Using a serverless framework for implementing a cognitive tutor: experiences and issues

By: Nirmal K Mukhi, Srijith Prabhu, Bruce Slawson
Roadmap

1. Background of our application
2. Why we chose a serverless architecture
3. Design decisions
4. Problems we faced and how we solved them
5. What we have yet to try
Background of application

Figure 1: Watson Tutor Architecture
Background cont.

• Tutoring application that converses with the student.
• Composed of 3 major actions:
  • startSession – sets up the initial configuration for the conversation.
  • converse – does the majority of work with interacting with the student.
  • endSession – returns analytics for the session.
Why choose serverless?

• Good for small and large workloads
• The code for the component was lightweight.
• The flow of the program was mostly calling APIs in sequence and using conditionals to figure out which APIs to call.
• The component didn’t store any information in memory.
• No overhead to maintain REST API.
Example flows

**Figure 2: Start Session sequence**

**Figure 3: Converse sequence**

**Figure 4: End Session sequence**
Approaches

• Disclaimer: some design choices were based on what was available at the time and the speed at which we wanted to create the actions.

• Small granular actions used in a sequence of conditionals for the major action.

• Small to medium sized actions used mostly in sequence.
Small granular actions

• Pros:
  • Reusable
  • Readable
  • Testable

• Cons:
  • Sequence might become more complex and might outweigh the benefits from this approach.
  • Mostly unable to call services in parallel.
Small to medium sized actions

• Pros:
  • Sequence is simpler.
  • Create reusable actions for the components in the sequence that are the same for all major actions.
  • Still able to write unit tests for the actions.
  • Could add some parallel calls to services.

• Cons:
  • Some actions are less readable and reusable.
Problems we faced

• CORS
  • Our app needed to allow for credentials to be passed between client and “server”.
  • API Gateway didn’t support our use case, since it used wildcards in the CORS header.
  • SOLVED: created a utility to produce the proper headers.

• Spin up time
  • Spin up of individual actions added up in the sequence with small workloads.
  • Load testing covered this up for the most part.
  • Combined the sequences into large actions.
Things we have yet to try

• Improve efficiency
  • Use small one file actions and orchestrate using Composer.
Questions?