1)command to list space details of all files in a Directory?

**du -ah /home/user/documents**

2) linux command to list last 5 restarts?

**last -n 5 reboot**

3) use of sar command in linux?

The **sar** command in Linux is a system monitoring tool that collects, reports, and analyzes system activity. It stands for "System Activity Reporter.

* **sar -u :** View CPU usage statistics:
* **sar -r :** View memory usage statistics:
* **sar -b:** View disk I/O statistics:
* **sar -n DEV:** View network statistics:
* **sar -q:** View overall system activity:

4) how to identify hostname in linux and windows?

**hostname**

**5)** what are all the option to host a static website in AWS?

* Amazon S3 (Simple Storage Service):
* Amazon CloudFront: You can use CloudFront in combination with S3 or an EC2 instance
* AWS Amplify:
* AWS Elastic Beanstalk:
* Amazon EC2

**6)** method to reduce cost in a ec2 creation?

* Choose the right instance type
* Utilize spot instances
* Optimize instance size
* Use Reserved Instances
* Implement auto-scaling
* Leverage AWS Cost Explorer and AWS Budgets
* Schedule instances

**7)** Difference between application load balancer and classic load balancer?

* **Layer 7 Load Balancing**: ALB operates at the application layer (Layer 7) of the OSI model and can make intelligent routing decisions based on content, such as HTTP headers, URL paths, or hostnames. CLB, on the other hand, primarily functions at the transport layer (Layer 4) and performs basic load balancing based on network-level information like IP addresses and ports.
* **Target Groups**: ALB uses target groups to route requests to registered targets (such as EC2 instances, containers, or Lambda functions). It allows more advanced routing rules and enables features like path-based routing and multiple port mapping for a single instance. CLB, however, uses Elastic Load Balancer (ELB) instances to route traffic.
* **WebSockets and HTTP/2**: ALB supports WebSockets, allowing it to handle real-time bidirectional communication between clients and servers. ALB also supports HTTP/2, which enhances performance and efficiency for modern web applications. CLB does not support WebSockets and only supports HTTP/1.1.

**Target Health Checks**: ALB provides more advanced health check options, including the ability to perform HTTP/HTTPS checks on different paths or ports. CLB has limited health check capabilities and only supports basic TCP or HTTP checks.

**8)**  Difference between network load balancer and classic load balancer?

* **Layer of Operation:** The Classic Load Balancer operates at Layer 4 (transport layer) of the OSI model and provides basic load balancing based on IP addresses and ports. The Network Load Balancer, on the other hand, operates at Layer 4 as well but provides more advanced load balancing capabilities, including handling millions of requests per second with ultra-low latencies.
* **Connection Handling:** The Classic Load Balancer uses a fixed set of IP addresses for client connections, whereas the Network Load Balancer uses a dynamically assigned source IP address for each client connection. This allows the Network Load Balancer to maintain the source IP address of the client throughout the request's lifecycle, which is useful for applications that require the preservation of the client IP.
* **Elastic IP Support:** The Classic Load Balancer requires an Elastic IP address to be associated with it, which can be a limited resource in an AWS account. The Network Load Balancer does not require an Elastic IP and can be accessed through Amazon DNS.
* **Port Mapping:** The Classic Load Balancer allows mapping multiple ports on the load balancer to a single port on the backend instances. The Network Load Balancer supports mapping multiple ports on the load balancer to multiple ports on the backend instances.

9)Difference between ALB and NLB?

* Layer of Operation:
  + ALB operates at Layer 7 (application layer) of the OSI model and can make routing decisions based on content, such as HTTP headers, URL paths, or hostnames. It supports advanced application-level features like content-based routing, path-based routing, and host-based routing.
  + NLB operates at Layer 4 (transport layer) of the OSI model and routes traffic based on IP addresses and ports. It is designed for low-latency and high-throughput scenarios and is particularly well-suited for handling TCP and UDP traffic.
* **Load Balancing Algorithms:**

ALB supports a variety of load balancing algorithms, including round robin, least outstanding requests, and least connections. It can intelligently distribute traffic based on the configured algorithm and health checks.

