**We would like you to design an e-commerce platform to support a service like the Amazon website where customers can buy items and sellers can maintain inventory and the platform owner can maintain a marketplace ecosystem. This platform that you will build will run at a similar scale as Amazon. Focus on the backend platform infrastructure (databases, operating systems, networking, automation, redundancy, monitoring etc) that will support the e-commerce applications, but not necessarily the front-end applications themselves. You have flexibility to choose technology building blocks that you are most familiar with, and please justify or explain why you made particular technical and architectural choices.**

Important points to cover:

* **How will you handle the loss of a region?**

To prevent risk of going out of business when we lose a region, high availability is not enough we need Disaster Recovery. In case of HA, we ensure there exists a fallback mechanism for our services i.e. having a blue-green setup means we can do failover routing on Route53 for example. The service that runs in HA is handled by hosts running in different availability zones but in the same geographical region. This approach, however, does not guarantee that our business will be up and running in case the entire region goes down. DR will enable us to recover from a different region that’s separated from our main region. The DR implementation can Active/Passive model, meaning that we always have minimum critical services running in different regions, but a major part of the infrastructure is launched and restored when required, while the day-to-day business is run from our usual region e.g. eu-west

* **How will you factor in growth and scale?**

By using Kubernetes, we ensure that the system or service is designed to scale based on user demand, so if a microservice on the application is getting a lot of requests e.g. more products are being bought, then we need to scale that service by scaling out during peak times and scaling down during less busier times. That means spinning up more replicas, Docker containers and clusters (EC2 nodes running on EKS) to handle the load.

* **Security, Reliability, Speed are all important factors too.**

1. **Speed**

The speed of the service is crucial and that means having our backend and front end within the same region to reduce latency for users. Caching mechanisms can also be used to ensure that content or information does not need to travel a greater distance for user consumption. By also using microservices, we ensure that the applications are faster since different components are decoupled. Also using tools like Terraform and Kubernetes means we can spin up different infrastructure elements in minutes during failures.

1. **Security**

Security on the cloud is equally as important as running on data centres. A lot of risk can be mitigated by having hybrid clouds e.g. customer data can be stored on data centres or cloud providers we can easily manage and have complete visibility of using regions closer to headquarters. Hybrid clouds let you choose where to place workloads and data based on compliance, audit, policy, or security requirements—protecting particularly sensitive workloads on a private cloud, while operating less-sensitive workloads in the public cloud. We can also mitigate security risks by using trusted software e.g. Puppet for config management ensuring users have correct access rights through automation etc.. , managing lifecycles

1. **Reliability**

By using Cloud based technologies, we ensure that our application is resilient and reliable 24 / 7. Much of this is due to the fact that cloud services like RDS or EC2, they can be deployed across different Availability Zones ensure that if 1 server / cluster fails in our AZ, we can also have an application available in a different zone. Some other factors to consider may include: -

1. Data replication between regions must be **fast and reliable**
2. Services should not have local state — they must be **stateless**, and state should be shared between regions.
3. Synchronous cross-regional calls should be avoided when possible. Applications should **use regional resources**
4. For distributed data stores, asynchronous replication decouples the primary node from its replicas at the expense of introducing replication lag or latency.
5. Use local resources and avoid multi-cross regional calls for back-end and front end e.g. resources stored on Amazon S3 or RDS can be replicated in multiple regions to allow for access from any region
6. Also Route 53 allows cross regions DNS registration meaning users in different regions can have access to resources local or closer to them by using routing policies
7. As mentioned already, Kubernetes ensures that containers that are not working are removed from the cluster and replaces with new ones

* **You can design this architecture on the cloud or on-prem.**

See Architecture Diagram attached on GitHub repo using AWS as the cloud provider

* **The technology you choose will be important, so if you use a particular database make sure you can explain why it’s a good fit**

Tech stack can include

* Angular and Node JS for front-end
* Java Spring Boot for Backend functionality
* RabbitMQ for Messaging between the different microservices
* Kubernetes and Docker on EKS for containerisation and orchestration
* PostgreSQL for the Databases and Mongo for No-SQL?
* Spinnaker for cross cluster deployment
* Jenkins for CI/CD and maybe Travis as well?