Q. What are the Roles and responsibilities of a DevOps Engineer on daily basis

The daily roles and responsibilities of a DevOps Engineer can vary depending on the specific organization, but some common tasks and duties may include:

1. Collaborating with cross-functional teams to design, develop and deploy software releases and infrastructure changes.
2. Writing automation scripts to streamline the deployment and delivery processes.
3. Implementing and maintaining continuous integration and delivery pipelines.
4. Monitoring and analyzing system performance and taking corrective actions when necessary.
5. Managing and configuring cloud-based infrastructure and services such as AWS, Azure or Google Cloud.
6. Maintaining and improving source control systems, such as Git or SVN.
7. Managing and automating software and infrastructure testing.
8. Documenting processes and procedures for development and operations teams.
9. Keeping up to date with new technologies and industry trends.
10. Troubleshooting and resolving technical issues across multiple environments and platforms.

Overall, the DevOps Engineer is responsible for ensuring that the software development and operations teams work together efficiently and effectively to deliver high-quality software and infrastructure changes in a timely manner.

GIT Commands

Here are some commonly used Git commands:

1. git init: initializes a new Git repository.
2. git clone: creates a copy of an existing Git repository on your local machine.
3. git add: adds changes made to the code to the staging area.
4. git commit: creates a new commit with the changes made in the staging area.
5. git push: uploads the changes made in the local repository to the remote repository.
6. git pull: downloads changes made in the remote repository to the local repository.
7. git branch: creates a new branch in the Git repository.
8. git checkout: switches between branches in the Git repository.
9. git merge: merges changes made in one branch to another branch.
10. git status: displays the current status of the Git repository.
11. git log: shows a log of the commits made to the Git repository.
12. git remote: displays the list of remote repositories connected to the Git repository.
13. git fetch: downloads changes made in the remote repository to the local repository without merging them.
14. git stash: temporarily saves changes made to the code without committing them.
15. git reset: removes changes made to the code from the staging area.

These are just a few examples of the many Git commands available. For more information, you can refer to the official Git documentation or use the command "git help" followed by the command name to get help on a specific command.

AUTOSCALLING AWS :

Q1. What are the errors we could face on AWS Autoscaling

There are several errors that could occur when working with AWS Auto Scaling:

1. Capacity errors: Capacity errors occur when the Auto Scaling group is unable to launch or terminate instances to meet demand. These errors may include insufficient capacity, launch configuration errors, or instance launch failures.
2. Health check failures: Health check failures occur when instances in the Auto Scaling group fail health checks. These checks are used to ensure that instances are running and responding to requests. If an instance fails a health check, it may be terminated and replaced with a new one.
3. Scaling policy errors: Scaling policy errors occur when there are issues with the scaling policies that are defined for the Auto Scaling group. These errors may include invalid policies, incorrect scaling settings, or incorrect target tracking configuration.
4. Configuration errors: Configuration errors occur when there are issues with the configuration of the Auto Scaling group or its associated resources. These errors may include issues with load balancer configuration, security group configuration, or IAM permissions.
5. Network errors: Network errors occur when there are issues with the network configuration for the Auto Scaling group. These errors may include issues with routing, DNS, or VPC configuration.

It is important to monitor the Auto Scaling group for these errors and take corrective action as needed to ensure that the group is functioning correctly.

Q2. Reason for health check error on AWS autoscaling

There can be several reasons why an AWS Auto Scaling Group may face health check errors, such as:

1. Unhealthy instance: The instance may have failed its health check due to issues such as network connectivity, disk space, or high CPU utilisation.
2. Load balancer configuration: The load balancer may be configured incorrectly, which can cause health check failures.
3. Security group rules: The security group rules may be configured in such a way that the instance is not able to communicate with the load balancer or other instances, leading to health check errors.
4. Auto Scaling Group configuration: The Auto Scaling Group may be configured with incorrect or outdated settings, such as the health check type or frequency.
5. Application issues: The application running on the instances may have issues that cause the health check to fail, such as software bugs or crashes.
6. Other external factors: External factors such as network latency, DNS issues, or AWS service disruptions may also cause health check errors.

To troubleshoot health check errors on AWS Auto Scaling, it is important to check the logs and metrics of the instances, load balancers, and Auto Scaling Group. It may also be helpful to consult AWS documentation and support resources to identify and resolve any issues.

