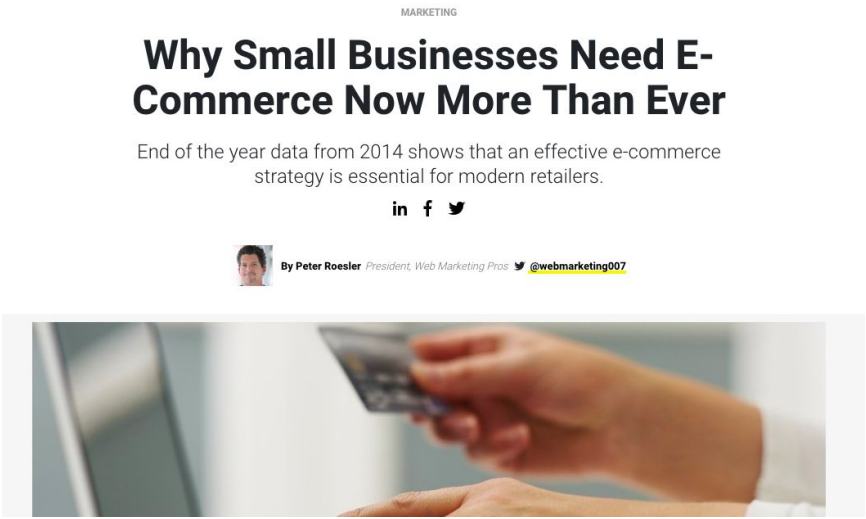
AWS Distributed (Book)store

Team Members: Harrison Ratcliffe, Julia Goyco, Stephanie Schoch, Toby Duncan

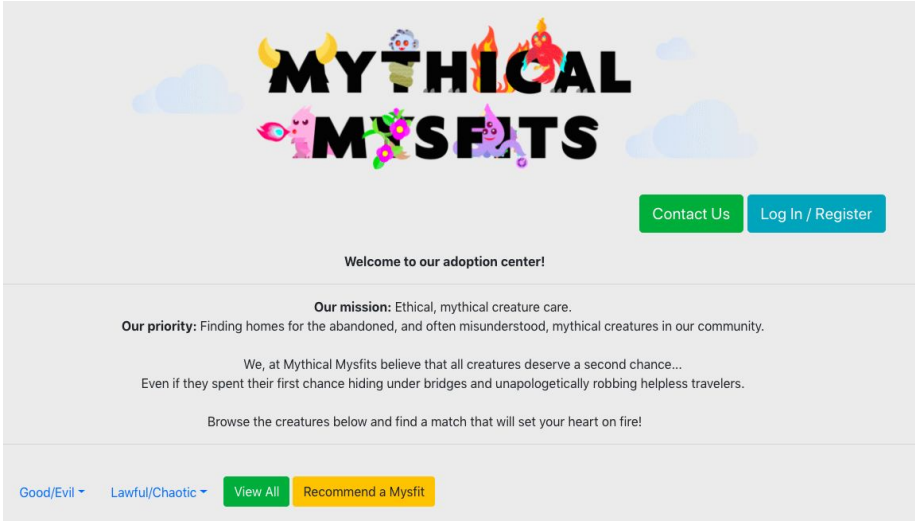
Motivation

- **Question:** What can we realistically build in this timeframe that addresses a need?

