Kubernetes:

1. What is replication controller and ReplicaSet?
2. How many types of Services we have in Kubernetes, name them?
3. What is the difference between pod and container?
4. What is the difference between virtualization and Containerization?

Terraform:

1. What does the terraform init do?
2. What is the difference between plan and apply?
3. How do we secure terraform state file?
4. How do you define local variables in terraform?

Azure:

1. What is difference between api gateway and app gateway?
2. What is the difference between service endpoint, private end point and private link?
3. What are the storage types available in azure?
4. what is CDN why do we use that?

Pipelines:

1. what CI/CD pipelines you worked on, explain the stages Involved?

Docker:

1. what is docker prune?
2. Difference between docker add and copy?

Motivity lab:

**Azure DevOps:**

What is deployment group?

Explain Azure DevOps Pipeline like stages, build, release pipeline?

Explain how to release dev to stage to production?

**Kubernetes:**

1. How many services there in k8s ?
2. What is ingress controller in k8s?
3. How to pull image from private cluster in k8s? 1. service connection establishing

**Docker:**

What is docker file?

What is different b/w add and copy?

What is different b/w cmd and entrypoint

What is difference b/w env and argument?

IBM:

1.Explain CICD pipeline from end to end?

2.What is statefile and benefits?

3.What is backend file?

4.What is provider in terraform?

5.How to delete vnet ,subnet in terraform?

Ans: Terraform destroy

If you want to delete only specific resources (e.g., just the subnet), you can target them using the **-target** flag:

Ans: terraform destroy -target=azurerm\_subnet.example

6.What is terraform refresh command?

Ans:   
In Terraform, the **terraform refresh** command is used to update the state file with the real-world infrastructure. When you run **terraform apply**, Terraform makes changes to your infrastructure, and **terraform refresh** helps to update the state file to reflect the current state of the real-world resource

Terraform refresh

What is statefile and benefits?

In Terraform, the state file is a critical component that stores information about the infrastructure managed by Terraform. It contains a representation of the resources that Terraform has created, modified, or is managing. The state file is typically a JSON file named **terraform.tfstate**, and its purpose is to keep track of the current state of your infrastructure.

Here are some key aspects and benefits of Terraform state files:

1. **Tracking Resource State:**
   * The state file keeps track of the current state of your infrastructure, including details about resources such as Virtual Machines, networks, storage, etc.
   * It records attribute values of resources, dependencies, and metadata.
2. **Concurrency and Collaboration:**
   * Terraform uses the state file to understand the existing infrastructure so that it can make accurate and safe changes during the **terraform apply** process.
   * When working in a team or collaborative environment, the state file enables multiple users to work on the same infrastructure by sharing the same state. This allows changes to be made and applied consistently.
3. **Dependency Management:**
   * Terraform uses the state file to manage dependencies between resources. It understands the relationships between different resources and ensures they are created or modified in the correct order.
4. **ID Mapping:**
   * Each resource created by Terraform is assigned a unique identifier (ID). The state file maps resource names to their corresponding IDs, enabling Terraform to track and manage resources accurately.
5. **Remote State:**
   * In production scenarios or when working in a team, it's common to store the state file remotely. This could be in a shared storage service, such as Azure Storage or AWS S3, or using a backend like Terraform Cloud. This ensures that the state is consistent across all team members and environments.
6. **Plan and Apply Operations:**
   * During the **terraform plan** and **terraform apply** operations, Terraform reads the current state file to understand the existing infrastructure and determine the necessary changes to achieve the desired state.
7. **Rollback and Recovery:**
   * In case of failures or errors during the **terraform apply** process, the state file provides a way to understand the current state of the infrastructure and potentially rollback changes. It aids in recovery and debugging.

**Terraform**:

1.What is difference b/w provider and provisioner?

**Provisioner:**

1. **Definition:**
   * A provisioner in Terraform is responsible for executing scripts or commands on a local machine or a remote resource after a resource is created. Provisioners are used to perform tasks such as configuration management, initialization, or custom actions.
2. **Examples:**
   * Common Terraform provisioners include:
     + **local-exec** for executing commands on the local machine where Terraform is run.
     + **remote-exec** for executing commands on a remote resource (e.g., a virtual machine)

2. How to create resource in multiple instances using terraform?

In Terraform, you can create multiple instances of a resource using the **count** parameter or by using the **for\_each** expression, depending on your specific use case.

Let me explain both approaches:

Using “Count”

**resource "aws\_instance" "example" {**

**count = 3**

**ami = "ami-0c55b159cbfafe1f0"**

**instance\_type = "t2.micro"**

**}**

**In this example, three instances of the aws\_instance resource will be created. The instances will be identical, and the count parameter specifies how many instances to create.**

**The instances will be named**

**aws\_instance.example[0],**

**aws\_instance.example[1],**

**and aws\_instance.example[2].**

**Using for\_each:**

variable "instance\_names" {

type = list(string)

default = ["instance-1", "instance-2", "instance-3"]

}

resource "aws\_instance" "example" {

for\_each = toset(var.instance\_names)

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

tags = {

Name = each.value

}

}

**In this example, you use the for\_each expression with a set of instance names provided through a variable. This approach allows you to create instances with different configurations. The instances will be named according to the values in the instance\_names variable.**

**Choose the approach that best fits your requirements.**

**If you need identical instances, use count.**

**If you need instances with different configurations or names, use for\_each.**

**Additionally, note that for\_each is more flexible and powerful, but it requires Terraform 0.12.6 or later. If you're using an older version, you might need to upgrade to take advantage of for\_each**