1 Service (Ingress)

2 Service (LB)

3 Service (LB)

4 dashboard
- Dashboard (nginx/react)
- Dashboard (nginx/react)

5 backend
- puffy (expressjs/MVC)
- puffy (expressjs/MVC)
- puffy (MVC)

6 Service (Ingress w/CORS)

7 Service (LB)

8 Service (LB)

9 cache
- Redis
- Redis

10 database
- MySQL
- MySQL

Design

Explanation about this design on k8s

1. There's two main paths to catch a Reactjs section of the app and a Expressjs section. The ingress is necessary for routing the request to the correct section.

2, 3, 7, 8. A simple load balancer to balance workload to its replicas.

4. The dashboard's deployment, if theres a change on the dashboard, only this is affected. It uses nginx to cache the build of the react app.

5. The MVC app's deployment, the og design of the app is in the MVC model, so theres not a pure front and back part (sorry...). Also, here's the API for the dashboard.

6. An Ingress service that has ALL the configurations to enable CORS for the API that uses the dashboard. As a design decision, the CORS part won't be implemented inside the expressjs app, so there should be another device that control this request type.

9. A Redis instance to manage session and cache common queries, it should be managed separated in a high traffic app.

10. A MySQL instance to manage all the app's data (users, products, orders, etc.).