Q3. what are the error we might face on AWS Load Balancer

There are various errors that can occur on AWS Load Balancer, some of the common ones are:

1. 502 Bad Gateway Error: This error occurs when the load balancer is unable to establish a connection with the target instance.
2. 503 Service Unavailable Error: This error occurs when the target instance is not responding or has stopped responding to requests.
3. SSL Certificate Error: This error occurs when the SSL certificate installed on the load balancer is invalid or has expired.
4. Internal Server Error (HTTP 500): This error occurs when there is an issue with the application running on the target instance.
5. Connection Timeout Error: This error occurs when the load balancer is unable to establish a connection with the target instance within the configured timeout period.
6. Invalid HTTP Header Error: This error occurs when the HTTP header sent by the client is invalid or malformed.
7. Cross-Zone Load Balancing Error: This error occurs when there is an issue with the configuration of cross-zone load balancing, which is used to balance traffic evenly across all the target instances in different availability zones.

KUBERNETES

Q1. explain about Kubernetes Architecture, how it works

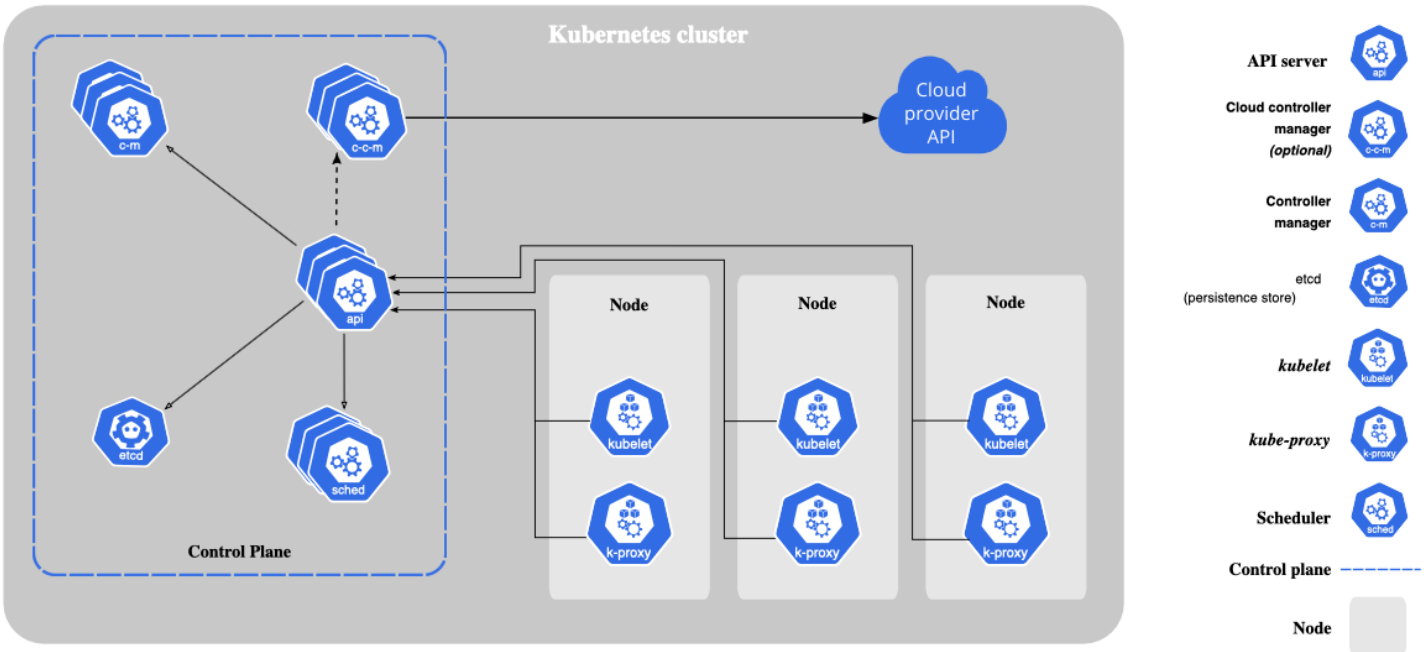
Kubernetes is an open-source container orchestration platform that enables the management and deployment of containerized applications. It follows a distributed architecture model that consists of various components working together to provide a robust and scalable infrastructure for running containers. Here is an overview of the key components in the Kubernetes architecture:

