Kaique Ferreira

Student number 74227

Question 2

This is a screenshot of a terminal on a MacBook Pro running the zsh shell. The user executed the ls -al command in a directory named “sign\_in”. This command lists all files and directories in the current directory, including hidden ones, in a detailed format.

The folders “.” and “..” are default folders that are present whenever you create a new directory. “.” represents the current directory and “..” represents the parent directory.

We can see that there are some user-created folders and some files. For example, README.md is a document file.

The listing format contains several columns:

**First column**: Shows the file permissions. The letter d indicates a directory. The letters “r” represents read, “w” write, and “x” execute permissions, respectively.

**Third column**: Shows the username of the file owner.

**Fifth column**: Shows the size of the file in bytes.

**Sixth to eighth columns**: Show the date and time of the last modification.

**Ninth column**: Shows the name of the file or directory.

The “.git” directory indicates that the project is under version control using Git. This suggests that the project is being versioned, which is a common practice in software development to keep track of code changes.

Basically, this is a screenshot showing the detailed structure of a project directory, where we can see the organisation of files and the information about permissions, owner, size, and modification date provided by the ls -al command in the terminal.

Question 3

**Amazon Web Services (AWS) Instances: Detailed Explanation**

AWS Instances are virtual servers offered by Amazon Web Services (AWS), enabling users to run applications and workloads in the cloud. They are a cornerstone of AWS's Infrastructure as a Service (IaaS) offering, providing scalable computing capacity. These instances are created from Amazon Machine Images (AMIs), which contain all the necessary information to launch the instance, such as the operating system, application server, and applications.

**Key Features of AWS Instances**

**Variety of Instance Types:** AWS provides a range of instance types tailored for different use cases, including general-purpose, compute-optimized, memory-optimized, storage-optimized, and GPU instances.

**Elasticity:** Users can scale instances up or down based on demand, ensuring cost efficiency and flexibility.

**Global Reach:** AWS instances can be deployed in multiple regions and availability zones, ensuring high availability and disaster recovery.

**Integrated Services:** AWS instances can integrate with other AWS services like S3 for storage, RDS for managed databases, and CloudWatch for monitoring and logging.

**Advantages and Disadvantages**

**Advantages**

**Scalability:**

* AWS instances can be scaled up or down according to the workload, ensuring optimal resource utilisation and cost efficiency.
* Auto-scaling features automatically adjust the number of instances in response to changes in demand.

**Cost Efficiency:**

* The pay-as-you-go pricing model allows users to pay only for the compute time they use.
* Various pricing options, including On-Demand, Reserved Instances, and Spot Instances, provide flexibility in cost management.

**Reliability and Availability:**

* High availability and fault tolerance are ensured through the use of multiple availability zones and regions.
* AWS provides a Service Level Agreement (SLA) that guarantees a certain level of uptime.

**Security:**

* AWS instances benefit from AWS's robust security infrastructure, including network firewalls, encryption, identity and access management (IAM), and compliance certifications.

**Ease of Use:**

* The AWS Management Console, Command Line Interface (CLI), and APIs make it easy to manage and automate instances.
* Preconfigured AMIs simplify the process of launching instances with specific configurations.

**Disadvantages**

**Cost Management:**

* While AWS offers cost-efficient options, managing costs can become complex, especially with dynamic scaling and varied pricing models.

**Complexity:**

* The wide range of services and configurations can be overwhelming for new users.
* Effective use of AWS instances often requires a solid understanding of cloud architecture and best practices.

**Dependence on Internet Connectivity:**

* AWS instances are cloud-based, requiring reliable internet connectivity for access and management.
* Latency issues may arise for applications requiring real-time processing if deployed far from the user base.

**Vendor Lock-in:**

* Migrating workloads from AWS to another provider can be challenging due to the proprietary nature of some AWS services and APIs.

**Personal Reflection**

**Current Usefulness**

At present, AWS instances are invaluable for hosting web applications, running development and testing environments, and managing scalable workloads. They allow for:

**Rapid Deployment:** I can quickly spin up instances for various purposes without the need for physical hardware.

**Experimentation:** The flexibility to try different configurations and services without long-term commitments.

**Cost Management:** The ability to shut down instances when not in use, saving costs.

**Future Benefits**

Looking ahead, I anticipate that AWS instances will be increasingly beneficial in several ways:

**Scaling Applications:** As applications grow, the ability to scale effortlessly will be crucial.

**Global Reach:** Expanding services to a global audience will require the geographical flexibility AWS provides.

**Advanced Technologies:** Leveraging advanced AWS services like machine learning (SageMaker), big data (Redshift), and serverless computing (Lambda) will enhance application capabilities.

**Disaster Recovery:** Implementing robust disaster recovery solutions will ensure business continuity.

AWS instances provide a powerful, flexible, and scalable solution for modern computing needs, offering both immediate and long-term benefits. The key is to manage and optimise their use effectively to leverage their full potential.

