SCALABLE ARCHITECTURE

The open food truck finder application has been designed to provide the user with information on the name and location of trucks open in the city of San Francisco at a specific time. The prototype uses the Socrata API to obtain the data and displays the results in batches of size 10.

While the program is sufficient for a single user, scaling it requires additional components.

1. A web interface can be designed for the user to request and view the data through a **Web Server** that sits behind a **gateway**.
2. The command-line program can be **configured as a service** using frameworks such as Flask/Django in Python. This would provide an endpoint that can be called from the Web Server.
3. Scaling of Web Servers and the API services can be achieved using **horizontal scaling** functionality provided by services such as Heroku and AWS (EC2, Autoscaling). The scaling would be triggered based on pre-defined load metrics.
4. A **load balancer** placed before the API services would distribute the requests so that a single server is not overloaded, providing high **availability**.
5. Based on the data, it can be observed that the information from the API is not refreshed frequently. We can store the information state in a **cache** which would reduce the number of API calls made. Redis/Memcache can be used to temporarily store response data. Subsequent requests can first check the cache to provide faster responses. The cache data can be periodically cleared or refreshed using another service. **Consistency** can be maximized using efficient caching strategies.
6. Scaling also brings challenges with **logging and monitoring**, which can be resolved using services such as CloudWatch, or with stacks such as ELK and EFK.

Additionally, containerization and orchestration can be used to reduce scalability and availability latencies.