1. Master Node:
2. The master node is responsible for managing the Kubernetes cluster. It controls the overall state and coordinates the cluster's activities. The main components on the master node are:
    - API Server: Acts as the central control point for the cluster and exposes the Kubernetes API.
    - etcd: A distributed key-value store that stores the cluster's configuration and state information.
    - Controller Manager: Manages different controllers that handle cluster-level operations such as scaling, replication, and self-healing.
    - Scheduler: Assigns pods to nodes based on resource availability and scheduling policies.
3. Worker Node:
4. Worker nodes, also known as minion nodes, are the worker machines that run the applications in containers. They are managed by the master node and execute tasks assigned by the master. The main components on worker nodes are:
    - Kubelet: Responsible for communication between the master node and the worker node. It manages the containers and ensures they are running correctly.
    - Container Runtime: The underlying software that runs the containers, such as Docker or containerd.
    - Kube Proxy: Handles network communication for services running on the node.
5. Pod:
6. A pod is the smallest deployable unit in Kubernetes. It represents one or more containers that are deployed together on the same worker node and share the

same network namespace. Pods are scheduled onto worker nodes and are the atomic unit of deployment in Kubernetes.

7. ReplicaSet:
8. ReplicaSet ensures that a specified number of pod replicas are running at any given time. It is responsible for scaling pods up or down based on the desired state defined in the configuration.
9. Service:
10. A service defines a logical set of pods and provides a stable network endpoint to access them. It enables load balancing and service discovery within the cluster.
11. Volume:
12. A volume is used to persist data beyond the lifecycle of a pod. It provides storage for containers and allows data to be shared between containers within the same pod.
13. Namespace:
14. Namespaces provide a way to divide cluster resources into virtual clusters. They help in organizing and isolating applications, resources, and policies within a cluster.

These are just some of the key components in the Kubernetes architecture. Kubernetes is highly modular and extensible, allowing additional components and features to be added as needed. It provides a scalable and flexible platform for deploying and managing containerized applications.



Q2. Explain about Kubelet on k8s

Kubelet is a component of Kubernetes that runs on each node in the cluster. Its main responsibility is to manage the state of the pods that are running on the node. Specifically, the kubelet performs the following tasks:

1. Communicates with the Kubernetes master to receive instructions on which pods to run on the node.
2. Downloads the necessary container images for the pods from a container registry.
3. Starts the containers for the pods and monitors their status, restarting them if necessary.
4. Mounts the volumes required by the pods and manages their lifecycle.
5. Reports the status of the node and the pods to the Kubernetes master.

In summary, the kubelet is responsible for ensuring that the pods on the node are running as expected, and for communicating their status back to the rest of the cluster.

Q3. Use cases of Init Container on Kubernetes

An init container is a special type of container in Kubernetes that runs before the main container(s) in a pod. Its primary purpose is to perform pre-initialization tasks that are necessary before the main application containers can start running. Here are some common use cases for init containers:

1. Database setup: Init containers can be used to set up and initialize databases, such as creating a schema, populating initial data, or configuring database connections.
2. Configuration files: Init containers can be used to populate configuration files or environment variables that are required by the main containers in the pod.
3. Dependencies: Init containers can be used to install dependencies that are required by the main application containers.
4. Health checks: Init containers can be used to perform pre-startup health checks and ensure that all dependencies are available before the main application containers start running.
5. Secret management: Init containers can be used to pull secrets from a secret store, such as Kubernetes Secrets or Vault, and make them available to the main containers.
6. Image preparation: Init containers can be used to prepare the image for the main application container, such as pulling required files or packages, or setting up the environment.

Overall, init containers provide a flexible and powerful way to ensure that all dependencies are met and the environment is properly configured before the main application containers start running, leading to more reliable and efficient deployments.

Docker

Q. How to get a public ip of a Docker container

By default, Docker containers are not assigned public IP addresses. They are isolated from the host machine and other containers by a virtual network. However, you can map a container port to a host port to allow external access to a service running inside the container. Here's how to do it:

1. Start a container and map a port:

```
php
```

```
docker run -p <host_port>:<container_port> <image_name>
```

For example, to start an Apache HTTP server container and map the host port 80 to the container port 80:
arduino

```
docker run -p 80:80 httpd
```

2. Find the IP address of the host machine:
   css

   ```
   hostname -I
   ```

   This will return the IP address of the host machine, for example: `192.168.0.100`
3. Access the container from a web browser or a tool like cURL using the host IP address and the mapped port. In our example, you can access the Apache server by visiting `http://192.168.0.100` in a web browser.

