in several different states, enumerated by the `SandboxStatus` type. tate tracking is implemented via a combination of:

1. **State Resolution Function**:

The `createGetSandboxStateFunction` method creates a specialized function that:

- Calls the sandbox's native `getState()` method
- If the state is 'unknown', it performs additional CloudFormation checks to determine the true state
- Falls back to checking CloudFormation stack existence via AWS SDK calls
- Uses stack existence as a proxy for sandbox existence (stack exists = 'stopped', no stack = 'nonexistent')

```
   createGetSandboxStateFunction(
    sandbox: Sandbox,
    backendId: BackendIdentifier,
  ): () => Promise<SandboxStatus> {
    return async () => {
      const state = sandbox.getState();
      if (state === 'unknown') {
        try {
          const cfnClient = this.awsClientProvider.getCloudFormationClient();
          const stackName = BackendIdentifierConversions.toStackName(backendId);
          const response = await cfnClient.send(
            new DescribeStacksCommand({ StackName: stackName }),
          );

          if (!response || !response.Stacks || response.Stacks.length === 0) {
            return 'nonexistent';
          }

          return 'stopped'; // Stack exists, default to stopped
        } catch (error) {
          if (String(error).includes('does not exist')) {
            return 'nonexistent';
          }
        }
      }
      return state;
    };
  }
```

1. **Event Listeners**:

The system sets up comprehensive event listeners on the sandbox instance to track all state transitions:

- `deploymentStarted`
- `successfulDeployment`
- `deletionStarted`
- `successfulDeletion`
- `failedDeployment`/`failedDeletion`
- `successfulStop`/`failedStop`
- `initializationError`

For each event, the handler:

- Logs the event
- Checks current sandbox state
- Constructs appropriate status data
- Emits socket events to all connected clients

State Propagation:
The state is propagated through Socket.IO events to all connected clients, with additional metadata
Frontend State Management:
In the React application, the `App.tsx` component:

- Maintains local state with `useState<SandboxStatus>('unknown')`
- Subscribes to sandbox status events from the server
- Updates UI based on state changes
- Implements state-specific behaviors (e.g., disabling buttons when deploying)
- Periodically polls for status if in 'unknown' state

Critical Considerations:

- State transitions aren't always perfectly predictable, so the system includes validation and warning logs if unexpected transitions occur
- DevTools cannot manage a sandbox which was started in the command line because the event listeners would not be connected — the sandbox must be started from devtools to be managed by DevTools. The core issue stems from how the sandbox instance is created and managed:
    - When started through DevTools (`npx ampx sandbox devtools`), a single sandbox instance is created by the `SandboxSingletonFactory` and maintained within the DevTools process.
    - When started through the command line (`npx ampx sandbox`), a separate, independent sandbox instance is created in a different Node.js process. These are completely separate instances without any communication channel between them.
    - Even though DevTools tries to determine the sandbox state through CloudFormation checks, it cannot reliably track state transitions of an externally managed sandbox. For example, if a deployment is in progress in a command-line sandbox, DevTools might see the state as 'deploying' but won't receive 'successfulDeployment' or 'failedDeployment' events. Log messages generated by the command-line sandbox won't be captured by DevTools' logger.
    - If users try to perform operations in DevTools while also using the command line, they may conflict in unpredictable ways.
- The sandbox itself doesn't maintain persistent state information between sessions. When it starts up, it doesn't have inherent knowledge of whether its infrastructure exists in AWS. This is why we have a secondary check of the stack status. The check for CloudFormation stack should tell us definitively whether the sandbox exists or not (assuming it is

not in a deploying or deleting state). We assume if the stack exists, it is in a “stopped” state. This assumption is generally safe — but could cause issues especially if there is a sandbox running in the command line when devtools starts. If the sandbox is in a deploying or deleting state this could be particularly problematic.
    - This safety issue must be dealt with before production.
- There can be edge cases where CLI and UI state might temporarily disagree during rapid state transitions

LOCAL STORAGE IMPLEMENTATION

The LocalStorageManager implements a robust persistence mechanism for storing and managing DevTools data across sessions
Storage Location:

- Uses OS-specific temporary directory: `path.join(tmpdir(), '.amplify', 'amplify-devtools-{identifier}')`
- Namespaces storage by sandbox identifier to isolate different sandbox instances' data
- Creates separate subdirectories for different types of data (logs, CloudWatch logs)

Atomic File Operations:

