Deployment Steps for the Flask application and MongoDB on a Minikube Kubernetes cluster:

Start Minikube:

**minikube start**

Apply the Kubernetes configurations:

**kubectl apply -f mongodb-statefulet.yaml**

**kubectl apply -f mongodb-service.yaml**

**kubectl apply -f flask-deployment.yaml**

**kubectl apply -f flask-service.yaml**

**kubectl apply -f hpa.yaml**

Access the Flask application:

Get the NodePort assigned to the Flask service

**kubectl get svc flask-service**

Access the application using **http://<minikube-ip>:<node-port>**

DNS Resolution Explanation:

In Kubernetes, DNS resolution allows pods to communicate using service names instead of IP addresses. Each service is assigned a DNS name (e.g., mongodb-service) that resolves to the service's ClusterIP. This simplifies inter-pod communication, as the Flask application can connect to MongoDB using the service name directly in the connection string.

Resource Management Explanation:

Resource requests and limits in Kubernetes help ensure that applications have the necessary resources to run efficiently while preventing any single application from monopolizing cluster resources.

Requests specify the minimum amount of resources required (e.g., 0.2 CPU, 250Mi memory).

Limits define the maximum resources an application can consume (e.g., 0.5 CPU, 500Mi memory).

These configurations help maintain stability and performance across the cluster.

Design Choices:

StatefulSet for MongoDB: Chosen for its ability to manage stateful applications with persistent storage.

Horizontal Pod Autoscaler: Implemented to automatically scale the Flask application based on CPU usage, ensuring responsiveness under load.

NodePort Service for Flask: Allows external access to the application while keeping MongoDB internal.

Testing Scenarios:

To test the autoscaling and database interactions:

Simulate High Traffic: Using a tool Apache Benchmark to send a high volume of requests to the Flask application.

Example command using ab:

**ab -n 1000 -c 10 http://<minikube-ip>:<node-port>/**

Monitor HPA: Use the following command to monitor the HPA:

**kubectl get hpa**

Check Logs:

Monitor the logs of the Flask application and MongoDB to ensure data is being processed correctly.

Analyze Test Results:

After running the ab command, you should see output similar to the following:

Concurrency Level: 10

Time taken for tests: 3.108 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 154000 bytes

HTML transferred: 20000 bytes

Requests per second: 322.02 [#/sec] (mean)

Time per request: 31.076 [ms] (mean)

Time per request: 3.108 [ms] (mean, across all concurrent requests)

Transfer rate: 48.44 [Kbytes/sec] received

Connection Times (ms):  
 min mean[+/-sd] median max

Connect: 0 0 0.3 0 3

Processing: 10 31 15.6 23 67

Waiting: 9 30 15.5 22 66

Total: 10 31 15.6 23 67

These results indicate that Flask application can handle around 322 requests per second with a mean response time of 30.992 ms when subjected to a concurrency level of 10 requests. The percentile information shows the distribution of response times, with 50% of requests completing in 23 ms and 90% in 67 ms.

The output also includes some connection time breakdowns, showing the time spent in the different stages of the HTTP request-response cycle.

Overall, these results provide a performance snapshot of your Flask application under the specified load conditions. You can use this information to assess the application's scalability and identify potential bottlenecks.