TCSS 562: Software Engineering for Cloud Computing – Fall 2018
School of Engineering and Technology
University of Washington – Tacoma
http://faculty.washington.edu/wlloyd/courses/tcss562
Instructor: Wes Lloyd

Term Project – Serverless Cloud Native Application
Version 0.1

Project Proposal Due Date: Friday October 19th, 2018 @ 11:59 pm
Project Presentation Date: Wednesday December 12, 11:00am – 1:05pm
Project Final Due Date: Friday December 14 @ 11:59 pm

Teams
TCSS 562 Term Projects will typically be conducted in 3-person teams. The team should submit a single proposal.

Objective
To goal for the TCSS562 term project is to implement a serverless cloud native application using the AWS Lambda serverless computing platform. A predefined project has been provided as an example. Groups are free to adopt the predefined project, or are welcome to propose their own serverless Lambda project that meets the _project criteria_ below. In addition, groups may propose a Term Project that is not related to building a serverless application as long as it involves prototyping design alternatives of cloud systems and includes a cost and performance evaluation. Project proposals are ultimately approved by the instructor. Groups not planning to implement the predefined project should plan to meet and discuss project ideas with the instructor to ensure approval.

Project Criteria:

Projects should implement a cloud application using AWS Lambda. The application should support a case study on one or more of the following: Service Composition/Architecture, "Switchboard" Architecture, Application Flow Control, or Data Provisioning.   The goal of the case study is to contrast performance and cost implications of serverless application implementation alternatives using specific input data as examples. Groups may implement Lambda applications in Java, Node.JS, Python, or any language desired that is supported by the platform.

Types of applications may include:
* Data transformation (as in the predefined project)
* Statistics / Data Aggregation / Graph Generation
* Image Processing / Transformation / Filtering
* Machine Learning: For example, build an application using:
    https://cs.stanford.edu/people/karpathy/convnetjs/

Service Composition: For projects that investigate service composition, the application should have at least three separate services that perform a series of operations on input data. Individual Lambda functions serve to split operations into

separate stages. At least one stage (service) of the computation must take a significant amount of time to compute for at least one example input (~60-seconds). Ideally, the runtime of all services combined would exceed five minutes, though this may be difficult to achieve. It should be possible to compose the application in alternate ways:

Composition #1: Service-A   Service-B   Service-C
Composition #2: Service-A+Service-B-combo   Service-C
Composition #3: Service-A   Service-B+Service-C-combo
Composition #4: Service-A+Service-B+Service-C-combo

Service-A probably can't be composed directly with Service-C because of the expected sequence of operations...Service-C would typically only operate on the output of Service-B...

"Switchboard" Architecture: In addition to combining service code, the notion of a "switchboard" architecture as in Composition #4 can be explored. For a "Switchboard" architecture, all service code is combined into a single deployment package. However, individual calls are made to perform function A, function B, and function C. The switchboard architecture minimizes the number of service deployment packages by bundling all source code together into a single Lambda function. "Switchboard" code at the front of the service then map the inputs to perform the requested processing using internal classes/methods. Minimizing the number of deployment packages is likely to alter the overall cost and performance because of the serverless infrastructure freeze/thaw cycle.

Serverless Infrsatructure Freeze/Thaw:
We will talk about this later, but see this paper, section I. B. for a discussion:
http://faculty.washington.edu/wlloyd/papers/KeepAlive_submitted.pdf

Application Flow Control: A case study on application flow control will compare alternate methods to implement a sequence of service calls and their subsequent data exchange for composition #1 above.

With a laptop-client: The laptop calls all services synchronously and is responsible for moving data to and from each of them: A, then B, then C

Within Lambda: A client makes an asynchronous call to Lambda Service A. Service A then either calls Service B (1) directly, via the (2) Simple Notification Service (SNS) or using the (3) Simple Queueing Service (SQS) to trigger then next call. At the end of the calling sequence final results are retrieved by the original client from the Simple Storage Service (S3) or an alternate location such as the Simple Queuing Service (SQS).

With AWS Step Functions: AWS provides Step functions to define a workflow of serverless functions. A state machine is defined to capture the flow of execution across a set of functions.

Data Provisioning: Data can be provided to a Lambda Function as a standard payload (limited to 6MB), or via an alternate cloud service such as the Simple Storage Service (S3), or alternate cloud services such as Dynamo DB, Amazon

Aurora, Amazon RDS, etc. The goal of a data provisioning case study is to examine implications for data transfer (up and down) when large data sizes are involved for serverless applications. Projects in this category are likely to explore data transfer sizes exceeding 6MB.

