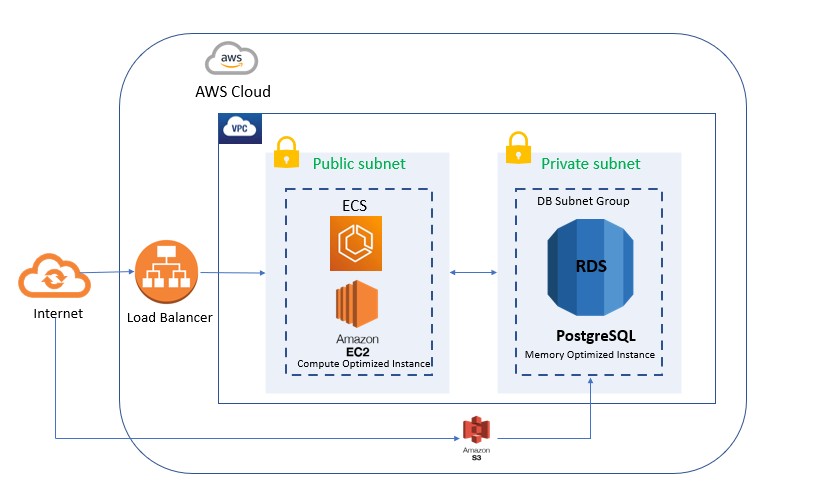
***Scenario:*** *Imagine that for providing data to fuel this service, you need to receive and insert big batches of new prices, ranging within tens of thousands of items, conforming to a similar format. Each batch of items needs to be processed together, either all items go in, or none of them do.*

*Both the incoming data updates and requests for data can be highly sporadic - there might be large periods without much activity, followed by periods of heavy activity.*

*High availability is a strict requirement from the customers.*

1. **How would you design the system?**

The detail design diagram of such a system is shown below.



The application contains an API part and a database part. The API can be launched in an ECS service where the cluster can be run on EC2 instances managed by us. The postgres database can be maintained in AWS RDS.

Since our service include heavy workloads, we can use Memory Optimized Instances for our database. Instead of installing postgres in an EC2 instance, we can use a separate RDS instance for the PostgreSQL database. The RDS is a better way to manage database as it is highly available, and we can scale the database up and down depending on our workloads. The database inserts can be done using an S3 bucket in between. The data to be imported can be added to the s3 bucket and from there it can be imported to RDS.

Our API can be run as containers in an ECS cluster. Since we require highly available service, we can go for compute optimized EC2 instances(C5) for our API as it will guarantee low latency.

If the web service is expected to go through a period of inactivity, on demand instances would be the best option.

1. **How would you set up monitoring to identify bottlenecks as the load grows?**

Cloudwatch is the best option to monitor AWS resources. Cloudwatch collects and tracks metrics. We can create dashboards to display metrics of our instance. We can also create alarms that watch the metrics and send notifications when any of the threshold is breached. We can monitor the CPU usage and disk reads and writes of our Amazon EC2 instances and then use that data to determine whether we should launch additional instances to handle increased load. We. can also use this data to stop under-used instances to save money. This way we can identify any bottlenecks in our service.

We can also watch our RDS instance the same way. By default, RDS sends metrics to CloudWatch. We can also monitor custom metrics. We can create an alarm for threshold breach.

We can even configure SNS to send email notifications for threshold breach.

1. **How can those bottlenecks be addressed in the future?**

Autoscaling feature of AWS comes in handy in this situation. This feature provides automatic resource management based on the server’s load. The two main components involved are AMIs and load balancer. First, we need to create an AMI for our current server. It is a template of our current configuration and it will contain all the settings of our current system. Now when the traffic starts increasing, the AWS autoscaling would launch another instance with same configuration using the AMI we created. The load balancer would take care of the traffic. It would divide the traffic equally among the instances.

RDS monitors database storage utilization, and when current usage is close to the provisioned size, it scales up storage capacity available to the database instance.

**Additional questions: -**

***Scenario:* Code updates need to be pushed out frequently. This needs to be done without the risk of stopping a data update already being processed, nor a data response being lost.**

This situation arises when the minimum number of instances available is 1. Thus, the instance would be unavailable till the deployment is complete. This will lead to unavailability of the webservice and hence issues with data responses and requests.

This scenario can be avoided by using minimum of 2 or more instances available for the service. Also, during deployments using code deploy, update only one instance at a time. The load balancer will route the traffic between the remaining available instances. This will enable zero downtime as any one of the instances will be available all the time.

Also, since postgres runs as separate entity in RDS, it is independent of the code changes happening in the API server.