NLB supports flow-based load balancing, which uses a flow hash algorithm to distribute incoming connections to targets. This ensures that the same source IP is always directed to the same target, which can be useful for preserving client IP

* Target Groups and Routing:
  + ALB uses target groups to route requests to registered targets (such as EC2 instances, containers, or Lambda functions). It allows advanced routing rules, including path-based routing and multiple port mapping for a single target.
  + NLB uses listeners and target groups to route traffic. It supports fewer routing options compared to ALB but excels in handling large-scale traffic and high-throughput workloads.
* Cross-Zone Load Balancing:
  + ALB supports cross-zone load balancing, which distributes traffic evenly across availability zones.
  + NLB also supports cross-zone load balancing, but with an additional mode called "static IP" that enables assigning static IP addresses to each availability zone.

**10)** what is the use of terraform back end file and where its storing?

* In Terraform, the backend configuration file specifies where Terraform state files are stored and managed. The state file is used to store the current state of the infrastructure managed by Terraform, including resource metadata and dependencies
* The backend configuration file is defined using the **-backend-config** option in the Terraform command or specified within the **backend** block in the Terraform configuration file (usually named **backend.tf**). It typically contains information such as the backend type, the storage location, access credentials, and other configuration settings specific to the chosen backend.

The backend file can be stored in various locations,

* Local File System: If using the local backend, the backend file can be stored locally on the machine where Terraform is executed. The file path is usually specified as a local file system path.
* Remote Storage Services: Terraform also supports various remote storage backends, such as Amazon S3, Azure Blob Storage, Google Cloud Storage, or HashiCorp Consul. In these cases, the backend file is stored in the respective storage service, and the backend configuration specifies the necessary connection details (e.g., bucket name, key, credentials) to access and manage the state file remotely.
* Terraform Cloud or Terraform Enterprise: If using Terraform Cloud or Terraform Enterprise as the backend, the backend file is stored on the Terraform Cloud or Terraform Enterprise platform. The backend configuration includes the necessary information to authenticate and connect to the specific instance.

11)what is nice command in linux?

The **nice** command is a Linux utility that allows you to adjust the priority of a process. It is used to launch a program with a specific priority level, which determines how much CPU time and resources it will receive compared to other processes running on the system.

The **nice** command is typically used in combination with other commands to start a process with a lower or higher priority. The priority range is typically from -20 to 19, where -20 represents the highest priority (most favorable scheduling) and 19 represents the lowest priority.

* **nice [options] [command]**
* **nice -n -10 command:** start the command with high priority
* **nice -n 10 command:** start the command with low priority

12)Difference between SSH and SCP?

SSH (Secure Shell) and SCP (Secure Copy) are both network protocols used for secure communication and file transfer between systems.

SSH (Secure Shell):

* SSH is a network protocol that provides secure remote login and command execution over an encrypted connection.
* It establishes a secure and authenticated channel between two systems, allowing users to remotely access and manage a remote system's command-line interface.

Eg: ssh username@remote\_host

SCP (Secure Copy):

* SCP is a command-line utility that uses the SSH protocol for secure file transfer between systems.
* It allows users to securely copy files and directories between local and remote systems or between two remote systems.
* SCP uses the same authentication and security mechanisms as SSH, ensuring data confidentiality and integrity during file transfers.
* Once you are connected to the remote system using SSH, you can use SCP to securely copy files.

**Eg:** **scp local\_file username@remote\_host:/remote/path/**

Note: Replace **local\_file** with the path to the file on your local system that you want to copy. Similarly, replace **username**, **remote\_host**, and **/remote/path/** with the appropriate values for the remote system.

13)how to change ssh port to custom port and command to access it?

