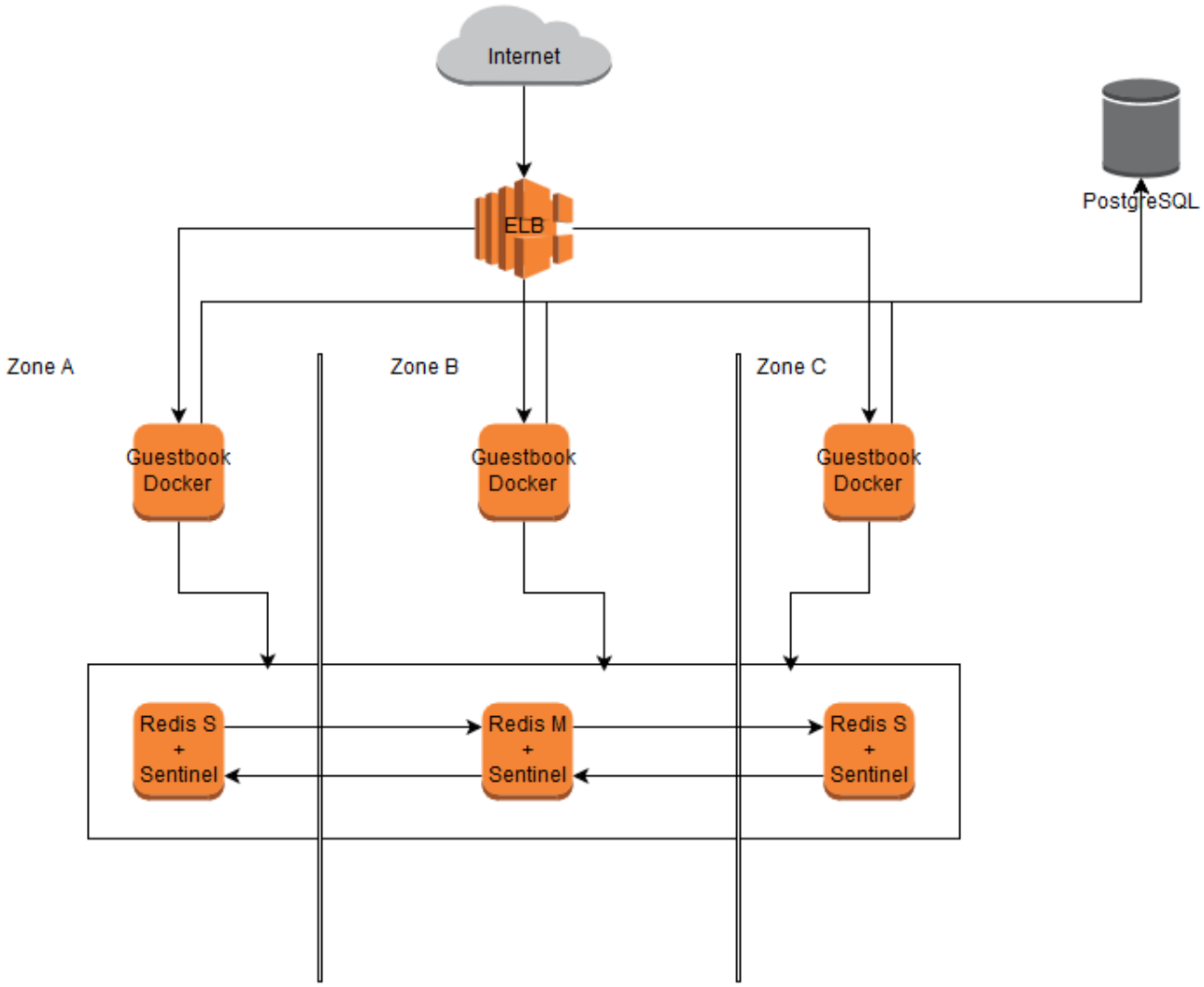
Guestbook Architecture.

Assumptions.

- Guestbook application is stateless (or more accurately its state is stored externally) and can have multiple instances running in parallel.
- PostgreSQL DB configuration is assumed to be outside of the scope of this document.
- Redis DB stores data that needs to be shared by all Guestbook instances in a deployment (the guest counter).

Diagram.



Explanation.

The architecture proposed in this document is an (almost) minimal one that is still highly available. It would also scale well with some caveats (see Redis details).

Guestbook

The guestbook application does not store any state internally which makes it a perfect candidate for containerizing. This would be beneficial from the deployment point of view (application ships with all dependencies) and would also make it easy to scale flexibly with desired degree of automation. Architecture proposed here uses Salt configuration management for container deployment which would be sufficient if dynamic scaling was not required. Otherwise container orchestration (e.g. ECS) could be used instead.

ELB would provide load balancing requests across running instances and the instances with Guestbook containers would be placed in multiple zones to increase availability.

Redis

Redis database would be deployed in a master/slave configuration. Since three instances are required for Sentinel cluster to work properly and it would share instances with Redis DB we can assume to have one Redis master and at least two slaves.

Redis Sentinel would monitor Redis hosts and automatically promote a slave in case of a master's failure. As with Guestbook the Redis instances would be placed in different availability zones to make the setup more resilient.

This architecture would be possible to scale for read-only operations by increasing the number of slaves (probably not very useful here). Also scaling for writes would normally require partitioning the data which would not really be possible in case of Guestbook app (only one key to store).

Using Redis Sentinel would require a small modification in the application code (Redis.new call).

Configuration

Service provisioning and configuration is handled by Saltstack. The provided minimal version would be suitable for development use and could be extended to implement the production architecture proposed above.

Docker, postgres and redis formulas are unmodified versions from github.com/saltstack-formulas (Postgres has been included to make testing easier). Sensitive information (e.g. DB password) is passed via salt pillars which is a recommended and secure way.

Guestbook application has been dockerized. The provided guestbook salt formula takes care of building the image and deploying container. In production the image would most likely be pulled from a Docker registry which would shorten the deployment time.

Vagrant instructions.

To run the provided minimal version it should be sufficient to execute 'vagrant up' in the top directory where the Vagrantfile is located. It is configured to forward localhost:9292 to the application. In case of any issues with built-in port forwarding ssh can also be used (on a different port):

```
$ vagrant ssh -- -L localhost:8080:localhost:9292
```