Cloud Computing and Architecture for Data Scientists

Scalable Data Science Beyond The Local Machine

Data science is a term that represents the intersection of many important things. In an article of mine entitled What Is Data Science, and What Does a Data Scientist Do?, I discuss what I call the pillars of data science expertise.

These pillars of expertise include:

- Business domain
- Probability and statistics
- Computer science and software programming
- Written and verbal communication

The third so-called pillar is all about having an understanding of computer science and software programming.

While it's likely not immediately obvious to up-and-coming data scientists, this area also often includes things like devops, cloud computing, data pipelines, data engineering, expertise querying different types of databases, building and deploying production software solutions, and so on.

Also, data scientists need to develop solid programming skills, but they may not be as educated or experienced in computer science, programming concepts, or general production software architecture and infrastructure as well-trained or experience software engineers. When

anyone starts in data science, they'll find themselves installing Python and/or R on their local computer, and then writing and executing code in a local integrated development environment, or IDE such as the Jupyter Notebook Application or RStudio.

In addition, as advanced analytics becomes more prevalent and data science teams grow, there is growing need for collaborative solutions to delivery insights, predictive analytics, recommendation systems, and so on. Reproducible research and notebook tools combined with code source control is part of the solution for collaboration, while leveraging collaborative cloud-based online tools and platforms in another.

Collaborative needs also extend to include those outside of data science teams, particularly since data science is largely deployed to achieve business goals. As such, the stakeholders of a data science project can include executives, department leads, and other data team staff such as architects, data engineers, analysts, and so on.

This article is intended to give data scientists insight into what's beyond the local laptop or desktop machine, particularly in the context of putting data science solutions into production, or in terms of expanding your computing power and capabilities. Depending on your experience with these particular topics, this content should be relevant to data scientists of all skill levels.

Let's get started!

What Is the Cloud Exactly?

Ah yes, the infamous and still often not well understood cloud. Besides sounding very hypothetical and abstract, the cloud is actually very concrete in its intended meaning. Let's define some key concepts before moving onto the concept of the cloud.

Computers connected together that share resources are called a network, with the Internet itself being the largest and most famous example of a computer network. Home networks, such as a Local Area Network (LAN) or WiFi Service Set Identifier (SSID), in which multiple computers are connected, is another example, although they are much smaller. The resources described here can include web pages, media, data storage, app servers, printers, etc.

Computers in a network are usually called nodes, and they communicate with each other using well-defined protocols such as HyperText Transfer Protocol (HTTP), Transmission Control Protocol (TCP)/ Internet Protocol (IP), and so on. These communications can be for status updating, monitoring, request/response, and many other uses.

Further, computers are often not located on-premise, meaning that applications and data are often hosted on computers located in a data center. These places provide all necessary infrastructure, such as power, cooling, security, disaster protection, etc. for maintaining and successfully running a large number of computers that are accessible to companies, or the outside world in general.

Because computers and storage have become relatively cheap over time, many solutions now employ multiple computers working together that are not too costly to scale, as opposed to scaling solutions by purchasing a single super powerful and very expensive computing machine. Part of this ‘working together’ is to ensure that the solution continues running automatically even if one of the computers fails and also that the system is able to automatically scale to handle any imposed load on the system.

Twitter, Facebook, Instagram, Snapchat, Netflix, and YouTube are perfect examples of cloud-based applications that need to scale in both of these ways. It is highly unlikely to see their applications ‘go down’ completely, and they are also able to handle literally millions of daily users engaging with their platforms.

When a certain group of computers are connected to the same network and are working together to accomplish the same task or set of tasks, this is referred to as a cluster. A cluster can be thought of as a single computer, but can offer massive improvements in performance, availability, and scalability as compared to a single machine. These benefits will be discussed later on in this post.

The term distributed computing or distributed systems refers to software and systems that are written to leverage clusters to perform specific tasks, such as Hadoop, Spark, and MapReduce.

Finally, onto the definition of the cloud. In addition to the shared resources described above, other important solution resources can include servers, services, microservices, networks, and so on. A cloud describes the situation where a single party owns, administers, and manages a group of networked computers and shared resources typically to host and provide software-based solutions. Given this definition, while the Internet is definitely considered a network, it is not a cloud since it's not owned by a single party.

For a deeper dive into cloud computing, and discussion of key concepts in creating scalable software and big data architectures, check out my three-part in-depth series on this topic.

Data Science in the Cloud

This post has now discussed cloud computing and other related concepts in enough depth to hopefully illustrate the concepts involved. If your exposure to software architecture and engineering at this point is limited to local development only, you may be wondering why this is all relevant to data scientists. This is what is covered next.

If you're familiar with the data science process, you know that often most of the data science workflow is carried out on a data scientists local

computer. The computer has the languages of choice installed on it, like Python and R, and also the data scientists preferred IDE. The other primary development environment setup is to install relevant packages either via a package manager like Anaconda, or by installing individual packages manually.