Note that if you're running Docker on a cloud provider like AWS or Azure, you may need to configure the security group or firewall rules to allow incoming traffic to the mapped port.

Q2. How to increase a Docker container volume
To increase the volume of a Docker container, you can follow these steps:

1. Stop the container: Use the `docker stop` command to stop the container that you want to increase the volume size for.
   arduino

   ```
   docker stop <container_name>
   ```

2. Remove the container: Use the `docker rm` command to remove the container, but keep the volume.
   bash

   ```
   docker rm <container_name>
   ```

3. Create a new container with a larger volume size: Use the `docker run` command to create a new container with a larger volume size than the original container. You will need to specify the new volume size using the `-v` option.
   php

   ```
   docker run -v <new_volume_size>:<path_inside_container> --name <new_container_name> <image_name>
   ```

   For example, to create a new container with a volume size of 5GB and mount it to the `/data` directory inside the container, you would run:
   php

```
docker run -v 5GB:/data --name <new_container_name> <image_name>
```

4. Copy data from the old volume to the new volume: Use the `docker cp` command to copy the data from the old volume to the new volume.
ruby

```
docker cp <old_container_name>:<path_inside_container>
<new_container_name>:<path_inside_container>
```

For example, to copy data from the old container's `/data` directory to the new container's `/data` directory, you would run:
ruby

```
docker cp <old_container_name>:/data <new_container_name>:/data
```

5. Start the new container: Use the `docker start` command to start the new container.
php

```
docker start <new_container_name>
```

You should now have a new container with a larger volume size and the data from the old container copied over to the new volume.

Q3. What is Docker compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you define a YAML file that contains all the services, networks, and volumes required for your application, and then use a single command to create and start all the containers defined in the YAML file. Compose simplifies the process of managing multiple containers, allowing you to easily define the relationships between the services in your application and ensuring that they are started in the correct order. This makes it much easier to develop, test, and deploy complex applications that require multiple containers.

Q4. Docker Commands

Here is a list of some commonly used Docker commands:

- `docker run`: Runs a command in a new container.
- `docker start`: Starts a stopped container.
- `docker stop`: Stops a running container.
- `docker ps`: Lists all running containers.
- `docker ps -a`: Lists all containers, including stopped ones.
- `docker images`: Lists all images on your local system.
- `docker pull`: Downloads an image from a registry.
- `docker push`: Uploads an image to a registry.
- `docker build`: Builds an image from a Dockerfile.

- `docker rm`: Removes a container.
- `docker rmi`: Removes an image.
- `docker exec`: Runs a command in a running container.
- `docker inspect`: Returns low-level information on a container or image.
- `docker logs`: Shows the logs of a container.

These are just some of the basic commands available in Docker. There are many more commands available for managing containers, images, networks, volumes, and more. You can learn more about all the Docker commands by consulting the official Docker documentation.

Q5.

Docker Network commands:

- `docker network create`: creates a new network
- `docker network ls`: lists all available networks
- `docker network inspect`: displays detailed information about a network
- `docker network rm`: removes a network

Docker Volume commands:

- `docker volume create`: creates a new volume
- `docker volume ls`: lists all available volumes
- `docker volume inspect`: displays detailed information about a volume
- `docker volume rm`: removes a volume

Q6. Explain Multistage build on Docker

Multistage builds in Docker allow developers to create smaller and more efficient container images. In a traditional Docker build, each step creates a new layer in the container image. This can result in large and inefficient images, especially when dealing with compiled languages like Java or Go.

Multistage builds solve this problem by allowing developers to create multiple stages in a single Dockerfile. Each stage can be used to perform a specific task, such as compiling code or running tests. The resulting image only includes the files necessary for the final stage, which reduces the overall size of the container.