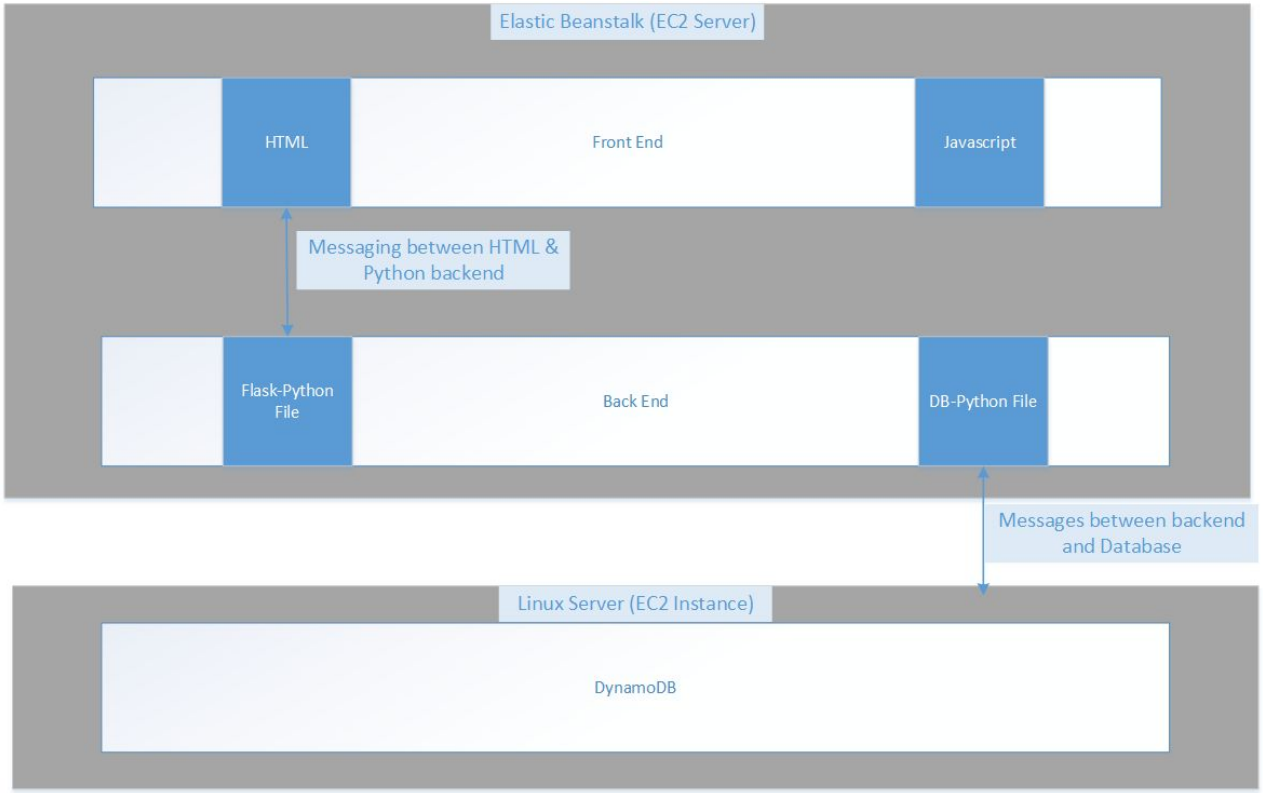
E-Commerce:



Examples on AWS:



Final Product (System Overview)



Product Structure

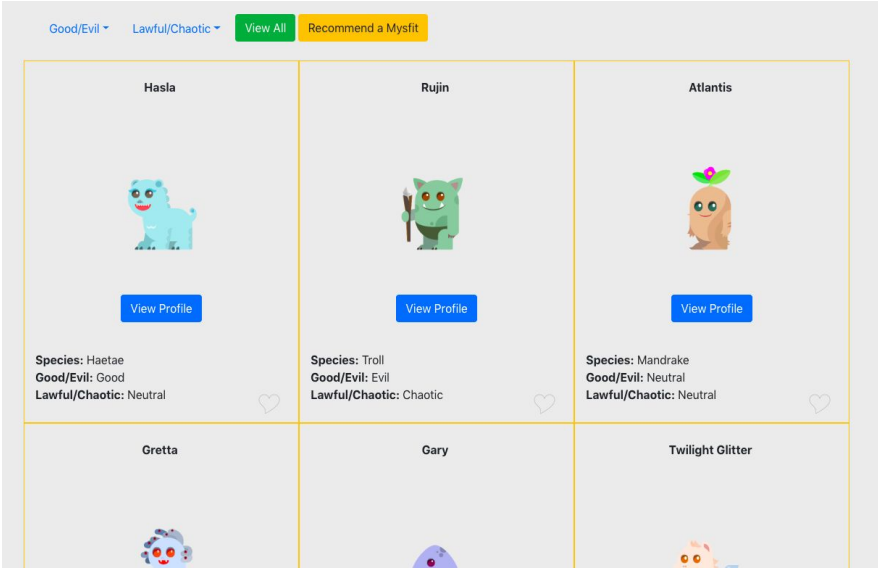
- Front End was made up of HTML files and Javascript.
- Used JQuery to retrieve information from HTML documents and send them to backend
- Backend was made up of 2 different Python file: one for interacting with HTML files and one for the database
- Used Flask as our HTML framework
  - Allowed for linking paths to HTML documents and sending data from backend to HTML files
- Used Paramiko to communicate between backend and database
  - SSH into instance server with database and ran commands from terminal
- Simulated bookstore by having two main pages - order page and admin page
  - Admin page allowed for deleting, updating, and adding items to bookstore
  - Order page allowed for ordering items
  - Had a admin login page to verify that user was admin
  - Bookstore stock was printed to order and admin page and updates with changes

