**SWED Übung 8**

**Aufgabe 1**

**Virtual Machines (VMs):**

* Run a complete operating system with its own kernel
* Include the full OS overhead (several GBs in size)
* Provide strong isolation through hardware virtualization
* Slower startup times (minutes)
* Managed by a hypervisor (like VMware, VirtualBox)
* Each VM has its own allocated resources (RAM, CPU, storage)

**Docker Containers:**

* Share the host operating system's kernel
* Much lighter weight (MBs in size)
* Provide process-level isolation
* Fast startup times (seconds)
* Managed by container runtime (Docker Engine)
* Share host resources more efficiently

**Key Difference:** VMs virtualize hardware and run complete operating systems, while containers virtualize the operating system and share the host kernel, making them much more lightweight and efficient.

**Aufgabe 2**

The correct answer is **c) frequently integrate code into a shared repository and run automated tests**.

**Explanation:**  
Continuous Integration (CI) is a development practice where developers frequently merge their code changes into a central repository, typically multiple times per day. Each integration triggers automated builds and tests to detect integration errors quickly.

The other options are incorrect because:

* a) Automatic deployment is part of Continuous Deployment (CD), not CI
* b) Merging "every hour" is too rigid - CI focuses on frequency but not specific timing
* d) Building Docker containers is a possible CI task but not the main purpose

The core goal of CI is early detection of integration problems through frequent code integration and automated testing.

**Aufgabe 3**

Befehle:

cd Schreibtisch/SWED/Exc8/website-monitor

sudo docker build -t website-monitor .

sudo docker run --rm website-monitor https://www.google.de

Falls es nicht funktioniert: sudo docker build –t website-monitor .

**Aufgabe 4**

The correct answer is b) Jenkins.

Explanation:  
Jenkins is a widely used open-source automation tool specifically designed for orchestrating CI/CD pipelines, including building, testing, and deploying projects.

The other options have different primary purposes:

* a) Kubernetes: This is a container orchestration tool used for deploying, scaling, and managing containerized applications.
* c) Ansible: This is a configuration management tool used to automate software provisioning and infrastructure management.
* c) Terraform: This is an "Infrastructure as Code" tool used for provisioning and managing infrastructure resources.

While these tools can be part of a CI/CD process, Jenkins is the tool specifically focused on orchestrating the entire workflow.

**Aufgabe 5**

These are several strategies to speed up a slow testing stage in a CI pipeline:

* Run Tests in Parallel: Instead of running tests one after another, execute multiple tests simultaneously across different browsers and devices. This is one of the most effective ways to significantly reduce the total execution time.
* Optimize the Test Suite: Manage your tests by categorizing them. Run fast unit tests on every code change, while running slower, more comprehensive integration or end-to-end tests less frequently (e.g., nightly or before a major release). You should also remove old or redundant tests that no longer add value.
* Use More Powerful Hardware: Increasing the CPU and memory of the build machines (vertical scaling) can directly speed up how fast tests are executed.
* Implement Caching: Cache dependencies and artifacts from previous builds. This prevents the pipeline from having to download or rebuild the same components for every run, saving a significant amount of time.
* Use Mockups for External Services: For unit tests, use mock objects to simulate third-party services instead of making actual network connections. This isolates the test to your own code and runs much faster, as you are testing your application, not the external service's driver.
* Isolate and Optimize Bottlenecks: First, measure your pipeline to identify which specific tests or jobs are the slowest. Focus your optimization efforts only on these bottlenecks to ensure you are making an impact on the total run time.
* Use Containers for Test Environments: Using tools like Docker provides a consistent and fast way to set up test environments. This reduces the time spent on provisioning and helps avoid environment-specific failures.