Here's an example of a multistage Dockerfile for a Go application:

```
# Build stage
FROM golang:1.14.2 AS build
WORKDIR /app
COPY . .
RUN go build -o myapp
# Final stage F
ROM alpine:3.12.0
COPY --from=build /app/myapp /usr/local/bin/myapp
CMD ["myapp"]
```

In this example, the first stage uses the `golang:1.14.2` base image and compiles the Go code. The resulting binary is saved to the `/app` directory.

The second stage uses the `alpine:3.12.0` base image and copies the compiled binary from the first stage into the `/usr/local/bin` directory. Finally, the `CMD` directive sets the default command to run when the container is started.

Using multistage builds can help to significantly reduce the size of Docker images and improve their efficiency.

AWS (Amazon Web Services)

Q1. AWS IAM Roles and responsibilities
AWS Identity and Access Management (IAM) is a service that enables you to manage access to AWS resources securely. IAM Roles are a fundamental part of IAM that allow you to delegate permissions to resources within your AWS account to another AWS service or AWS account. Here are some roles and responsibilities associated with IAM Roles in AWS:

1. Creation and management of IAM Roles: IAM Roles can be created by AWS account owners, administrators or developers. The IAM Role should have a clear purpose and an associated policy that grants the least privilege necessary to complete the task. IAM Roles can be managed by granting or revoking permissions and changing the role's policies as needed.
2. Delegation of permissions: IAM Roles allow you to delegate permissions to AWS services or users within your AWS account or another AWS account. These permissions are assigned through policies that specify the actions that can be performed and the resources that can be accessed.
3. Implementation of IAM Roles for security: IAM Roles are an important part of AWS security. IAM Roles help ensure that users or services are only granted the permissions necessary to perform their intended task. Properly implemented IAM Roles can reduce the risk of a security breach, and provide better visibility into who has access to what resources.
4. Implementation of IAM Roles for compliance: IAM Roles can help ensure compliance with industry regulations and security standards. By delegating permissions through IAM Roles, you can better control access to sensitive data and ensure that only authorized users have access.
5. Monitoring and auditing of IAM Roles: Monitoring and auditing IAM Roles is an important part of maintaining security and compliance. AWS provides tools to help you monitor IAM Roles, including AWS CloudTrail, which logs all IAM role-related events in your account.

Overall, IAM Roles play a critical role in securing your AWS environment and ensuring that permissions are delegated in a secure and compliant manner.

Q2. What is Load Balancer and Round robin method

A load balancer is a device or software that distributes incoming network traffic across a group of servers. It is used to balance the traffic load to ensure that no single server is overloaded.

Round-robin is a simple method used by load balancers to distribute incoming traffic equally among a group of servers. In this method, each server in the group is selected in turn to receive the next request. For example, if there are three servers in the group, the first request goes to server 1, the second request goes to server 2, and the third request goes to server 3, and the cycle repeats. This ensures that each server receives an equal number of requests.

While round-robin is a simple and effective method for load balancing, it may not be the most efficient method for all scenarios. Other load balancing algorithms, such as weighted round-robin, least connections, and IP hash, are available to suit different traffic distribution patterns and server capacities.

Q3. What is Target Groups and Listeners in Load Balancer

In AWS, a target group is a logical group of instances or IP addresses that receive traffic from a load balancer. Target groups route traffic to registered targets based on the routing algorithm defined in a listener rule.

A listener is a process that checks for connection requests, using the protocol and port that you configure, and forwards requests to one or more target groups. You can configure the listener to route requests based on the contents of the request (such as the URL), the origin of the request (such as the IP address of the client), or a combination of both.

Listeners can be configured with rules that evaluate the request path or host headers, and forward requests to the appropriate target group. Rules are evaluated in priority order, from the lowest number to the highest number, and the first rule that matches the request is applied.

In summary, listeners listen for incoming traffic and then route that traffic to a specific target group, which is responsible for handling the traffic.

Q4. About S3 bucket Uses

Amazon S3 (Simple Storage Service) is a highly scalable and durable object storage service offered by AWS. It is designed to store and retrieve any amount of data, at any time, from anywhere on the web.

S3 buckets are the primary storage containers in Amazon S3. They are used to store and organise data objects, such as files, images, videos, and documents, in the cloud. S3 buckets can be used for a variety of purposes, such as:

1. Storing and distributing static website content, including HTML pages, images, and scripts.
2. Storing and archiving data backups and disaster recovery data.
3. Storing and sharing media files, such as audio and video files.
4. Serving as a data lake for data analytics and big data processing.
5. Storing and distributing files for content delivery networks (CDNs).

