Part 1: Health Endpoint

Problem:

Health monitoring refers to monitoring an application regularly to make sure the service(s) are running as intended, while managing any faults that could or have occurred. Remote services are easier to manage as the service environment is directly manipulable by the owners, however cloud based solutions are sometimes more difficult to monitor. Cloud hosting can be affected by offloaded host environment, network latency, performance and availability of resources, storage options, and bandwidth between them. As there are many behind the scene factors that play a part, there is a need for regular monitoring.

Solution:

Use endpoints for health monitoring, the application will perform necessary checks and return a indication of the status to these endpoints. Tests should cover all service instances that are exposed to a customer, as well as back end tasks as well.

Two Factors:

- The health check is performed in response to a request
- Analysis of the results after the health check

Similar to response codes provided from websites, where numbers have a standardized meaning. Response time/latency is evaluated by the monitoring tool. Response times can be evaluation by measuring the delay from the request to response, and possibly removing a slight delay threshold.

Typical checks:

- Response code validation (200=ok, 404=bad)
- Content validation, even if a error check code returns positive, data on a page must be checked
- Response time
- Resource usage (within or above accepted tolerance)
- SSL certificate

These checks can be done from different hosts, to compare the results.

Requirements:

- Active service has constant uptime
- Application required multiple services that work together
- Application utilized multiple frameworks to communication processes
- Multi-tier system, where one service relies on another. Points of failure in one service can ruin the whole software

Discussion:

Summarize the problem, the solution, and the requirements for the pattern given in part 1.

The problem for part 1 was observability, and how can we monitor our applications performance and health regularly. By invoking request to endpoints related to different components of a system, we can analyze response codes and figure the corresponding errors if they exist.

Which of these requirements can be achieved by the procedures shown in parts 2 and 3?

Part 2 utilizes circuit breaker pattern that will help create a recovery time in the system in case of an error code. The lab mimicked a fake error code sent to an online service, and during this time, the circuit breaker would delay request. Once a new request was sent after the service was back up, we can see that the connection was being confirmed from host to server, and the status was notified as 200 (OK).

Part 3 demonstrated how return statements can be used to monitor our application. By invoking method calls on our application based on the endpoint routing, we passed JSON information to the endpoint, and monitored if our output response was as it should be. In case 1 we should have been returned the same JSON object, however in part 2 we noticed that an empty key value pair still returned a transparent color.

Design:

1. Why are Kubernetes persistent volumes so important?

They provide a way for data to persists when conatainers are managed, whether they are deleted or restarted. Some data can be temporary during virtualization, but to maintain them throughout we need a way to remember them.

2. How to implement this?

Automatically scaling a cluster up and down spending on the demand/resources being used. Horizontal Pod Autoscaler. Pods will get deployed in response to a growing load.

3. Provide an example in which persistent volumes are needed?

With kubernetes, applications that are configured are usually stateless, but persistent volumes are needed when we are dealing with stateful applications. Example would be a shopping cart application