Aim

To demonstrate an understanding of how to construct a scalable multi-Cloud application. You will propose, and then implement, test, evaluate, and demonstrate, a multi-Cloud application that computes Value at Risk for some well-known companies listed on stock markets. Your application will need to meet a set of provided **Requirements,** with analyses to be run over provided data as described in the **Approach,** for **Submissions** as outlined.

Requirements

i. You must use both Google App Engine and EC2. You should additionally use one of the other *scalable services*: AWS Lambda, or Elastic MapReduce (EMR), or – should you wish to explore separately – EC2 Container Service (ECS).
In subsequent references to *scalable services*, we mean EC2 plus your choice of (Lambda or EMR or ECS).

ii. Your system must offer a persistent front-end through which you can present information about financial risk for each *name* within the financial data to end users (where a name is, for example, Amazon, Google etc.).

iii. The system must present a Google Chart that shows the *time series* (of Adj. Close values) and *returns series* for the name.
VaR values should be presented with the chart. There are several ways in which such data could be presented, and the rationale for your final selection should be justified carefully. At minimum, they must be presented on the same page.

iv. The *scalable services* must run the Monte Carlo analysis, with results made available for presentation by the persistent front end.

v. The other kinds of analysis can be run anywhere, though the choice of where it runs, and why, will need to be explained.

vi. The *scalable services* must be used dynamically – i.e. any resources used in calculating the Monte Carlo values should be switched on and off (automatically, via code) for the purpose and should not be left on continuously.

vii. It must be possible to specify a value of **Investment** through the persistent front-end to be used to generate the values of actual losses.

viii. It must be possible to specify the following values for use in the analysis through the persistent front end:
   a. a value of **T**, as the number of data points to use for Historical and Covariance analysis.
   b. a value of **M**, for the total number of Monte Carlo samples to be used.
   c. a value of **R**, as the number of resources (in the *scalable services*), to be used for the analysis, so that each runs approximately **M**/**R** simulations. You should also allow for **R** to be combined with a selection between the scalable services.

ix. Data ('time series') will be provided, and you will need to document a simple mechanism for loading the data into your system. For the demonstration, you *may* be given a new set of data in the same format which you will need to load into your system.

Your system may incorporate additional Cloud components, for example for storage. However, the mantra of Keep It Stupid-Simple should be followed and additional component should not be added unnecessarily.

Approach

i. The analysis involves crafting some small fragments of code which will construct three kinds of Value at Risk (VaR: see http://www.investopedia.com/articles/04/092904.asp for more information) using the value for **Investment**:

Step 1. Identify the adjusted close (Adj Close) prices quoted.

Create the ***returns series*** for each *name* using:

$$\frac{\text{closing\_price}(day_t) \; - \; \text{closing\_price}(day_{t-1})}{\text{closing\_price}(day_{t-1})}$$

for all days for which this is possible – i.e. if you have ***(T)*** data points initially, your new series will consist of ***(T**-1)* data points.

Step2. Calculate the VaR value with two confidence values (95%, 99%) using three methods:

i. **Historical:** Sort the ***return series*** from largest *profit* to largest *loss*.
The first Value at Risk (VaR) is the **Investment** multiplied by the loss value at **95%**, e.g. for (**T** =) 101 close prices giving 100 *returns*, the 95th value in this series.

Similarly, find the VaR at **99%** confidence.

ii. **Covariance:** Calculate the mean (average) and standard deviation of the returns series.
The VaR at 95% confidence is found at 1.65 standard deviations (**σ**) from the mean:

- (mean + (1.65 * **σ**)) * **Investment**

The VaR at 99% confidence is similarly found at 2.33 standard deviations.

iii. **Monte Carlo:** Use the mean and standard deviation calculated in (ii), and provide these 2 values to a random number generator. Use this random number generator to derive a new data series of a specifiable length **M** (or **M**/**R** if using multiple *scalable services*)

Using only the most recent price, as **p**: for each random number **q**, that represents a percentage change, generate a new price **n**:

n = (1+q) * p

such that a new close price series (comprising a list of **M, or M**/**R** elements, as relevant) is produced that can be used with **Step 1** above to obtain the values at 95% and 99% confidence**.**

When using **R** resources from *scalable services*, compute the mean (average) VaR over **R** separately calculated values – so, if **R**=10 and **M**=1,000,000, each **R** will produce 100,000 values, and you'll take the mean of the 10 values at 95%, and the mean of 10 values at 99%.
HINT: In Python, **random.gauss** can be useful; in Java, **java.util.Random's** nextGaussian().

Step3. Collate and present the final 6 VaR results. In testing your system, you may find it useful to compare the influence on VaR of increasing and decreasing **M**.