Question 1

How would you improve the current design to achieve better:

* High availability
* Resilience
* Performance
* Cost efficiency

You can list items based on priority ranked from high to low.

Answer :

we need to configure autoscaling on ECS Fargate to scale the number of workers based on data. This will help to achieve High availability, Resilience, performance, cost efficiency.

Question 2

The number of scan requests can increase/decrease randomly in a day and on most weekends the system receives almost no requests at all.

What strategy would you suggest to save cost while still maintaining the best possible performance and scan completion times?

Answer: we have our RabbitMQ queue depth metrics on CloudWatch, we need to configure autoscaling on ECS Fargate to scale the number of workers based on these data.

The values for autoscaling will depend on our setup. The best way to find accurate values is test and error. If we have peaks on our workload that triggers a lot of tasks in a short span of time, we'll want to scale to a big number of workers to be able to process those quickly. However, if we have a steady workload we'll want to scale more slowly.

First of all, we need to create a CloudWatch metric alarm. This alarm will trigger the autoscaling policies based on RabbitMQ values. For test environments that are barely used, it might be interesting to have 0 workers by default and create them dynamically as there are messages on the queue. This will add some delay to the processing of new tasks (Fargate takes a few minutes to deploy new tasks), but it might be worthwhile.

Question 3

In step 6, each job needs to mount the source code folder into every engine that needs to run. How would you store the source code and make sure that engines can run in a scalable way?

Answer :

We can use EFS.

The efs-provisioner allows you to mount EFS storage as PersistentVolumes in kubernetes. It consists of a container that has access to an AWS EFS resource. The container reads a configmap which contains the EFS filesystem ID, the AWS region and the name you want to use for your efs-provisioner.

The Amazon EFS CSI driver supports Amazon EFS access points, which are application-specific entry points into an Amazon EFS file system that make it easier to share a file system between multiple pods. Access points can enforce a user identity for all file system requests that are made through the access point, and enforce a root directory for each pod.

Question 4

Propose a high-level disaster recovery plan for the current architecture.

Answer: We can write a Terraform code for creating all infrastructure and integrate with Jenkins pipeline. We can use RDS multi-az, and Multi region, at any point of failure we can restore our services.

# **Part 2: Technical Challenge**

## **Schedule batch jobs**

We are running a cluster with multiple running nodes and have a list of batch jobs containing multiple jobs that need to run. Each job is defined with specific resource requests and resource limits.

Demonstrate how worker service creates the scan jobs by writing a script using kubernetes client (prefer Golang but not required) to schedule the batchJob1 jobs on a minimum number of nodes, based on the 3 scenarios provided below.

Note that the goal is to always schedule the jobs on the minimum number of nodes. This will reduce the number of times the source is downloaded, which lead to:

* Reducing the preparation time for scan jobs
* Reducing the stress on the source code repository service
* Increasing resource utilization of the nodes

## Sequential Kubernetes Job: In the case that you want to run batches of jobs in sequence rather than have one very long job. This is especially good if you are worried that jobs may fail and need to restart, if a job is consuming a lot of memory/resources and needs to be limited, or if you just don’t want background jobs running too long.

Python script for running sequential job :   
from k8s\_scheduler import Scheduler

from k8s\_scheduler.LogHandler import LogHandler

scheduler = Scheduler(host="mongo\_db\_host",

username="user\_name",

password="password",

db\_name="test\_schedule")

# Create schedules in the mongodb collection

job\_id\_1 = scheduler.create\_schedule(schedule=test\_job\_1)

job\_id\_2 = scheduler.create\_schedule(schedule=test\_job\_2)

job\_id\_3 = scheduler.create\_schedule(schedule=test\_job\_3)

# Start Scheduler application

scheduler.start\_scheduler()