Exercise 7

Sonntag, 22. Juni 2025 15:22

1) Comparison of computing models

	Mainframe Computing	**Cloud Computing**
Centralization	Centralized, powerful machine used by many users	Distributed over many data centers and locations
Ownership	Fully owned and operated by the organization	Usually operated by third-party providers (e.g., AWS, Azure)
Cost Structure	High upfront hardware and maintenance costs	Pay-as-you-go or subscription-based
Scalability	Limited and expensive to scale	Easily scalable on demand
Accessibility	Typically on-premise, limited remote access	Accessible from anywhere with internet
Technology Stack	Legacy software, COBOL, batch processing	Modern stack, microservices, APIs, real-time apps
Maintenance	Requires in-house specialists	Maintenance done by cloud provider

Why the dramatic change over time?:

The transition was driven by a combination of **technological advancements**, **cost efficiency**, and **changing user needs**.

- **Cost Efficiency:** Cloud computing reduces upfront investment and offers flexible, usage-based pricing, which is more practical for modern businesses.
- **Scalability and Flexibility:** Unlike mainframes, cloud services can scale resources up or down easily to meet changing demands.
- **Global Accessibility:** With the rise of the internet, cloud resources became available from anywhere, while mainframes were mostly limited to local access.
- **Faster Innovation:** Cloud platforms support rapid development and deployment, enabling companies to bring products to market faster.
- **Simplified Maintenance:** Cloud providers handle updates, backups, and security, reducing the need for in-house IT staff.

⇒ This made cloud computing the more attractive and adaptable model over time.

2) Varia - Advantages of this architecture (comparison to exercise 6)

On-demand resource scaling
The cloud can spin up servers dynamically and shed them when tasks are complete, ensuring efficient resource use
In my implementation, resource usage and checks are fixed and always running locally—no elastic capacity.

Zero upfront infrastructure cost
Cloud computing avoids buying or maintaining expensive hardware and real estate; you only pay for what you use
My current tool requires a local machine and manual setup—costs are fixed regardless of load.

Flexible, usage-based billing
With cloud services, costs scale with usage—no need to budget for idle resources .

My solution has no usage-based model; it's limited to local compute without cost optimization.

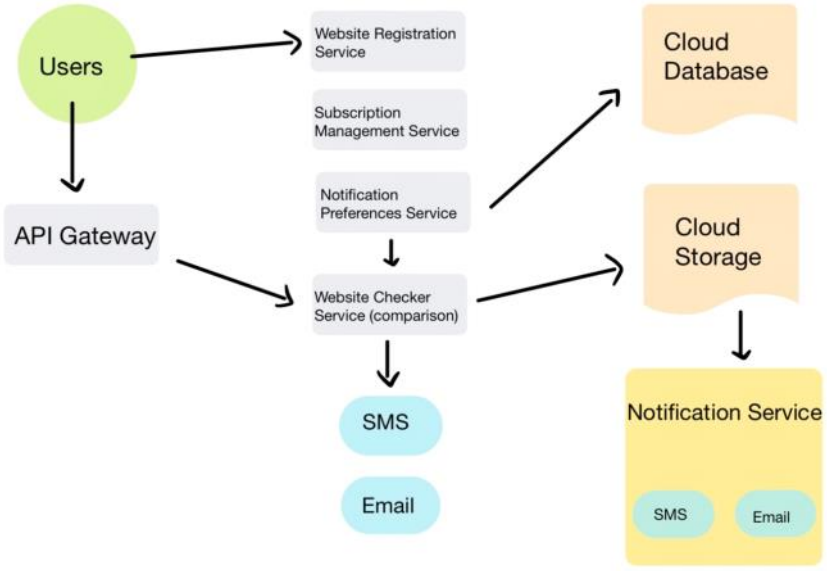
Massive parallel processing
Cloud supports running hundreds or thousands of instances in parallel (e.g., via Hadoop), drastically reducing processing time
My system processes one website at a time and doesn't support parallel or distributed checking.

Managed services and automation
Cloud providers offer APIs for automation (auto-scaling, provisioning, queuing, storage, etc.)
My implementation uses manual polling logic without APIs, auto-scaling, or robust queueing mechanisms.

Summary of Differences

	Cloud Architecture	**Your Implementation**
Resource Scaling	Elastic + on-demand	Fixed, local execution
Infrastructure Cost	Zero upfront, pay-as-you-go	Local hardware, full cost always
Billing	Usage-based	No cost model integrated
Parallelism	High parallel processing	Single-threaded polling
Automation & Services	Auto-scaling, API-driven	Manual scheduling or triggers

3) Reiterated cloud architecture



4) Failed programming task for AI-assisted coding tool

Example: “Write a Java method to check if a given string is a palindrome, ignoring case and non-alphanumeric characters.”

AI-Solution:

```
public boolean isPalindrome(String s) {
   s = s.toLowerCase();
   return s.equals(new StringBuilder(s).reverse().toString());
}
```

Why is the solution incorrect?:

- The method converts the string to lowercase but **does not remove non-alphanumeric characters** (like spaces, punctuation, etc.).
- For input like "A man, a plan, a canal: Panama", this method will return false because of commas and spaces.
- The requirement is to **ignore non-alphanumeric characters**, which the method fails to handle.

Corrected Version:

```
public boolean isPalindrome(String s) {
   StringBuilder cleaned = new StringBuilder();
   for (char c : s.toCharArray()) {
      if (Character.isLetterOrDigit(c)) {
         cleaned.append(Character.toLowerCase(c));
      }
   }
   String cleanedStr = cleaned.toString();
   return cleanedStr.equals(new StringBuilder(cleanedStr).reverse().toString());
}
```