- Uses `write-file-atomic` package for all file operations
- This ensures data integrity by:
    - Writing to a temporary file first
    - Using fsync to ensure data is flushed to disk
    - Atomically renaming the temporary file to the target file
- This implementation satisfies CodeQL security requirements by preventing "time of check to time of use" vulnerabilities
- Sets 0o600 file permissions (owner read/write only) for security
- When multiple processes might write to the same file, regular writes might interleave, creating corrupt data. Atomic operations ensure one complete write succeeds without interference

LOG STREAMING ARCHITECTURE

Polling vs. Subscription Approach:

- Uses polling for CloudWatch logs instead of subscriptions
- DevTools runs on a private IP address, so AWS EventBridge or CloudWatch Logs subscriptions would require a publicly accessible endpoint or tunnel service
- Polling provides a simpler implementation with no external dependencies

Adaptive Polling Algorithm:

- Implements a dynamic polling frequency in `setupAdaptiveLogPolling()` method
- Initial polling interval: 2000ms (2 seconds)
- Increases interval up to 10000ms (10 seconds) after 3 consecutive empty polls
- Decreases back to 2000ms when new logs arrive
- This balances responsiveness with resource efficiency

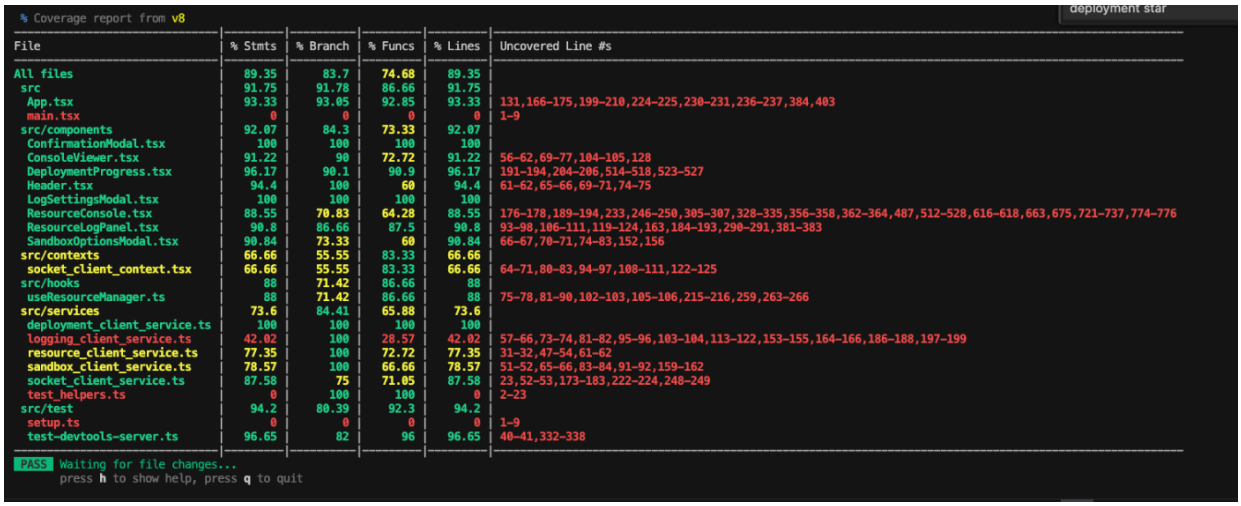
Testing

The project currently has unit, component, and integration tests on both the frontend and backend.

Most importantly, end-to-end tests are currently missing. These should include:

- Full application launch and lifecycle testing
- Cross-browser testing
- Multi-tab testing
- Performance testing under load

