**Amazon**

**Software Engineer**

In this role, you will:

* Work with other team members to investigate design approaches, prototype new technology and evaluate technical feasibility
* Work extensively AWS technologies such as SQS, SNS, S3, Lambda, DynamoDB, RDS, Redshift, CloudFormation, CloudWatch, Data Pipelines, and Kinesis
* Working in an Agile/Scrum environment to deliver high quality software against aggressive schedules
* Establish architectural principles, selecting design patterns and mentoring team members on their appropriate application

Qualifications

* Bachelor's degree or foreign equivalent in Computer Science, Engineering, Mathematics, or a related field
* Experience developing software services and an knowledge of design for scalability, performance and reliability
* Experience with a variety of modern programming languages (Java, JavaScript, C/C++) and open-source technologies (Linux, Spring)
* 4+ years of experience designing and developing large-scale, distributed software applications, tools, systems and services using Java, C#, or C++, and Object Oriented Design.

· Bachelor's degree or foreign equivalent in Computer Science, Engineering, Mathematics, or a related field

· 4+ years experience developing software services and an understanding of design for scalability, performance and reliability

· 4+ years experience with a variety of modern programming languages (Java, JavaScript, C/C++) and open-source technologies (Linux, Spring)

· 4+ years of experience in the job offered or a related occupation must involve: designing and developing large-scale, distributed software applications, tools, systems and services using Java, C#, or C++.

· Knowledge of object oriented analysis and design patterns (OOA/OOD)

**Nike**

**Senior Software Engineer**

* MS/BS degree in a computer science or related discipline
* 4+ years’ experience in front-end engineering
* Excellent demonstrated experience using current web technologies (NodeJS, React, Angular, Electron, VueJS, etc.)
* Working knowledge of basic computer science data structures and algorithms
* Strong skills around integrating with RESTful API’s
* Good exposure to AWS technology stack
* Experience in trouble shooting and root cause analysis for any performance issues
* Experience with source control tools (GIT) and ability to review code, provide feedback on pull requests

**Lead Data Engineer**

* Experience developing cloud scalable, real time and high-performance data lake solutions
* Deep experience with Python, Spark, Shell & SQL
* Experience with AWS components and services, particularly EC2, EMR, ECS/EKS, S3 and Lambda/Serverless
* Experience writing automation to deploy R (RStudio), Spark ML, and/or Python apps (pandas, numpy, scipy, keras, etc.)
* Experience with Container Orchestration (Kubernetis, Docker,etc)
* Experience with software engineering best practices including unit testing, continuous integration (Jenkins, CircleCI, etc), and source control (GitHub, Bitbucket,etc)
* Experience with workflow scheduling instruments like Airflow
* Thorough understanding of statistical modeling and machine learning techniques
* Experience building APIs and micro service architecture
* Has an excellent problem solving and critical thinking mindset
* Demonstrated effective portfolio of accomplishments of learning new tools and technologies
* Able to influence and communicate effectively, both verbally and written, with team members

**Data Scientist**

Requirements;

* Demonstrated technical, analytical skill set to work with analytics toolkit (R, Python, SQL, Hadoop, Amazon Web Services (AWS), SAS, Tableau, scheduling tools) on large data sets
* Experience building and maintaining production analytics systems and models at enterprise scale

**Google**

Minimum qualifications:

* BS degree in Computer Science, similar technical field of study or equivalent practical experience.
* Software development experience in one or more general purpose programming languages.
* Experience working with two or more from the following: web application development, Unix/Linux environments, mobile application development, distributed and parallel systems, machine learning, information retrieval, natural language processing, networking, developing large software systems, and/or security software development.
* Working proficiency and communication skills in verbal and written English.

Preferred qualifications:

* Master’s, PhD degree, further education or experience in engineering, computer science or other technical related field.
* Experience with one or more general purpose programming languages including but not limited to: Java, C/C++, C#, Objective C, Python, JavaScript, or Go.
* Experience developing accessible technologies.
* Interest and ability to learn other coding languages as needed.

**Interviews for all roles**

Here's our advice to help you be ready for your interview.

