**Topic Selected: How would you build Netflix?**

**High Level Approach:**

First, you would need to design and set up a scalable, highly available infrastructure to host the video content and serve it to users. This would involve using a combination of on-premises servers and cloud-based storage and computing resources to ensure that the system can handle a large number of users and handle sudden spikes in traffic.

Next, you would need to design and implement the system for storing, transcoding, and delivering video content to users. This would involve choosing the right file formats and codecs for the video files and developing a system for efficiently storing and accessing the files in a way that allows for fast and seamless streaming to users.

You would also need to design and implement the user interface and front-end systems that allow users to browse and search for content, as well as the systems that handle user accounts and payment processing. This would involve choosing the right technologies and frameworks for building a responsive and user-friendly interface, as well as integrating with payment processors and other external systems.

Finally, you would need to design and implement the systems for monitoring and analysing the performance and usage of the platform, in order to identify any bottlenecks or problems and optimize the system for maximum efficiency and reliability. This would involve setting up metrics and logging systems, as well as developing tools and processes for analysing the data and making informed decisions about how to improve the system.

**Design and Architectural Level Approach:**

When building a service like Netflix from a computer architecture point of view, several design patterns may be useful in order to create a scalable, reliable, and efficient system. Some potential design patterns that could be used include:

Microservices: This design pattern involves breaking the system down into smaller, independent components that can be developed, tested, and deployed independently, and can communicate with each other through well-defined interfaces. This can help to make the system more flexible, scalable, and maintainable, and can allow different parts of the system to be optimized and improved without affecting the rest of the system.

CQRS (Command Query Responsibility Segregation): This design pattern involves separating the data access and manipulation logic (the "command" side) from the data retrieval and presentation logic (the "query" side), and using separate data stores for each. This can help to improve the performance and scalability of the system, as well as allowing for more flexible and efficient data access and manipulation.

Event sourcing: This design pattern involves storing all changes to the data in the system as a sequence of events, rather than just storing the current state of the data. This can make it easier to track and audit changes to the data and can allow for more flexible and powerful querying and analysis of the data.

Cache-aside: This design pattern involves storing frequently used data in a cache, in order to improve the performance and scalability of the system. The cache can be updated on demand, or asynchronously, in order to ensure that it stays up to date with the latest data from the persistent store.

These design patterns, among others, could be useful in building a service like Netflix from a computer architecture point of view, in order to create a scalable, reliable, and efficient system.