Kelly Plummer

10-25-14

Write up for technology and design decisions for the Developer Technical Team Evaluation.

**Technology**

For handling REST requests the Jersey framework was chosen as it implements the Java API for RESTful Web Services (JAX-RS). Jersey was chosen over others for several reasons. Those include broad adoption, adequate documentation and active development. The persistence layer is a simple mock that is backed by a Hashmap. Effort was put in to integrating the Cassandra NoSQL database but time did not allow for completion. The Hystrix library is used for the circuit breaker implementation.

**Design**

The design of the inventory microservice is separated by concerns, querying the persistence layer and handling REST requests. The *ProductDao* class, used for the persistence interface, provides a static factory method, *getProductDao,* which returns a concrete *ProductDao* implementation. This approach was chosen because it would allow changing the underlying persistence fairly easily. The *ProductDao* also provides a simplified query interface. The interface has 5 methods: *index, create, get, update* and *delete*. This provides basic CRUD functionality to the inventory microservice.

The *ProductService* class provides handling of REST requests for the *Product* data. Like the *ProductDao* interface, *ProductService* exposes 5 endpoints. This allows the user to create, update, delete and edit Product data. I an addition to the RESTful endpoint and persistence layer the Circuit Breaker pattern was required to ensure the inventory service would not fail due to unavailable or underperforming dependencies.

Hystrix was a requirement for implementing the circuit breaker functionality. The inventory service uses the *HystrixCommand* class to manage business logic for the REST methods in the *ProductService.* For each of the methods there is a corresponding *HystrixCommand*. Each command implements the abstract *run* method with the logic required to perform each task.

**Deployment**

In an effort to increase automation, the application should be deployed using Jenkins, or another CI server. Jenkins should monitor for changes in the source repository. If a change is detected a Jenkins job should begin. The first step in the job should run unit, integration and functional tests. If the tests are successful a war artifact of the service should be created by the Jenkins job. The artifact can either be stored in an artifact repository, i.e. Nexus or Artifactory or simply have Jenkins archive it. The final step in the Jenkins job should be to deploy the new artifact to either an existing AWS instance or AWS Beanstalk. There are plugins to Jenkins for managing the final deploy step.