Q.1. Describe a high level design for a Distributed Code Deployment System.

Part 1: Building

Let's say a developer wants to deploy code corresponding to a commit SHA.What seems to be a right choice is a Queuing mechanism to queue all the requests. And worker pool building the code commits in FIFO manner.Workers will then store the binary in blob store (can be S3 or GCS).

Since we want to persist the history of jobs. We can think of a SQL table for storing the job entries (this can represent our queue).We define our status enum as { QUEUED, RUNNING, FAILED, COMPLETED, CANCELLED}.How to work this as a queue ?We can select the jobs where status = QUEUED and has the oldest created_at timestamp.Hence the index on status and created_at will improve performance.The fact that we have SQL database, we

have ACID transactions. This enables our X number of workers to query and update the jobs as every run is a transaction and hence concurrency safe.

What if our worker crashes while building code ?

We can tackle it by adding a column last_heartbeat in our jobs_queue table. Whenever a worker picks up a job and is running it. It will continuously send heartbeat signals to the job table. Let's say a build takes around 15 mins, the workers will send this heartbeat every 3 mins.

Now we introduce a new monitoring service that will monitor the jobs_queue table for all the "RUNNING" jobs. If we find a job's heartbeat not updated for 2*heartbeat duration i.e. 6 mins in our case, then we rollback that job's status to "QUEUED"

How many workers are actually needed ?

Assuming 5000 builds/day. And 15 mins/build → ~ 100 builds/day that a single worker will perform. Hence we can say that 5000/100 = 50 workers are required on average. We can horizontally scale the number of workers on peak hours and vice versa.

Another key thing here is to update the job's status only when the binary is successfully stored in the blob storage. As discussed we have multiple regions globally where these binaries are going to be deployed. Assuming 5 regions where we have clusters of application servers, we can have a regional blob store at each of these regions and each regional blob store serves the binary to application servers of its region.

What if we want the developers to be able to deploy the build only if the binary is replicated to all the regions ?

To tackle this requirement we can have a really simple service which polls the master blob store for any new binary and tracks the status of that binary by polling to all the regional blob stores.The build is only deployable if the replication_status in the table is “COMPLETED”.

Part 2: Deploying

We assumed earlier that the build takes 15 mins and let’s say replication takes another 5 mins. So for us to meet the 30 min deadline for entire deployment, we are left with 10 mins. For 100K machines to download a 10 GB file from the blob store over network seems unreasonable. Hence we can create a Peer-to-Peer network. All the machines in a region are part of a Peer to Peer network, this will enable them to download multiple such binaries really fast.

What exactly happens when a developer presses the “Deploy” build “Build_1” ?

We can take advantage of our clustered architecture to introduce a Key-value store (like zoo-keeper, consul) at every regional cluster. And a master Key-value store which is actually updated when the developers presses the button. The idea is to have a config like -

```
{

 build_number: "build_0"

}
```

Whenever the build number in the KVS is updated the machines downloads the new binary and deploys it.

The regional KVS polls the master KVS for any change in the build_number and whenever the application servers are in steady state they poll their regional KVS for any change in the build_number.

Let’s take a look at our final design: