DevOps Interview Questions

- Prudhvi vardhan

1. What is web server, what is the difference between Apache and nginix, describe their working mechanism?

Ans: a **web server** is a software program that **delivers web content** to clients **over the internet**.

Apache and **Nginx** are two popular web servers, with some key differences between them. Apache is more **traditional and uses more resources**, while Nginx is **faster and more** efficient due to its event-**driven architecture**. Apache also has a larger ecosystem of modules and extensions, while Nginx's configuration is simpler and more concise. Both **servers handle requests** from clients and respond with the appropriate content, but **Nginx is better** suited for **high-traffic environments** due to its **scalability.** Ultimately, the choice between Apache and Nginx depends on the specific needs of the website or application being developed.

2. What are the meaning of different error codes such as 202,404, 501?

Ans: HTTP status codes are **three-digit number**s that indicate the **outcome of a request** made by a web browser or client to a web server. Here are some commonly known status codes:

1. **202**: The request has been **accepted** for processing, but the **processing** has not been completed yet. This code is often used in **asynchronous** processing, where a server **needs more time to process** a request and **cannot respond immediately.**
2. **404**: The requested resource could **not be found** on the server. This error code is commonly displayed when a **user enters an incorrect URL** or follows a **broken link**. It indicates that the server was **unable to find** the requested resource.
3. **501**: This error code indicates that the server **does not support** the requested method, such as a particular HTTP verb or request type. It

means that the client's **request could not be completed** because the **server does not understand** or cannot support the requested action.

In summary, HTTP status codes provide information on the **outcome** of a client's **request to a web server**. The 202 code indicates that processing is still ongoing, 404 indicates that the requested resource could not be found, and 501 indicates that the server does not support the requested method.

3. Elaborate different EC2 purchasing options?

Ans: When it comes to Amazon EC2 instances, there are three main purchasing options available: **On-Demand, Reserved, and Spot Instances**. Each of these options is designed to meet different needs and offer various benefits.

On-Demand instances allow you to **pay** for compute capacity on **an hourly or per-second basis**, **without** any **upfront payment** or long-term commitment. This option is suitable if your compute requirements are unpredictable, short-term, or irregular.

Reserved Instances require a **one-time upfront payment** in exchange for a significant **discount** on the hourly charge for instances. Users commit to a **specific instance type, operating system**, and **term (**either one or three years). This option is ideal for workloads with predictable and steady-state usage over a **long period**.

Spot Instances allow you to **bid** on unused EC2 capacity, with instances provisioned **when the bid price is higher than the current spot price**. This purchasing option can provide significant **cost savings**, up to **90% compared to On-Demand pricing**. However, you must be prepared for **the possibility of instance termination** if the spot price exceeds your bid price. Spot Instances are ideal for workloads that have flexible timing for their compute jobs.

Overall, each of these EC2 purchasing options has its advantages, and the choice depends on your specific needs, workload requirements, and budget.

4. How AWS lambda works?

Ans: AWS Lambda is a **server less computing** platform that allows you to run your code in the cloud without worrying about **managing servers or infrastructure.** With Lambda, you only pay for the compute time that your code actually uses, making it a cost-effective and efficient solution for running your applications.

Here's how **AWS Lambda** works:

- You start by writing your code in one of the supported programming languages, such as Node.js, Python, Java, or C#. You can also use other tools and frameworks, such as AWS SAM (Server less Application Model), to package and deploy your code**.**
- Once you have your code ready, you can create a Lambda function using the AWS Management Console, CLI, or SDK. You specify the function code, runtime environment, and other configuration settings**.**
- When you **invoke the Lambda function**, **AWS creates a container to run your code and manages all the underlying infrastructure, including scaling and load balancing. You don't need to provision or manage any servers or resources.**
- After the function completes its execution, **AWS destroys the container,** and **you only pay for the compute time that your function used.**

Lambda functions can be triggered by various event sources, such as API Gateway, S3, Kinesis, Dynamo DB, and more. You can also configure your functions to **run on a schedule** using Cloud Watch Events.

In summary, AWS Lambda is a server less computing platform that allows you to run your code in the cloud without managing servers or infrastructure. You write your code, create a Lambda function, and let AWS handle the rest, including scaling, load balancing, and resource management

5. Describe AWS IAM user, groups, policies and roles?

Ans: **AWS Identity and Access Management (IAM)**: It is a service that allows you to **manage access** to your AWS resources securely.

IAM helps you **create** and **manage user accounts, groups, roles, and policies** that control **who can access your resources and what actions they can perform**.

IAM Users: An IAM user is an entity that **represents a person** or an **application that interacts** with AWS resources. IAM users have **unique credentials** (username and password or access key and secret access key) and permissions that **determine what actions they can perform** on AWS resources. IAM users are **often used to grant permissions** to people or applications that need to interact with AWS resources.

IAM Groups: An IAM group is a **collection of IAM users**. You can use groups to **simplify the management of permissions for multiple users** who perform the same job functions. For example, you might **create an "Admin" group and assign permissions to the group instead of individual users.**

IAM Roles: An IAM role is an **AWS identity** that you can **create and assign** to an AWS resource or an IAM user. Roles provide a way to **grant temporary permissions** to access AWS resources **for specific use cases** or scenarios. For example, **you can create a role that allows an EC2 instance to access an S3 bucket, and then assign that role to the instance.**

IAM Policies: An IAM policy is a **document** that defines **permissions and restrictions** for AWS resources. Policies are **attached to users, groups, or roles to grant or deny access to specific** AWS resources or actions. You can use AWS-managed policies or **create your own custom policies that define fine-grained permissions for your resources.**

In summary, AWS IAM allows you to manage access to your AWS resources securely using users, groups, roles, and policies. IAM users are entities that represent a person or application that interacts with AWS resources. IAM groups are collections of IAM users that simplify the management of permissions. IAM roles are temporary permissions granted to an AWS resource or an IAM user. IAM policies are documents that define permissions and restrictions for AWS resources.

6. How we can set up a proper VPC?

Ans: When setting up a **VPC** in AWS, there are a few important things to consider to ensure that you have a properly configured and secure environment for your resources.

1. **Create** a **VPC and subnets** with a specified **IP address range**.
2. Configure **routing tables** to enable communication between resources and the internet.
3. Create and configure **security groups** to control inbound and outbound traffic.
4. Optionally, create **network ACLs (access control lists)** for additional security.
5. **Launch** resources within your VPC and configure them to use the appropriate subnets, security groups, and ACLs.

These steps will help you create a properly configured and secure VPC for your AWS resources.

7. Describe how we can create S3 bucket and Use it?

Ans: Amazon **S3** (**Simple Storage Service):** It is a **cloud-based storage container** that provides a reliable, secure, and scalable way to store and retrieve data from anywhere on the internet. It can be used to store any type of file, from simple text documents to large multimedia files, and can be accessed using a variety of AWS tools and APIs.

To **create** an **S3 bucket**:

- **Log in** to AWS Management Console and navigate to **S3 service**.
- Click "Create Bucket" and enter a **unique name and region**.
- Choose access control and any additional options.
- Click **"Create Bucket".**

To **use** an **S3 bucket**:

- **Upload files** and create folders to organize data.
- **Grant permissions** to other AWS accounts or users.
- Use **S3 lifecycle policies** to automatically move or delete data.
- Use **S3 events** to trigger other AWS services or applications.

This will help you create an S3 bucket and use it effectively for your storage needs.

8. Why we use code commit, code deploy and code build, how

We integrate all of them?

Ans: Code Commit, Code Deploy, and Code Build are used together in a "**continuous integration and deployment pipeline"** to **automate**.

The process of building, testing, and deploying software.

- **Code Commit** is for **storing** and **versioning code**.
- **Code Build** is for **building** and **testing code**.
- **Code Deploy** is for **deploying** code to different environments.

When used together, Code Commit, Code Deploy, and Code Build provide a seamless end-to-end CI/CD pipeline. By integrating these services, developers can quickly and easily build, test, and deploy their software, which leads to faster delivery of high-quality applications.

9. What is the purpose of Code Pipeline when we can already use Code Commit, Code Build, and Code Deploy together to create a continuous integration and deployment pipeline?

Ans: **Code Pipeline** is a fully-managed service that provides an **integrated continuous delivery pipeline** to **automate** the software **release process**.

While Code Commit, Code Build, and Code Deploy are **individual services** that can be used together.

Code Pipeline ties them all together into one continuous delivery workflow. It allows developers to define the **stages** of their release process and automate the transition of code changes through each stage, from build to testing to deployment.

Code Pipeline provides **an end-to-end view** of the entire software release process and helps to reduce errors and accelerate releases.

10. What is IAC, describe Terraform workflow?

Ans: IAC (Infrastructure as Code) which is the practice of defining infrastructure resources like **servers**, **databases**, and **networks** in a code format. By using IAC tools like **Terraform**, infrastructure can be **version-controlled, tested, and automated, leading to faster and more reliable infrastructure changes.**

Terraform is a popular **IAC tool** that allows developers to define **infrastructure as code and automate** the provisioning and **management of resources in a cloud environment.**

- ❖ The Terraform workflow involves writing code in a human-readable language to **describe the desired infrastructure,** running a command to generate an execution plan, reviewing and approving the plan, and applying the changes to the infrastructure.
- ❖ Terraform tracks the **state** of the infrastructure and **allows for changes** to **be made through code instead of manual processes**.
- ❖ This helps to **improve the consistency** and reliability of infrastructure changes and reduce errors caused by manual intervention.

11. Can You Explain or Elaborate Various Terraform Commands?

Ans: Terraform has several core commands used in its workflow:

- **terraform init** initializes a **new** or existing **Terraform configuration**. It downloads and **installs** the necessary providers and modules needed to manage the **infrastructure**.
- **terraform plan** creates an **execution plan** of what resources will be created, modified, or deleted when Terraform is applied. It helps **detect any issues before applying** the changes to the infrastructure.
- **terraform validate** checks the configuration files for **syntax** and other **errors**. This command is useful for **catching any errors** before running **terraform plan** or **apply**.
- **terraform apply** executes the **changes defined** in the configuration files. It creates, modifies, or deletes the infrastructure resources. **Terraform apply** can be run after **terraform plan** to apply the execution plan.

- **terraform destroy** **deletes** all the resources created by the configuration files. This command is **useful for cleaning up resources that are no longer needed.**

By using these core commands in combination with other Terraform features, developers can more easily manage and automate their infrastructure and improve their overall efficiency.

12. What are attributes and arguments in Terraform?

Ans: In Terraform, attributes and arguments are used to **define and manage resources** in infrastructure as code.

- **Attributes** are **read-only values** that represent the **current state** of a resource. They are used to **retrieve information about the resource, such as its IP address or status.**
- **Arguments** are the parameters used to **create or modify a resource**. They are defined in the **Terraform configuration files and can include options like region, size, and other settings.**

Attributes and arguments are used together to manage resources in a **declarative way**.

When changes are made to the configuration files, **Terraform compares the new desired state with the current state and determines the necessary actions to update the infrastructure.**

By using attributes and arguments, developers can create more flexible and maintainable infrastructure as code.

13. What is Terraform taint? And how it is useful?

Ans: **Terraform's taint** command is used to **mark a resource** managed by Terraform as "tainted," which means that Terraform **will destroy and recreate that resource on the next terraform apply run.**

This command is useful when a **resource becomes corrupted** or there are **issues with it that cannot be resolved by updating the configuration files**.

By using **taint**, the resource is deleted **and recreated with the new configuration settings**. This can help to resolve issues with the resource and ensure that it is in the correct state.

In short, **taint** is a powerful command that can help to manage problematic resources and ensure that they are in the desired state.

14. Difference between Docker and kubernetes?

Ans: **Docker** is a **containerization platform** that packages applications and their dependencies into **container images**.

Kubernetes is **a container orchestration platform** that **automates** the **deployment, scaling, and management** of containerized applications across a **cluster of nodes**.

- While Docker provides **consistency and isolation** for applications,
- Kubernetes manages containers at **scale, providing tasks such as load balancing, automatic scaling, rolling updates, and self-healing.**

Together, Docker and Kubernetes provide a powerful and scalable way to build, deploy, and manage modern applications in the cloud.

15. Explain deployment, replica set in K8's?

Ans: Deployment and Replica Set are both important concepts in Kubernetes **for managing containerized applications**:

- A **Deployment** is a Kubernetes object that **describes** the **desired state of a set of pods**. It manages the deployment of new replicas and the scaling of existing replicas. **Deployments are useful for managing application updates and rolling out changes in a controlled and automated way.**
- A **Replica Set** is a Kubernetes object that ensures that a **specified number of pod replicas are running at all times.** It is responsible for **creating and scaling replicas of a pod**. Replica Sets work together with Deployments to ensure that the desired number of replicas are running, and if not, it creates or destroys them as needed.

Together, Deployments and Replica Sets help to automate the deployment and scaling of containerized applications, ensuring that they are **always available and running** smoothly. **They provide a scalable, reliable, and automated way to manage applications in a Kubernetes cluster**.

16. Describe k8s services and ingress and store objects on it?

Ans: Kubernetes Services and Ingress are used to **manage networking** in a Kubernetes cluster:

- A **K8's Service** is a Kubernetes object that **exposes an application running on a set of pods as a network service.** Services allow for communication between different parts of the application, both inside and outside of the cluster. They can be used to **load balance traffic between pods, provide a stable IP address for a set of pods, and manage access to the application.**
- An **Ingress** in Kubernetes is a resource that allows you to expose HTTP and HTTPS routes from **outside the cluster to services within the cluster**. Ingress can be used **to manage SSL/TLS encryption, load balancing, and URL-based routing.**

The Three Ways to Make a Service accessible externally.

The first method, involves setting the Service type to **Node Port**, where each node in the cluster opens a port and redirects traffic to the underlying service. **The second method**, is to set the Service type to **Load Balancer**, which provides a dedicated load balancer that redirects traffic to the node port across all nodes.