Question 4

**Docker**

Imagine you're a cook preparing a meal. You want to ensure it tastes consistent no matter where you cook it or who eats it. Docker assists software developers in achieving a goal, with their applications. It enables them to wrap up their applications along with all components to operate them into what’s known as a "container."

Think of a container as a kitchen that contains all the ingredients, tools and recipes. Whether you're cooking at a friend’s place in the outdoors or at a luxury hotel the meal will turn out the same each time. Likewise using Docker gives developers the assurance that their application will function uniformly on their computer on a colleague’s device or, on a cloud server.

For instance, if your application relies on a version of Python, a database system and certain libraries Docker consolidates all these elements into one container. This resolves the age issue of "it works on my machine but not yours" because the container guarantees consistency across all environments.

**CI/CD (Continuous Integration and Continuous Delivery/Continuous Deployment)**

Imagine a relay race where every runner has to hand over the baton to the one. In software development CI/CD guarantees that each code modification undergoes automated testing, integration and deployment similar, to ensuring that each runner seamlessly receives the baton without any mishaps.

Continuous Integration (CI) can be likened to a relay race where runners practice passing the baton repeatedly. Whenever a developer makes changes the code undergoes testing and is merged into the codebase. This frequent testing helps identify issues early, similar, to how practice sessions uncover baton passing problems before the race.

Continuous Delivery (CD) ensures that the code can be handed off to the runner (deployment environment) at any given moment. The code remains constantly ready for deployment although manual activation may still be necessary.

Taking it a step further Continuous Deployment automates the deployment process. Just like in practice sessions the baton is passed automatically during the race, without any manual intervention. Every change that successfully passes automated tests is instantly deployed to production ensuring that the software remains current at all times.

**DevOps**

Imagine a bustling restaurant kitchen where chefs (developers) and waiters (operations) must work in perfect harmony to serve customers (users) efficiently. DevOps is a cultural and professional movement designed to enhance the collaboration between software developers and IT operations.

In a traditional setting, chefs and waiters might work in separate silos, leading to communication breakdowns and delays. Similarly, in conventional IT setups, developers and operations teams often work independently, resulting in deployment delays and software reliability issues.

DevOps practices dismantle these silos. Chefs and waiters communicate constantly, sharing feedback and refining their workflow. In the same way, in a DevOps environment, developers and operations teams work closely together, frequently using automated processes to build, test, and deploy software.

a DevOps pipeline might include automated testing and deployment, ensuring that new features are quickly and reliably integrated into the main application. This continuous feedback loop enables faster and more reliable software delivery, much like a well-coordinated kitchen serving delicious meals promptly.

**SSH (Secure Shell)**

Imagine SSH, as a passage between two points that allows you to journey securely and privately in risky surroundings. SSH, also known as Secure Shell, serves as a method for establishing connections with computers across a network.

When you wish to reach a server for file management or executing commands SSH establishes an encrypted link to a secluded passage to safeguard your data from potential spies. This becomes crucial when handling data.

For instance, if you need to make changes, to a website hosted on a server away SSH enables you to log in and carry out the required updates. This ensures that your login credentials and the data you’re transmitting are encrypted and protected from cyber-attacks.

**Raster vs. Vector Images**

Imagine you’re designing a poster and have two types of images to choose from: a raster image and a vector image. Knowing the difference between these two is essential for creating a high-quality design.

Think of a raster image as a mosaic made up of tiny, individual tiles called pixels. Each pixel has its own colour, and together they form the complete picture. However, if you try to enlarge a raster image, the pixels become more noticeable, and the image can appear blurry or pixelated. Common examples of raster images are photographs and digital paintings, typically in formats like JPEG, PNG, and BMP.

In contrast, a vector image is akin to a set of mathematical instructions for drawing shapes. It’s not composed of pixels but of paths defined by points, lines, and curves. When you scale a vector image up or down, it retains its sharpness and clarity because the mathematical relationships between the points don’t change. This makes vector images ideal for logos, icons, and any graphics that need to be resized frequently. Common vector formats include SVG, EPS, and AI.

For instance, if you’re creating a company logo that will be used on business cards, websites, and billboards, a vector image is the best choice. It ensures the logo looks crisp and professional at any size, unlike a raster image, which might look great on a business card but become blurry on a billboard.

**Summary**

Basically, Docker standardises application environments, ensuring consistency across various platforms. CI/CD automates the testing and deployment process, enhancing software reliability and delivery speed. DevOps fosters collaboration between development and operations teams, streamlining workflows and improving efficiency. SSH provides secure remote access to servers, protecting sensitive data. Lastly, knowing the difference between raster and vector images is essential for creating scalable, high-quality graphics. With all of these concepts, we can form a foundation for efficient, secure, and scalable software development and deployment practices.