<div align="center">

Predefined Project:
AWS Lambda ETL (Extract, Transform, Load) Data Pipeline

</div>

The predefined project is to implement a multi-stage ETL pipeline as a set of independent AWS Lambda services. Those performing the predefined project will then specify which case study they will perform such as: Service Composition/Architecture, "Switchboard" Architecture, Application Flow Control, or Data Provisioning.

<div align="center">

Sales Database

</div>

Sales Data will be provided in CSV format. As sample input we have versions up to 1.5 million rows and 179 MB uncompressed. Data columns include:

Region	text
Country	text
Item Type	text
Sales Channel	text
Order Priority	text
Order Date	date
Order ID	integer
Ship Date	data
Units Sold	integer
Unit Price	float
Unit Cost	float
Total Revenue	float
Total Cost	float
Total Profit	float

Service 1 Data Transformations:

Service 1 includes either the CSV data directly, or a pointer to a CSV file in S3.

Service 1 retrieves CSV data from S3, or directly if provided in the data payload (e.g. REST multipart), and performs transformations on the data.

Service 1 transform and load:
1. Add column **[Order Processing Time]** column that stores an integer value representing the number of days between the [Order Date] and [Ship Date]
2. Transform [Order Priority] column:
   L to "Low"
   M to "Medium"
   H to "High"
   C to "Critical"

3. Add a **[Gross Margin]** column.   The Gross Margin Column is a percentage calculated using the formula: [Total Profit] / [Total Revenue]. It is stored as a floating point value (e.g 0.25 for 25% profit).
4. Remove duplicate data identified by [Order ID]. Any record having an a duplicate [Order ID] that has already been processed will be ignored.

Non-Switchboard Architecture: Transformed data should be written out in CSV format and stored in Amazon S3 for retrieval by Service 2.

"Switchboard" Architecture: Transformed data should be: (1) persisted locally as a CSV file under /tmp, (2) stored in memory, and (3) persisted to Amazon S3. With the "Switchboard" Architecture all services share the same infrastructure. When Service 2 is called, it may find the cached data in memory or under /tmp leftover from Service 1. If the data is unavailable, it is requested from Amazon S3. See article regarding data caching on AWS Lambda: https://medium.com/@tjholowaychuk/aws-lambda-lifecycle-and-in-memory-caching-c9cd0844e072

Scaling Scenario: If there is just 1 call to Service 1 to transform the data, but 10 calls to Service 2 to load the data, with the "Switchboard" Architecture 1 call would find the data locally, and 9 calls will need to request the data from Amazon S3.

Service 2 Data Load:

Service 2 requests include a pointer to the transformed CSV data in S3.

Service 2 loads the data from the CSV file into a single table relational database. The table is keyed by the [Order ID] field which must be unique. Duplicate rows should have been already filtered out by Service 1.

Database:

There are several options for a "data" tier for our serverless application.

Amazon Aurora is Amazon's serverless database service. Both a MySQL and PostgreSQL versions are supported. Our ETL pipeline will perform an initial data transformation (S1), create a relational representation (S2), and then allow multiple read-only queries to be performed (S3).   Since queries in S3 are read-only, using an external data service is not required.

The instructor plans to investigate the use of a local database (Derby DB or SQLite) and provide example code if this works out. The advantage is elimination of a dependency for an external data service for read-only queries. This will keep everyone's costs down. The instructor will also/alternately provide Aurora example code of Lambda.

Java's Derby DB:
https://db.apache.org/derby/integrate/plugin_help/derby_app.html

Groups can propose and adopt alternate backend database approaches and technologies for data storage and query processing here as part of their proposed case study. Design of a serverless application's data tier is likely to have a significant impact on overall performance and hosting costs.

For using a local file-based database with the "Switchboard" Architecture, once Service 2 loads data into a database, such as Derby-DB, the file can be (1) persisted locally under /tmp in the serverless container for later use by Service 3. For non-switchboard architectures, Service 2, exports the Derby DB file to Amazon S3 for retrieval and replication by Service 3.

Service 3 Filtering and Aggregation:

Service 3 requests will be in JSON format.

Service 3 consumes the Derby DB (or Amazon Aurora) and performs meaningful queries to produce output in JSON array format. Each row will be represented as a single JSON object in an array.