S3 buckets provide a highly available, reliable, and secure way to store data in the cloud. They also support various access control mechanisms, such as bucket policies and access control lists (ACLs), to manage and control access to the data stored in them.

Q5.

Linux

Q1. What is Umask value in Linux?
In Linux, umask (user file creation mode mask) is a command that determines the default permission settings for newly created files and directories. The umask value is represented in octal format and specifies which permissions should be turned off for new files and directories. The default umask value is usually set to 022, which means that new files will have permissions of 644 (-rw-r--r--) and new directories will have permissions of 755 (drwxr-xr-x). This means that the owner has read and write permissions and group and other users have read-only permissions on the files, and the owner, group, and other users have read and execute permissions on the directories. The umask value can be changed using the umask command.

Q2. What is Swap Memory
Swap memory, also known as virtual memory, is a space on a computer's hard disk used to temporarily store data that the system's random access memory (RAM) cannot hold. When the RAM reaches its maximum capacity, the computer moves some of the data from the RAM to the hard disk space allocated for virtual memory. This frees up space in the RAM and allows the computer to continue running smoothly. However, since accessing data on the hard disk is much slower than accessing data in RAM, using virtual memory can cause a computer to slow down.

Ansible

Q1. Explain about Ansible Roles
In Ansible, Roles provide a framework for breaking up a complex playbook into smaller, reusable components. A role is essentially a collection of tasks, files, templates, and variables that can be easily shared and reused across different playbooks.

Roles are typically organized around specific functions or application stacks. For example, you might have a "webserver" role that installs and configures Apache or NGINX, or a "database" role that sets up and manages MySQL or PostgreSQL.

Roles are typically structured in a specific way, with the following directories:

- `tasks/` contains the main list of tasks to be executed by the role
- `handlers/` contains handlers, which are used to trigger actions based on certain events
- `files/` contains static files that need to be deployed to managed hosts
- `templates/` contains Jinja2 templates that can be used to generate configuration files dynamically
- `vars/` contains variables that are specific to the role
- `defaults/` contains default variables that can be overridden by other variables
- `meta/` contains metadata about the role, such as dependencies

Roles can be installed using Ansible Galaxy, which is a public repository of Ansible roles that can be easily installed and used in your own playbooks. Alternatively, you can create your own roles and share them with others.

Jenkins

Q1. Explain about Master and Slave concept on Jenkins
In Jenkins, the Master-Slave architecture is used to distribute the build and testing workload across multiple machines, allowing for efficient and parallel processing of jobs. The Master is the central Jenkins server that manages the build jobs and schedules builds on the available Slave nodes.

A Slave is a separate machine that is connected to the Master through the Jenkins SSH plugin or Java Web Start. The Slave can be a physical machine or a virtual machine, and it can be located on-premises or in the cloud.

The Slave machines are configured to accept requests from the Jenkins Master server, which delegates tasks to the Slave nodes for parallel processing. The Slaves are managed by the Master, and the Master can allocate and deallocate build tasks to the Slaves based on their availability and workload.

The Master-Slave architecture in Jenkins provides scalability and flexibility in the deployment of large-scale software projects and allows for efficient resource utilization. It also helps to distribute the workload and minimize the build time for large projects by running multiple builds in parallel on different Slaves

1. Jenkings Slownes : and while their jobs several of them face slowness issue were the pipelines get stucks and running for few hours. my side we had graffana also for monitoring purpose so, i will used to do i will use graphana while using jenkins prod , where i can see the memory of Jankins up or not. Like if I want to see the memory of Jankins Slave Occupies 100% every time on when job was running. There is used to fix increase the memory of the Slave on which the Job is running, JVM parameter XML
2. Authentication issues in different tools: With JEnkins we interegrate many things Like SonarQube Nexus etc, mostly on git i faced, in git we have to provide PAT (password Auth Token) and that we need to use as password.
3. CrashLoop BackOff Error in Kubernetes:
4. Jenkins Crashloop
5. Creashloop case 2
6. Image pullback off
7. Sonar Kube Error
8. Internal Server Error
9. Compilation Error
10. Not found Error
11. Docker multiple issues
12. Integration Issues