Objectives:
This project is designed put together the concepts that we’ve been talking about and practicing throughout the semester. In this project a group of you will work together to build and test a large scale system deployed on Amazon’s Web Services.

Specifically this assignment is designed to get you to think about and design for:

- Availability in systems deployed in the cloud
- Think about how to design a system to support a variety of workloads
- Reason about consistency in a distributed environment
- Recognize how state management is related to systemic properties of interest
- Prepare the system to support scalability

Background:
You’ve had the opportunity to design and develop systems that are dealing with distribution, consistency, and state management in a transparent and rudimentary way. As you have had to write mechanisms to deal with this you’ve hopefully gained insight into the impact of the options at your disposal. You’ll now put this knowledge to use in a more realistic way in this system

Your business has expanded to the point where you no longer feel like you can reasonably manage the administration of your system’s infrastructure or worry about building or acquiring the systems needed to meet this growing demand. As a result you’ve decided to deploy your application in AWS.
**Project:**
You’ll design and develop a system that will be deployed on AWS. The design is up to you with some constraints.

You’ll be designing, building, and testing a system that will again sell bikes. You will be testing the system under various load conditions and configuration and optimizing the system based on the results. You’ll be required to provide both a reference to the system as well as the results of your tests.

The functional requirements for the system are:

- Create an account that includes:
  - A first name
  - Last name
  - Address
  - A username and password
- Browse inventory
  - Order inventory by
    - Price
    - Bike name
- Purchase bikes (must be logged in). **The customer can buy more than one bike (potentially different models).** For each order you should store (the customer information can be gotten from the account):
  - Customer first and last name
  - Shipping address
    - Address
    - City
    - State
    - Zipcode
  - Items purchased
    - Item number(s)
    - Item name(s)
    - Price(s)
    - Total price
    - Quantity
- Search order history (must be logged in)
- Send notification that the bike has shipped
  - The warehouse should indicate when the bike has shipped (this can be some random timeout after the order is placed but does need to be initiated by the warehouse system)
  - The system should send a notice to the user (we can assume that this would be an email but in this case it can just be a pop up notification)
- Allow the user to query the sales data for a specified time period. The system will return a ranked list of the quantity sold for each model in the specified time period.
Data
Your system will manage several kinds of data. You can manage them in any way you want but have to have the following kind of data:

- Customer’s account information
- Orders
- Inventory

You will need to generate data for your system. You can write a simple program that generates data incrementally (in other words has a counter or something that allows for names and so forth to vary). You should generate the following volume of data:

- **Customer Accounts** (100,000) that include at least:
  - First and Last Name
  - Address
  - City
  - State
  - Zip
  - Username (should be a counter)
  - Password (make it the same as the user id)

- **Orders** (10,000,000) that include:
  - Reference to a customer
  - Reference to a model number
  - Quantity
  - Price
  - Date (can be random day & month. The year should be random for 2009 - 2014)

- **Inventory** (100,000 models)
  - Model number
  - Description (have the description be the same with the model number in the beginning)
  - Price (this can be a random number in some range)
  - Quantity (this can be a random number between 1 & 10)

You should test the latency and throughput of the system in 3 configurations:

- With the retail system deployed on one node and the warehouse system deployed on a different node
- With two instances of the retail system and two instances of the warehouse system
- With 4 instances of the retail system and four instances of the warehouse system

For each configuration you should graph the response time under load. You should identify the point at which the system begins to degrade and the number of concurrent users when the system reaches an average response time of 2 seconds. Use this testing to improve the design. Record the data from all of
the tests that you run and record the changes that you’ve made to the system as well as the result (even if the change made the response time worse).

**Technology:**

You are free to use any services that you want on AWS except autoscaling (http://aws.amazon.com/). One concern is that costs run out of control. You should not exceed $50 per team (make use of the free usage tier). Autoscaling will cause your system to add nodes when some defined conditions are reached. You **DON’T** want to do this. It will make testing performance and monitoring costs more difficult. Be sure to remove instances when you are done testing them. Charges are incurred for as long as the instances exist even if they’re not in use. The other thing to be careful of is continually creating and removing instances. You get charged for an hour every time you create and remove an instance. You can find more information on the free tier here http://aws.amazon.com/free/.

You will need to sign up for an AWS account and send me your account name so that I can consolidate the billing. You can sign up for multiple accounts within the team to allow for parallel testing and to make use of the free usage tier but pick one account for the official deployment.

For simulating load on the system you can use JMeter (http://jmeter.apache.org/). The use cases that you should test are:

- Log in
- Get catalog of current inventory
- Attempt to place an order (should be logged in so you don’t need customer data to be entered), you can randomly select the bike id and quantity should be between 1 and 10

**Deliverables:**

You should deliver:

- your source code and deployment and execution instructions. The code and instructions should be zipped together and submitted via blackboard by the due date. Late assignments will be penalized one letter grade per day late.
- A link to the system deployed on your instance of AWS (with the one node configuration)
- A report showing
  - The initial design
  - The results of the tests with the initial design
  - Changes to the design
  - The results of the tests with each change
- Presentation describing:
  - Your design (and rationale)
  - The results of your testing
The changes and subsequent results
Analysis of the design and results (What were your observations? What do you think were the major contributors to the results? What could you change to improve the response time?)

You’ll submit one assignment per team. Each team will have 20 minutes to present (15 for the presentation and 5 for Q&A).

**Evaluation:**
You have a wide range of latitude with respect to how you implement this system. You will be evaluated along several dimensions:

- Quality of your implementation: It’s expected that you adhere to acceptable coding standards. The more you use generally acceptable practices as opposed to just “hacking” the solution together the better you will do along this dimension. It’s also expected that the system works. A fully working system is better than a partially working system, a partially working system is better than a system that builds but doesn’t execute, and so forth.
- Understanding: You’ll be evaluated based on your level of understanding with respect to what you’ve implemented and the impacts of these solutions. These will be discussed orally after you submit your assignment.
- **Bonus:** Bonus points will be given to the team that has the best response time with the 4 node deployment and 1000 concurrent users.