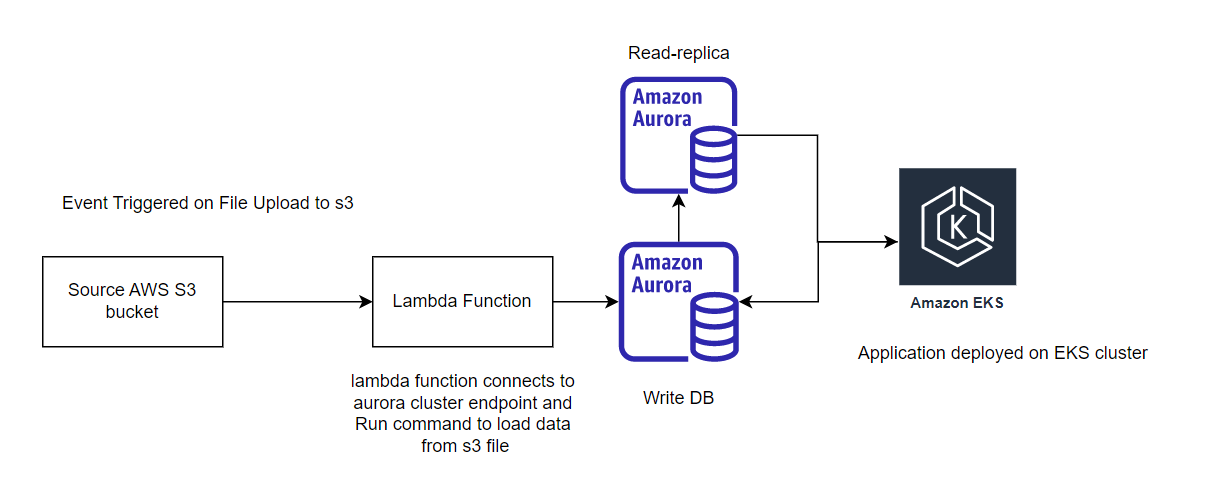
**Load data through s3 bucket:**

If we choose to load data externally from sources like s3 buckets below mentioned approach can be used. This approach has an advantage of not having any dependency on application and new changes to application can be pushed very frequently without affecting the current data loading process.

Prerequisites to use this approach:

1. Create IAM policy to allow access to s3 bucket.
2. Create IAM role and assign this policy
3. Assign IAM role to PostgreSQL aurora cluster.
4. Allow Load data from s3 command execution permission to database user

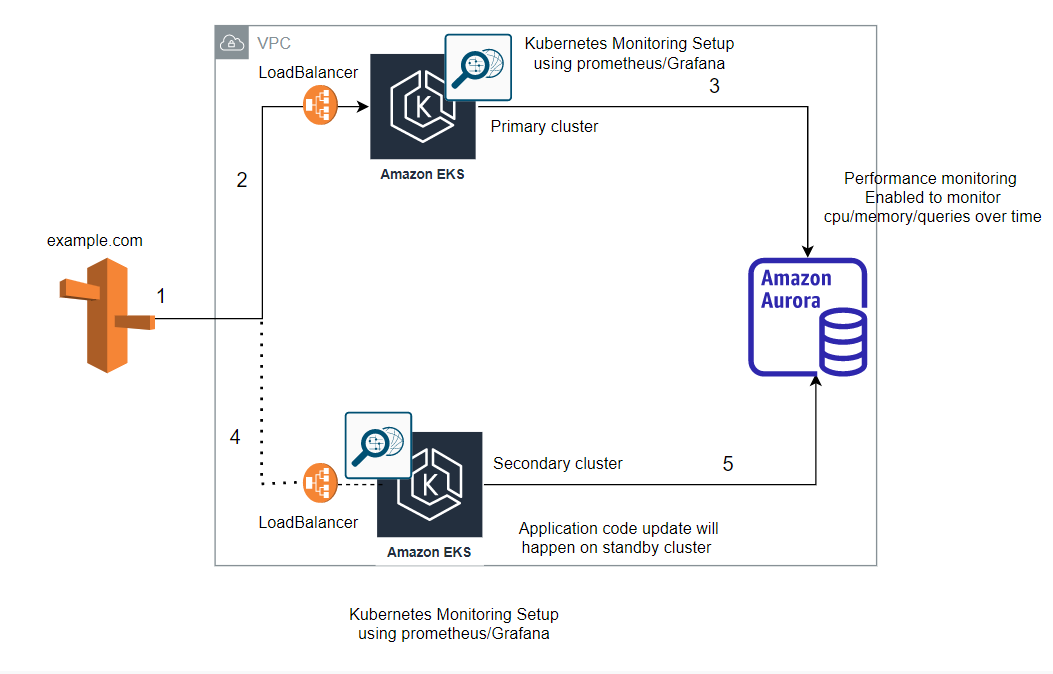


1. File will be uploaded to s3 bucket which will generate an event to trigger lambda function.

2. Lambda function will connect to PostgreSQL aurora endpoint and invoke PostgreSQL inbuilt command : **load data from S3 <s3 bucket location>**

**Why Aurora cluster:** This is fully managed database cluster which is auto scale able to handle variable load as compared to traditional RDS instance.

**Application Deployment Infra Architecture:**



Application will be deployed on AWS Elastic Kubernetes Service in the form of pods. PostgreSQL Aurora will be used with primary instance for write operations along with read replicas.  Aurora uses shared storage for writer and readers. As a result, all Aurora replicas are synced with the writer instance with minimal replica lag.

Prometheus will be used to fetch the EKS infra stats and will be visualized using Grafana. We can also use CloudWatch for EKS cluster nodes’ stats monitoring.

Prometheus can fetch the ingress stats like latency, number of requests received/processed, Success rate etc. which can be helpful to find out any potential issues with infra. Application logs can be fetched and send to Elasticsearch using filebeat for any application-level debugging.

Continuous Monitoring of both Application Infra as well as Database will enable us to find out potential issues/bottlenecks in our system. We can take steps accordingly to resolve those e.g in case of increase in response time with load, we can step up horizontal pods scaling to scale number of application replicas based on stats like CPU and memory and same can also be done to scale number of replicas of aurora cluster.

**Load data through Application:**

When we are loading bulk data through application, below discussed scenarios/solutions can be opted.

**1. The batch updates have started to become very large, but the requirements for their processing time are strict:** As we are using AWS aurora for backend DB, we can enable autoscaling based on different predefined or custom stats e.g CPU utilization. Moreover, we can enable performance insights of database to know more if there will any issues during processing of data.  
Auto commit needs to be disabled on database for particular session to either load all records or none.

**2. 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 can be achieved by having blue/green deployment method, where we will have two identical Kubernetes clusters.

One will serve live traffic and other will be used to deploy the changes and do testing. If everything goes fine traffic can be switched to other cluster either in batches or fully at once (This should be done after data batch loading)

**3. For development and staging purposes, you need to start up several scaled-down versions of the system:**

New EKS cluster can be created using terraform with lower spec EC2 nodes in it along with a smaller number of pod replicas.

**Which parts of the system are the bottlenecks or problems that might make it incompatible with the new requirements? :**

**1.** 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 can be a bottleneck with our infra design. We must wait to switch to the cluster with new application code until data processing finish.

To avoid using another cluster and switching to that one, we can think of segregating read/write traffic and direct each to different set of pods. By using this approach, we can update the pods with new code which are serving read traffic while data processing is being going on the other set of pods.