Original Plan/Changes

- The original plan was to create an online bookstore with multiple databases all on AWS
- For the database, we were going to use DynamoDB on AWS
- Unfortunately, our group ran into problems with permission with AWS
    - As result, were not able to access resources to deploy multiple databases on AWS
- We changed things by downloading a local copy of DynamoDB onto an EC2 instance running Linux
    - Used ssh to connect to EC2 instance and then used terminal command to perform database operations
- We then had another EC2 instance running Linux hosting the web application and had the two communicating between each other
    - Ended up using Elastic Beanstalk to deploy application to other EC2 instance and setup URL
- The main difference was multiple databases only became one database
- Still created a very basic distributed system that had one EC2 instance talking to another EC2 instance (database)

Similar Systems: Structure

- AWS Learning Modules
  - Based our database off the one provided by “Create and Manage a Nonrelational Database with Amazon DynamoDB”
  - Took inspiration from “Build a Modern Web Application: Deploy a web application, connect to a database, and analyze user behavior” for our application”
- Our system is much less complex than similar systems
  - Single NoSQL database
  - Application only allows a few basic functions



Shows “Mythical Mysfits” store interface (linked to database)

Similar Systems: Performance Evaluation

- zData: Pivotal Gemfire:
  - “In-memory, distributed data store that provides a SQL interface to table data”
  - Designed high availability, high scalability, and low-latency
  - **Performance Evaluation Techniques:** Yahoo! Cloud Serving Benchmark (YCSB)
    - Tested scaling from cluster of 4 nodes to cluster of 8 nodes
  - **Performance Evaluation Metrics:** Throughput, Latency
- Cassandra:
  - NoSQL database system which is designed for scalability across multiple data centers with unstructured data
  - **Performance Evaluation Techniques:** Yahoo! Cloud Serving Benchmark (YCSB)
    - Tested scaling from cluster of 3 nodes to cluster of 6 nodes of instances of Cassandra
  - **Performance Evaluation Metrics:** Throughput, CPU Utilization
- Metrics For Our System:
  - Throughput (in form of network usage), CPU utilization
  - **Note:** Difference between throughput and latency in our system would be minimal as it is extremely basic and not as data-intensive as similar systems.

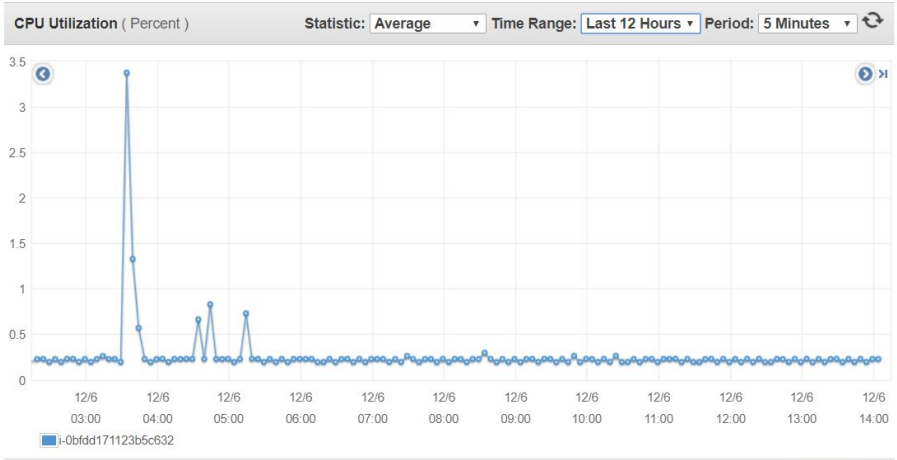
Performance Evaluation

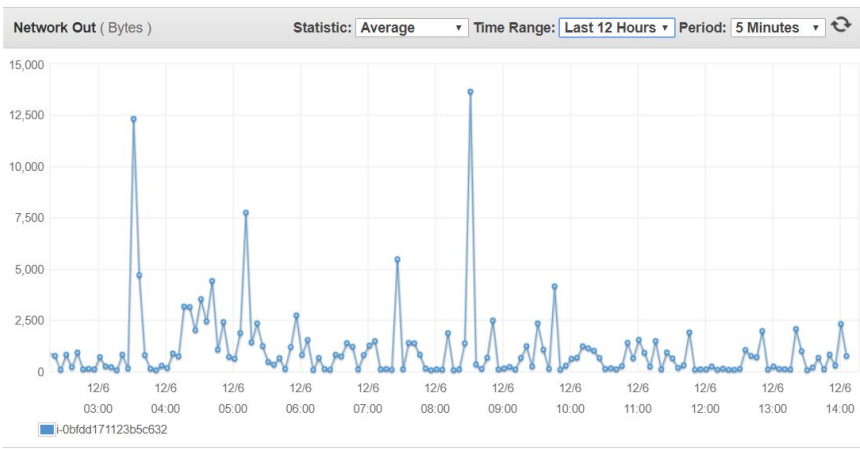
Evaluation Methods

- Used AWS CloudWatch
- Database metrics:
  - CPU usage - percentage of CPU usage that is currently in use on the instance is being tracked
  - Network-Out - volume of outgoing traffic in each instance
- Collected over a 12 hour period with five-minute increments
- Web app metrics:
  - CPU utilization, network-out
  - Load balancer metrics -
    - request count - number of requests completed or connections made in 1 minute intervals
    - target response time - keeps track of elapsed time after a request leaves the load balancer until a response from the target is received
- Collected over a 30 minute period in 1-minute increments