Lastly, we will delve into the creation of an **Ingress resource**, which offers a distinct approach to exposing multiple services through a single IP address and operates at the network layer 7, providing more advanced features than layer 4 services.

- **Kubernetes storage options**: It has **Persistent Volumes** and **Persistent Volume Claims**, **to store objects**. These objects can be used to store data, logs, and other important information required by the application.

In summary, Services and Ingress provide a scalable and reliable way to manage networking in a Kubernetes cluster, allowing for communication between different parts of the application and managing external access to the services.

Kubernetes also provides several storage options to store objects required by the application.

17. Do you know RBAC, can you explain it?

Ans: **RBAC (Role-Based Access Control):** It is an approach that is used for **restricting access to users and applications on the system/network**.

RBAC is used by Kubernetes for **authorization**. For example giving access to a user, adding/removing permissions and setting up rules, etc.

So basically, **it adds security to a Kubernetes cluster**. RBAC in Kubernetes is the way that you **restrict who can access what within the cluster.**

RBAC is a useful way to **manage access** to resources in large organizations because it's scalable and easy to maintain. Instead of managing access on an individual basis, you can set up roles and assign access to those roles. This makes it easier to manage and audit access to resources.

18. What is the difference between ansible and Terraform? Explain in detailed?

Ans: **Ansible** is a **configuration management tool** that automates the deployment and configuration of software. It's used to manage systems, install software, and perform other tasks related to infrastructure management. Ansible uses a simple, declarative language called **YAML to define tasks and configurations.**

Terraform is a tool for managing **infrastructure as code.** It's used to define and provision infrastructure resources, such as virtual machines, containers, and storage, across different cloud providers. Terraform uses a domain-specific language called **HCL** (HashiCorp Configuration Language) **to define resources and dependencies.**

In short, **Ansible is more focused on software configuration management,** while **Terraform is focused on infrastructure management**. They can both be used together to manage a complete infrastructure, but they serve different purposes and have different strengths.

19. What is cloud computing and what do you mean by virtualization?

Ans: **Cloud computing** means using computers and other resources that are located somewhere else, instead of having them right next to you. You can access these resources over the internet, which means you can use them from anywhere.

Virtualization is a special way of using computers and other resources that makes them work more efficiently. It lets you split up a computer into lots of smaller pretend-computers that can all be used at the same time, without getting in each other's way. This makes it easier for lots of people to use the same computer without any problems

20. Can you Write playbook for Apache and Nginix Installation in Ansible?

Ans: Ansible playbook that installs Apache and Nginx on Red Hat:

```
---
- name: Install Apache and Nginx
  hosts: all
  become: true

  tasks:
    - name: Install Apache
      yum:
        name: httpd
        state: present

    - name: Start Apache
      service:
        name: httpd
        state: started
        enabled: true

    - name: Install Nginx
      yum:
        name: nginx
        state: present

    - name: Start Nginx
      service:
        name: nginx
        state: started
        enabled: true
```

21. Difference Between Application Load balancer and Network Load balancer?

Ans: **Application Load Balancer (ALB)** is a layer 7 load balancer. It load balances HTTP requests. When a client opens a connection to the load balancer ALB reads and identifies HTTP requests, and distributes the HTTP requests across backend targets according to your rules.

Network Load Balancer (NLB) is a layer 4 load balancer. It load balances incoming connections. When a client opens a connection to the load balancer it opens a connection to a backend target, and forwards all packets from the client to the target

22. What are the different types of ports in devOps?

Ans: There are various ports in devOps mainly :

- **Jenkins**: 8080 (default) or custom port
- **SonarQube**: 9000 (default) or custom port
- **Ansible**: uses SSH (Secure Shell) port 22
- **Apache**: 80 (HTTP) or 443 (HTTPS) or custom ports
- **Tomcat**: 8080 (default) or custom port
- **Nexus**: 8081 (default) or custom port
- **Docker**: 2375 (default for unsecured Docker daemon) or 2376 (default for secured Docker daemon) or custom ports
- **Kubernetes**: 6443 (default for Kubernetes API server) or custom port
- **Terraform**: uses HTTP (port 80) or HTTPS (port 443) protocols
- **Node Port**: dynamically allocated port number (30000-32767 by default) or custom port
- **Load Balancer**: varies depending on the cloud provider and the type of load balancer used. For example, AWS Elastic Load Balancer (ELB) uses port 80 and 443 by default for HTTP and HTTPS traffic.

These port numbers are configurable and can be changed as per the specific needs of the application or infrastructure. Also, this is not an exhaustive list and there may be other ports used by DevOps tools depending on their configuration and setup.