DevOps Questions and Answers

1. What would you do if an EC2 instance is getting slow?

- Check instance metrics in CloudWatch (CPU, Memory, Disk IO, Network IO).
- Verify if the application or processes on the instance are consuming excessive resources.
- Consider resizing the instance (vertical scaling) if it's under-provisioned.
- Optimize the application or processes to reduce resource usage.
- Ensure there are no network bottlenecks or misconfigurations.
- Perform system diagnostics using tools like `top`, `iotop`, or `iotstat`.

2. If users can't access an application hosted on EC2, what steps would you take?

- Check the instance's status in the AWS Management Console.
- Verify the security group and network ACL rules.
- Ensure the instance is in the correct VPC and subnet.
- Confirm the application is running and listening on the expected ports.
- Test connectivity using tools like `ping` and `curl`.
- Analyze the application logs for errors.

3. What's the difference between a Load Balancer and a Reverse Proxy?

- **Load Balancer**: Distributes incoming traffic across multiple servers to improve availability and scalability.
- **Reverse Proxy**: Forwards client requests to backend servers and can perform caching, SSL termination, and request filtering.

4. How would you write a Terraform script to create an EC2 instance and run a script on every reboot?

```hcl
resource "aws_instance" "example" {
  ami           = "ami-0abcdef1234567890"
  instance_type = "t2.micro"
  user_data = <<-EOF
    #!/bin/bash
    echo "Running startup script..."
    /path/to/your-script.sh
  EOF
  tags = {
    Name = "example-instance"
  }
}
```

5. What is a Backend in Terraform, and why is it used?

- **Backend**: Configures where Terraform stores its state and performs operations (e.g., `local`, `S3`, `Consul`).
- Used for collaboration, state locking, and consistency across teams.

6. What is the Docker lifecycle?

1. **Create**: Define a container.
2. **Start**: Launch a container.
3. **Run**: Execute processes inside the container.
4. **Pause/Unpause**: Temporarily halt or resume processes.
5. **Stop**: Gracefully shut down a container.
6. **Kill**: Forcefully stop a container.
7. **Remove**: Delete a container.

7. What are the key Docker components?

- **Docker Engine**: Core runtime for building and running containers.
- **Docker Images**: Immutable templates for containers.
- **Docker Containers**: Running instances of Docker images.
- **Docker Registry**: Storage for images (e.g., Docker Hub).

8. What's the difference between a Docker Image and a Docker Container?

- **Docker Image**: Blueprint of a container (read-only).
- **Docker Container**: Instance of an image running as a process (read-write).

9. What should you do before creating a Docker container?

- Define a clear `Dockerfile`.
- Optimize the base image to minimize size.
- Test the application dependencies locally.
- Secure the image by removing unnecessary components.

10. What is Docker Compose, and how do you use it?

- Tool for defining and running multi-container Docker applications.
- Use a `docker-compose.yml` file to specify services, networks, and volumes.
- Run with `docker-compose up`.

11. What steps would you take if you see an "unhealthy" status in an ELB?

- Check the health check configuration.
- Verify that the backend instances are running and reachable.
- Analyze application logs for errors.
- Ensure the security group rules allow traffic from the ELB.

12. How do you optimize Docker images for better performance?

- Use smaller base images (e.g., `alpine`).
- Minimize layers in the `Dockerfile`.

- Combine commands using `&&` to reduce layers.
- Remove unnecessary files and dependencies.

13. How would you secure a Docker container?

- Run containers as non-root users.
- Use signed images from trusted registries.
- Set resource limits for CPU and memory.
- Use network namespaces for isolation.
- Regularly update the Docker Engine.

14. What is Jenkins scaling, and how do you achieve it?

- Adding nodes to handle more builds and distribute workloads.
- Use Jenkins's built-in support for distributed builds and configure master-node architecture.

15. What is the role of the Master and Node in Jenkins?

- **Master**: Manages jobs and orchestrates build pipelines.
- **Node**: Executes build jobs assigned by the master.

16. What is a Sidecar container, and when would you use it?

- A secondary container that provides additional functionality to the main container, such as logging, monitoring, or caching.

17. What is the difference between ConfigMap and Secrets in Kubernetes?

- **ConfigMap**: Stores plain text configuration data.
- **Secrets**: Stores sensitive data (e.g., passwords, tokens) in base64-encoded format.

18. What is the default deployment in Kubernetes?

- A `Deployment` object, which manages ReplicaSets and ensures a desired state for applications.

19. What are Taints and Tolerations in Kubernetes?

- **Taints**: Prevent pods from being scheduled on nodes.
- **Tolerations**: Allow pods to bypass taints and run on tainted nodes.

20. What is a Static Pod in Kubernetes, and how is it different from a regular pod?

- **Static Pod**: Managed directly by the kubelet and defined in a node's configuration.
- **Regular Pod**: Managed by the Kubernetes API server.

21. How do you check pod logs and attach Prometheus for monitoring?

- Use `kubectl logs <pod-name>` to view logs.
- Attach Prometheus by configuring it to scrape pod metrics through annotations or service monitors.

22. How would you define a ConfigMap and Secrets in Kubernetes?

- **ConfigMap**: Use `kubectl create configmap` or YAML to define.
- **Secrets**: Use `kubectl create secret` or YAML to define, with base64-encoded data.

23. What is the default scaling in Kubernetes, and how does it work?

- **Horizontal Pod Autoscaler (HPA)**: Adjusts pod replicas based on CPU or memory usage.
- Scaling is triggered by metrics from the Kubernetes Metrics Server.

24. What is RBAC in Kubernetes, and why is it important?

- **Role-Based Access Control (RBAC)**: Manages user and application access to Kubernetes resources.
- Ensures security by granting minimal required permissions.

25. What's the difference between ClusterRole and Role in RBAC?

- **Role**: Defines permissions within a specific namespace.
- **ClusterRole**: Defines permissions across the entire cluster or non-namespaced resources.