* **Predict the future**: You can anticipate 90% of the interview questions you’re going to get. “Why do you want this job?” “What’s a tough problem you’ve solved?” If you can’t think of any, Google “most common interview questions.” Write down the top 20 questions you think you’ll get.
* **Plan**: For every question on your list, write down your answer. That will help them stick in your brain, which is important because you want your answers to be automatic.
* **Have a backup plan**: Actually, for every question, write down THREE answers. Why three? You need to have a different, equally good answer for every question because the first interviewer might not like your story. You want the next interviewer to hear a different story and become your advocate.
* **Explain**: We want to understand how you think, so explain your thought process and decision making throughout the interview. Remember we’re not only evaluating your technical ability, but also how you approach problems and try to solve them. Explicitly state and check assumptions with your interviewer to ensure they are reasonable.
* **Be data-driven**: Every question should be answered with a story that demonstrates you can do what you’re being asked about. “How do you lead?” should be answered with “I’m a collaborative/decisive/whatever leader. Let me tell you about the time I … ”
* **Clarify**: Many of the questions will be deliberately open-ended to provide insight into what categories and information you value within the technological puzzle. We’re looking to see how you engage with the problem and your primary method for solving it. Be sure to talk through your thought process and feel free to ask specific questions if you need clarification.
* **Improve**: Think about ways to improve the solution you present. It’s worthwhile to think out loud about your initial thoughts to a question. In many cases, your first answer may need some refining and further explanation. If necessary, start with the brute force solution and improve on it — just let the interviewer know that's what you're doing and why.
* **Practice**: Everyone gets better with practice. Practice your interview answers—out loud—until you can tell each story clearly and concisely.
* **Coding practice**: You can find sample coding questions on sites like CodeLab, Quora, and Stack Overflow. The book “Cracking the Coding Interview” is also a good resource. In some sites, you'll have the option to code on either a Chromebook or a whiteboard, to offer a more natural coding environment. (Ask your recruiter what's available so you can practice.) Be sure to test your code and ensure it’s easily readable without bugs. Don’t stress about small syntactical errors like which substring to use for a given method (e.g. start, end or start, length) — just pick one and let your interviewer know.
* **Coding**: You should know at least one programming language really well, preferably C++, Java, Python, Go, or C. You will be expected to know APIs, Object Orientated Design and Programming, how to test your code, as well as come up with corner cases and edge cases for code. Note that we focus on conceptual understanding rather than memorization.
* **Algorithms**: Approach the problem with both bottom-up and top-down algorithms. You will be expected to know the complexity of an algorithm and how you can improve/change it. Algorithms that are used to solve Google problems include sorting (plus searching and binary search), divide-and-conquer, dynamic programming/memoization, greediness, recursion or algorithms linked to a specific data structure. Know Big-O notations (e.g. run time) and be ready to discuss complex algorithms like Dijkstra and A\*. We recommend discussing or outlining the algorithm you have in mind before writing code.
* **Sorting**: Be familiar with common sorting functions and on what kind of input data they’re efficient on or not. Think about efficiency means in terms of runtime and space used. For example, in exceptional cases insertion-sort or radix-sort are much better than the generic QuickSort/MergeSort/HeapSort answers.
* **Data Structures**: You should study up on as many data structures as possible. Data structures most frequently used are arrays, linked lists, stacks, queues, hash-sets, hash-maps, hash-tables, dictionary, trees and binary trees, heaps and graphs. You should know the data structure inside out, and what algorithms tend to go along with each data structure.
* **Mathematics**: Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because counting problems, probability problems and other Discrete Math 101 situations surround us. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of elementary probability theory and combinatorics. You should be familiar with n-choose-k problems and their ilk.
* **Graphs**: Consider if a problem can be applied with graph algorithms like distance, search, connectivity, cycle-detection, etc. There are three basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list) — familiarize yourself with each representation and its pros and cons. You should know the basic graph traversal algorithms, breadth-first search and depth-first search. Know their computational complexity, their tradeoffs and how to implement them in real code.
* **Recursion**: Many coding problems involve thinking recursively and potentially coding a recursive solution. Use recursion to find more elegant solutions to problems that can be solved iteratively.