Once the development environment is setup and good to go, the typical-ish data science workflow or process begins, with data being the only other thing needed for the most part. The iterative workflow steps typically include:

- Acquiring data
- Parsing, munging, wrangling, transforming, and sanitizing data
- Analyzing and mining the data, such as Exploratory Data Analysis (EDA), summary statistics, ...
- Building, validating, and testing models, such as predictive, recommendations, ...
    - Note: If not building models, then identifying patterns or trends, generating actionable insights, extracting useful information, creating reports, and so on. Here, this tutorial will consider either case "creating a deliverable".
- Tuning and optimizing models or deliverables

Sometimes however, it is not practical or desirable to perform all data science or big data-related tasks on ones local development environment. Here is a list of some of the main reasons why:

- Datasets are too large and will not fit into the development environment's system memory (RAM) for model training or other analytics

- The development environment's processing power (CPU) is unable to perform tasks in a reasonable or sufficient amount of time, or at all for that matter

- The deliverable needs to be deployed to a production environment, and possibly incorporated as a component into a larger application (for example, web application, SaaS platform, ...)

- It is simply preferred to use a faster, and more powerful machine (CPU, RAM, ...) and not impose the necessary load on the local development machine

When these situations arise, there are multiple options available. Instead of using the data scientist's local development machine, typically people offload the computing work to either an on-premise machine, or cloud-based virtual machine (for example, AWS EC2, AWS Elastic Beanstalk). The benefit of using virtual machines and auto-scaling clusters of them, is that they can be spun up and discarded as needed, and also tailored to meet ones computing and data storage requirements.

In the case of deploying deliverables to a production environment to be used as part of a larger application or data pipeline, there are many options and challenges to consider. Further discussion of that is out of scope for this article.

In addition to custom developed cloud-based or production data science solutions and tools, there are many cloud and service-based offerings available from very notable vendors as well, which often work well with notebook tools like Jupyter. These are available largely as big data, machine learning, and artificial intelligence APIs, and include options like the AWS Artificial Intelligence platform, Databricks, Google Cloud Platform Datalab and Machine Learning, and many more.

For a much more detailed, in-depth discussion of data science and

advanced analytics in production versus development, including recommendations for languages, packages, frameworks, and platforms, please check out my three-part series on this topic. My scalable software and big data architectures series is also a great complement on cloud computing.

Software Architecture and Quality Attributes

Software architecture involves designing a software system, usually cloud-based, that represents a product, service, or task-based computing system. You may also hear of the terms system architecture or software architecture, both of which mean more or less the same thing.

Part of designing a software architecture is to choose the appropriate programming languages and technologies (aka stack), components, packages, frameworks, platforms, and so on. This can require a lot of consideration, particularly around the systems intended purpose and any other important tradeoffs. This aspect of software architecture requires skills, knowledge, and experience gained by a person, like a software architect, over time.

Another critical aspect of systems and software architecture and engineering are so-called quality attributes or non-functional requirements. This is especially true for real-world production solutions.

Non-functional requirements typically includes things like:

- Availability
- Performance
- Reliability
- Scalability (up and out)

- Extensibility
- Usability
- Modularity
- Reusability

In this article, you'll get a brief overview of four of the most important ones, namely, availability, performance, reliability, and scalability. Note that the discussion is meant to be high level and not get into metrics-based definitions or requirements for these quality attributes.

Availability is just as it sounds, that the system is available, or in other words is up and running properly. This can mean many things and is also heavily dependent on reliability and scalability. "Up and running properly" means that the system works as intended and whenever it is needed. This can be by an end user trying to use the system, e.g., Facebook or Netflix, or it can be a set of cloud-based services used for processing data for example.

Reliability is a term that represents the ability of the system to run and work properly without failures or errors. The more that a system can run this way, the more fault tolerant the system is said to be. Since it's hard to think of, and test for all possible use and edge cases in advance, 100% reliability can be very difficult to achieve. Also, failures can happens for many reasons, with code bugs, environmental issues, and limited resources (CPU, RAM, disk memory, ...) being some of the primary culprits.

Performance is a term used to describe the speed by which the system carries out tasks, or another way to think about it is the time it takes for the system to perform a specific task. Take YouTube for example. You expect that when you go to watch a video, the video will load and start

playing in a reasonable amount of time. The more performant that Google can make YouTube, the faster the video loads and starts playing, the happier YouTube's users are, and the less likely it becomes that they'll abandon the application. The opposite would be if YouTube ran extremely slow to the point that it's not worth the benefit, and therefore people stopped using it. Luckily the former is the case most of the time, and not the latter.

Lastly, scalability is critical for certain applications, and is a term used to represent a system's ability to maintain a certain level of performance (as defined above, again high level) despite increasing load on the system. Load is a term used to mean the number of concurrent or simultaneous requests to the system.

A great example where scalability is required is when tickets first go on sale for a popular sports team or performing music artist. Depending on the popularity, the number of concurrent requests to purchase tickets at a web application like Ticketmaster can be in the hundreds of thousands when the tickets first go on sale. Depending on a systems ability to scale, this can either greatly decrease its performance, or shut the system down all together. Neither of these scenarios is good.

To address this requirement, techniques are used to either scale the system up or scale it out. Scaling up is when the computing device(s) in the system are replaced with more powerful (CPU, more cores, ...) and reliable machines with larger resources such as system memory or RAM. Machines like this can be expensive.

Scaling out on the other hand is where low cost commodity machines are used instead, and are not particularly powerful individually, but when used as a group are more than capable of handling the load on the system. Since solutions that use this technique require multiple computers, it is important to setup systems to automatically handle scenarios where one or more of the computers (nodes) fails or crashes

(failover), and so on.

Conclusion

Hopefully this article has helped shed some light on the many important aspects of data science beyond local development, and in the context of real-world production solutions. Cloud-based computing and architecture concepts are important to understand when working on production data solutions, or requiring additional computing power and resources.

This article was originally published on DataCamp's blog here.