Performance Evaluation

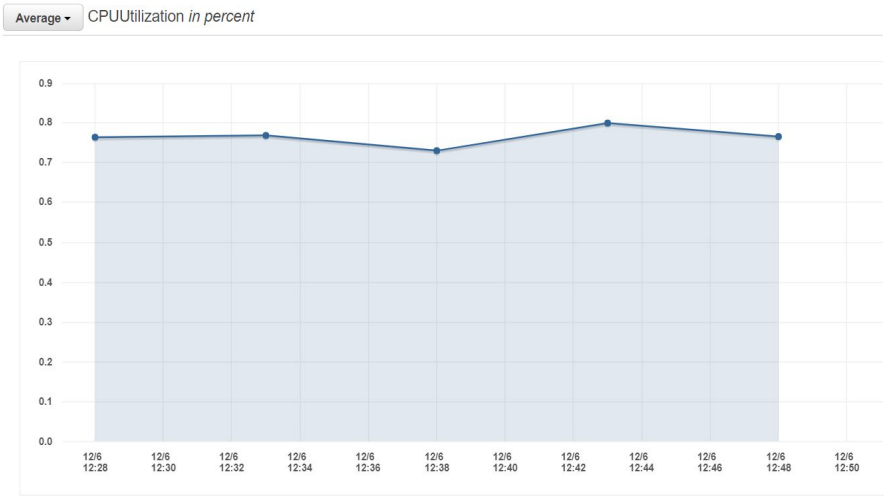
Results - Database metrics





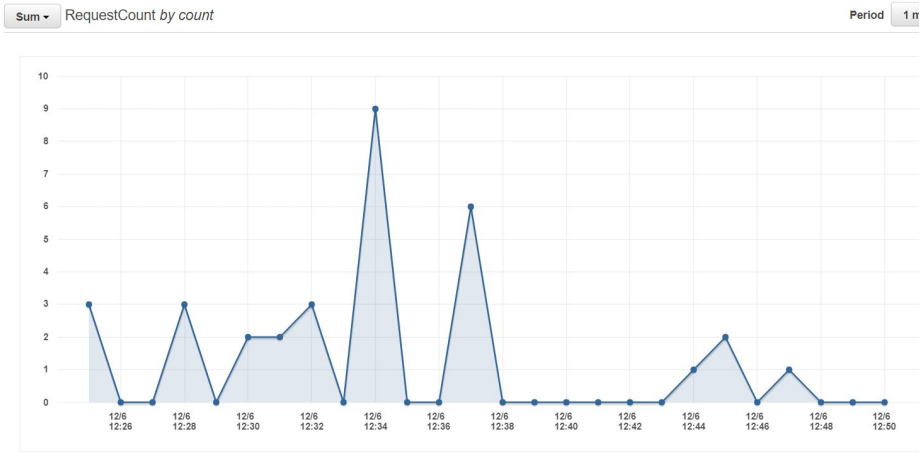
Performance Evaluation

Results - Web app metrics



Performance Evaluation

Results - Web app metrics



Conclusion/Future Directions

- Achieved initial goal of creating a Distributed Bookstore System
  - Users can buy and return books using the application
  - Administrators can update the inventory
  - Application and database communicate quickly and accurately
- Unable to create distributed databases due to AWS permission issues and time constraints
- Future Work:
  - Implement multiple distributed databases
  - Improve scalability of the system
  - Improve the applications user interface

Works Cited

[1] Avinash Kumar Reddy Subba Reddy Gari, *Performance evaluation of Cassandra in AWS environment: An experiment.* Blekinge Institute of Technology, (2017).

[2] Amazon, *Build a modern web application: Deploy a web application, connect to a database, and analyze user behavior.* https://aws.amazon.com/getting-started/projects/build-modern-app-fargate-lambda-dynamodb-python/?trk=gs_card

[3] Amazon, *Create and manage a nonrelational database with Amazon DynamoDB.* https://aws.amazon.com/getting-started/projects/create-manage-nonrelational-database-dynamodb/2/?refid=gs_card

[4] zData inc., *GemFire Performance Evaluation on AWS.* https://zdatainc.com/2015/02/gemfire-performance-evaluation-aws/

Questions?