To change the SSH port to a custom port and access it using the new port, you need to follow these steps:

* Connect to your server using the current SSH port (default is 22).
* Open the SSH configuration file using a text editor. The file location may vary depending on the operating system you're using. Here are some common locations:
  + Ubuntu/Debian: **/etc/ssh/sshd\_config**
  + CentOS/Fedora: **/etc/ssh/sshd\_config**
  + macOS: **/etc/sshd\_config**
* Locate the line that specifies the port and change it to your desired custom port. For example, let's say you want to change it to port 2222:

Remove the '#' at the beginning of the line if it exists, and update the port number accordingly.

* Save the changes and exit the text editor.
* Restart the SSH service to apply the configuration changes. The command to restart the SSH service varies depending on your operating system. Here are some examples:
  + Ubuntu/Debian: **sudo service ssh restart**
  + CentOS/Fedora: **sudo systemctl restart sshd**
  + macOS: **sudo launchctl stop com.openssh.sshd && sudo launchctl start com.openssh.sshd**
* If you have a firewall enabled, make sure to allow incoming connections on the new custom port. The specific steps to configure the firewall depend on the firewall software you're using.

**Command to connect to that machine: ssh -p 2222 user@your\_server\_ip**

14)What is nested stack in AWS?

* In AWS, a nested stack refers to the capability of AWS CloudFormation to include one stack as a resource within another stack.
* This feature allows you to modularize and manage complex AWS infrastructure deployments more efficiently by breaking them into smaller, more manageable units.

**Example Scenario:**

* Imagine you have an application consisting of a front-end web server, multiple backend microservices, and associated databases. Using nested stacks, you could create a parent CloudFormation template that includes separate nested stacks for each component:
* **Parent Stack:** Manages overall deployment and dependencies.
  + **Nested Stack 1:** Front-end web server resources.
  + **Nested Stack 2:** Backend microservices resources.
  + **Nested Stack 3:** Database resources.
* AWSTemplateFormatVersion: '2010-09-09'
* Description: Parent stack including nested stacks for front-end and backend services
* Resources:
* FrontendStack:
* Type: AWS::CloudFormation::Stack
* Properties:
* TemplateURL: !Sub 'https://s3.amazonaws.com/my-bucket/frontend-stack.yaml'
* Parameters:
* Environment: !Ref Environment
* BackendStack:
* Type: AWS::CloudFormation::Stack
* Properties:
* TemplateURL: !Sub 'https://s3.amazonaws.com/my-bucket/backend-stack.yaml'
* Parameters:
* Environment: !Ref Environment
* Parameters:
* Environment:
* Type: String
* Description: Environment name (e.g., dev, prod)
* Outputs:
* FrontendEndpoint:
* Description: Endpoint URL for the front-end application
* Value: !GetAtt FrontendStack.Outputs.EndpointURL
* Export:
* Name: !Sub '${AWS::StackName}-FrontendEndpoint'
* BackendEndpoint:
* Description: Endpoint URL for the backend API
* Value: !GetAtt BackendStack.Outputs.EndpointURL
* Export:
* Name: !Sub '${AWS::StackName}-BackendEndpoint'

15)How to configure OUTPUT.TF file in Terraform?

* In Terraform, the outputs.tf file is used to define output values that are exposed after Terraform completes provisioning and configuring infrastructure. These outputs can be useful for obtaining information about resources created by Terraform, such as IP addresses, endpoint URLs, or other attributes that might be needed by other parts of your infrastructure or external systems.
* Example:

Output.tf file

output "instance\_ip" {

  value = aws\_instance.example.public\_ip

}

output "s3\_bucket\_arn" {

  value = aws\_s3\_bucket.example.arn

}

output "database\_endpoint" {

  value = aws\_rds\_instance.db.endpoint

}

* instance\_ip outputs the public IP address of an AWS EC2 instance named example.
* s3\_bucket\_arn outputs the ARN (Amazon Resource Name) of an AWS S3 bucket named example.
* database\_endpoint outputs the endpoint URL of an AWS RDS instance named db.
* After applying your Terraform configuration (terraform apply), you can access these output values using terraform output command. For example:
* terraform output instance\_ip
* terraform output s3\_bucket\_arn
* terraform output database\_endpoint