Filtering and aggregation can be supported using SQL queries.

Each call to Service 3 will specify 1 or more columns to aggregate data on (GROUP BY), and 0 to many filters which involve including a WHERE clause to an SQL query to specfy column matching requests. Aggregation involves adding a GROUP BY clause to an SQL query and using a function such as SUM(), AVG(), MIN(), MAX(), and COUNT().

If using a local DB, Service 3 begins by checking if there is a local Derby DB file saved. If no file exists the master copy produced by Service 2 can be downloaded from Amazon S3 and cached to support Service 3 requests.

Service 3 will accept requests to filter the full data set by column:
- [Region]="Australia and Oceania"   (*example*)
- [Item Type]="Office Supplies" (*example*)
- [Sales Channel]="Offline" (*example*)
- [Order Priority]="Medium" (*example)*
- [Country]="Fiji" (*example*)

Service 3 will support the following data aggregations by column.
- Average [Order Processing Time] in days
- Average [Gross Margin] in percent
- Average [Units Sold]
- Max [Units Sold]
- Min [Units Sold]
- Total [Units Sold]
- Total [Total Revenue]
- Total [Total Profit]
- Number of Orders

Service 3 outputs each row of output from a relational database query as a separate JSON object in a JSON array. The JSON objects include the data aggregation(s) based on specified filters.

For other project ideas, consult the TCSS 562 slides or instructor.

1 Project Proposal Requirements

[5% of project grade]

The following are key requirements of the project proposal:

Each team will submit a 1 to 2 page short project proposal description.

The proposal must identify:

1. The member names of the project group.
2. The name of the group project coordinator. The project coordinator will be responsible for scheduling and arranging group meetings and work sessions, creating agendas for project check-ins for TCSS 562, and ensuring that tasks are assigned to group members.
3. A description of the proposed project. If conducting the predefined project, this can simply be: "Our team will complete the predefined project.". If an alternate serverless development project is proposed, a project description should be included which describes how the project will meet the project criteria and serve as a good project for one of the proposed case studies: Service Composition/Architecture, "Switchboard" Architecture, Application Flow Control, or Data Provisioning.   If the project is not a serverless project, the project description should describe the cloud systems evaluation that will be performed involve a study of performance and cost.
4. The proposal should then list which case study is planned and a brief description of how the work will be done. Groups may list more than one case study if planning to evaluate multiple design trade-offs. Case studies include but are not limited to: Service Composition/Architecture, "Switchboard" Architecture, Application Flow Control, or Data Provisioning.

Serverless project implementations will be evaluated by the project group(s) using the following evaluation criteria:
- average service turnaround (execution) time for each individual services
- average workflow turnaround time (seconds) for the complete sequence of services: a→b→c
- hosting cost of processing a batch of requests for individual services
- hosting cost of processing a batch of requests for the complete sequence of services: a→b→c
- scalability: performance with an increasing number of concurrent clients from 1 to 100 for example
- cold service performance: performance of service(s) on initial call after 45-minutes of inactivity
- warm service performance: performance of service(s) that have been actively used within the last 5-minutes

If available, include at the end of your proposal any references to websites of interest, or research papers that may help or relate to your proposed project.

Research paper searches can be supported using https://scholar.google.com.

2 Future Deliverables

The final project will involve a short group project presentation (5-10 minutes) during the final exam session on Wednesday December 12th. Requirements of the final project presentation will be provided later on.

The final project will also involve a written report in the IEEE conference format. In the project report, groups will describe their project and the alternate design explored for the case study. The report will describe benchmark testing results for evaluation criteria listed above, and provide a cost / performance comparison for performing batches of service calls (e.g. 1,000, 1,000,000 etc.) Project reports will also include a background and related work section to describe cloud technology used and any relevant comparison studies. Additional details and requirements for the final project report will be provided later on.

3 Project Check-ins (10% of the TCSS 562 course grade)

There will be two or three "written" project check-ins throughout the quarter roughly at two/three-week intervals. The project-checkins are grouped in the same category as activities and quizzes for TCSS 562. Groups are encouraged to meet with the instructor before/after class, during office hours, or by scheduling an appointment to seek clarification and for assistance.

4 Submission Deadline

Project proposals should be submitted in PDF format on Canvas no later than 11:59pm on Friday October 19th. Projects proposals will be approved or revisions requested by class on Wednesday October 24th.

Change History

Version	Date	Change
0.1	10/07/2018	Original Version