Frontend tests should be run from inside the react-app directory using vitest [—coverage].
Current Frontend Test Coverage (not included in GitHub workflow runner)



```
% Coverage report from v8
-------------------------------|---------|----------|---------|---------|-------------------
File                           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------|---------|----------|---------|---------|-------------------
All files                      |   89.35 |     83.7 |   74.68 |   89.35 |
 src                           |   91.75 |    91.78 |   86.66 |   91.75 |
  App.tsx                      |   93.33 |    93.05 |   92.85 |   93.33 | 131,166-175,199-210,224-225,230-231,236-237,384,403
  main.tsx                     |       0 |        0 |       0 |       0 | 1-9
 src/components                |   92.07 |     84.3 |   73.33 |   92.07 |
  ConfirmationModal.tsx        |     100 |      100 |     100 |     100 |
  ConsoleViewer.tsx            |   91.22 |       90 |   72.72 |   91.22 | 56-62,69-77,104-105,128
  DeploymentProgress.tsx       |   96.17 |     90.1 |    90.9 |   96.17 | 191-194,204-206,514-518,523-527
  Header.tsx                   |    94.4 |      100 |      60 |    94.4 | 61-62,65-66,69-71,74-75
  LogSettingsModal.tsx         |     100 |      100 |     100 |     100 |
  ResourceConsole.tsx          |   88.55 |    70.83 |   64.28 |   88.55 | 176-178,189-194,233,246-250,305-307,328-335,356-358,362-364,487,512-528,616-618,663,675,721-737,774-776
  ResourceLogPanel.tsx         |    90.8 |    86.66 |    87.5 |    90.8 | 93-98,106-111,119-124,163,184-193,290-291,381-383
  SandboxOptionsModal.tsx      |   90.84 |    73.33 |      60 |   90.84 | 66-67,70-71,74-83,152,156
 src/contexts                  |   66.66 |    55.55 |   83.33 |   66.66 |
  socket_client_context.tsx    |   66.66 |    55.55 |   83.33 |   66.66 | 64-71,80-83,94-97,108-111,122-125
 src/hooks                     |      88 |    71.42 |   86.66 |      88 |
  useResourceManager.ts        |      88 |    71.42 |   86.66 |      88 | 75-78,81-90,102-103,105-106,215-216,259,263-266
 src/services                  |    73.6 |    84.41 |   65.88 |    73.6 |
  deployment_client_service.ts |     100 |      100 |     100 |     100 |
  logging_client_service.ts    |   42.02 |      100 |   28.57 |   42.02 | 57-66,73-74,81-82,95-96,103-104,113-122,153-155,164-166,186-188,197-199
  resource_client_service.ts   |   77.35 |      100 |   72.72 |   77.35 | 31-32,47-54,61-62
  sandbox_client_service.ts    |   78.57 |      100 |   66.66 |   78.57 | 51-52,65-66,83-84,91-92,159-162
  socket_client_service.ts     |   87.58 |       75 |   71.05 |   87.58 | 23,52-53,173-183,222-224,248-249
  test_helpers.ts              |       0 |      100 |     100 |       0 | 2-23
 src/test                      |    94.2 |    80.39 |    92.3 |    94.2 |
  setup.ts                     |       0 |        0 |       0 |       0 | 1-9
  test-devtools-server.ts      |   96.65 |       82 |      96 |   96.65 | 40-41,332-338
-------------------------------|---------|----------|---------|---------|-------------------
PASS  Waiting for file changes...
      press h to show help, press q to quit
```

Test Utilities

- `packages/cli/src/commands/sandbox/sandbox-devtools/react-app/src/test/test-devtools-server.ts` : Simplified server for frontend integration testing
- `packages/cli/src/commands/sandbox/sandbox-devtools/react-app/src/services/test_helpers.ts` mock socket for service tests
- `packages/cli/src/commands/sandbox/sandbox-devtools/react-app/src/test/vitest.setup.ts` includes mocks for all cloudscape components.

Known Limitations/Critical Improvements Needed

- **UI Constraints**
  - Many widths are hardcoded, causing display issues on different screen sizes
  - No responsive design for mobile or tablet screens
  - Fixed layout doesn't allow customization of views or panels
- **Frontend Error Recovery Limitations**:
  - Missing React error boundaries which could lead to complete UI crashes
  - Insufficient guidance for users on how to resolve certain errors
- **Log Group Support**:
  - Limited support for different resource types' log groups
  - Some AWS resources' logs cannot be viewed
  - Requires manual extension for new resource types

- **Deployment Integration with Existing Utilities**:
  - Should implement Integration with CLI's CfnDeploymentProgressLogger for consistent experience
  - CLI uses a RewritableBlock to update terminal in place, whereas DevTools uses a different approach
  - Unifying these systems would improve consistency
- **Multi-Instance Support**:
  - Make port configurable or implement auto-selection
  - Add detection/management of multiple DevTools instances (we could do this with tab IDs)
  - Support for multiple projects/sandboxes in one UI
- **Sandbox State Issue [from Critical Considerations:]**

Future Directions

FEATURE ENHANCEMENTS

- **Enhanced Visualization**:
  - Resource dependency graph visualization
  - Log pattern recognition and alerting
  - Performance metrics dashboard
- **Developer Tools**:
  - Template generation for common resources
  - Schema validation for resource configurations
  - Integrated documentation viewer
  - AI helper for console logs which can help debug code
- **Data Management**:
  - Data seeding functionality (see design doc)
  - Resource export/import capabilities
- **Performance Improvements**:
  - Implement pagination for large resource sets
  - More aggressive log filtering on the server-side
  - More robust error handling for AWS API failures
  - Enhanced reconnection strategies