MICHIGAN STATE UNIVERSITY

Project Plan

Cloud Management Platform

The Capstone Experience

Team GE

Lyle Fann
Vincent Ma
Will McPeek
Aaron Rosenwenkel
Nick Rutowski

Department of Computer Science and Engineering
Michigan State University

Spring 2016

From Students…
…to Professionals

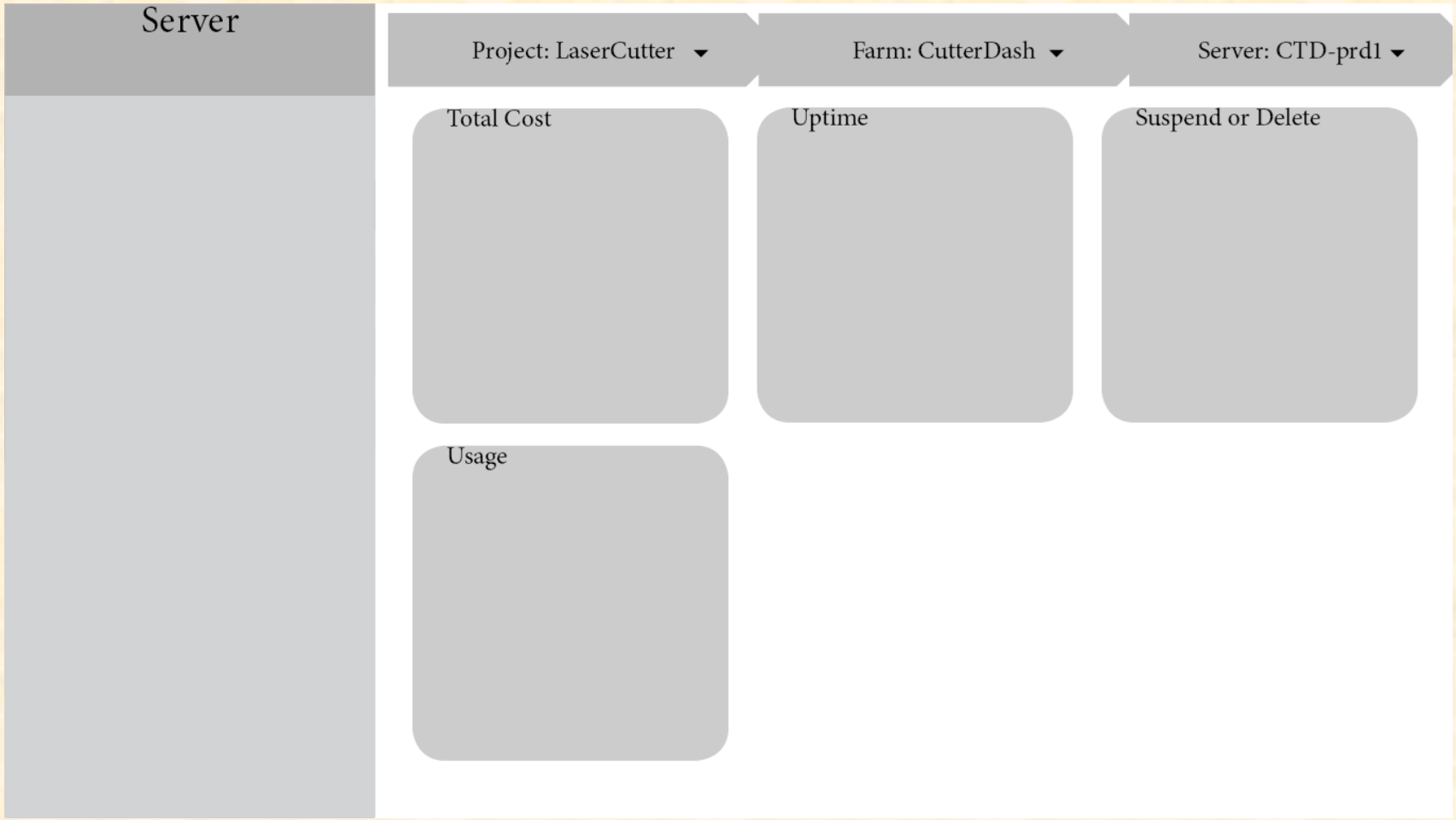
Functional Specifications

- Streamlines management of cloud resources
- Current cloud management system is too difficult and complex for day-to-day use
  - Wastes the time of IT professionals on simple tasks
- We propose a clean, easy to use webapp that will make the most commonly used features more accessible

Design Specifications

- Manage server “farms” on the cloud
  - Create/remove farms
  - Suspend/resume farms
  - Clone farms
  - View farm information
  - View costs associated with each project

Screen Mockup: Server Management



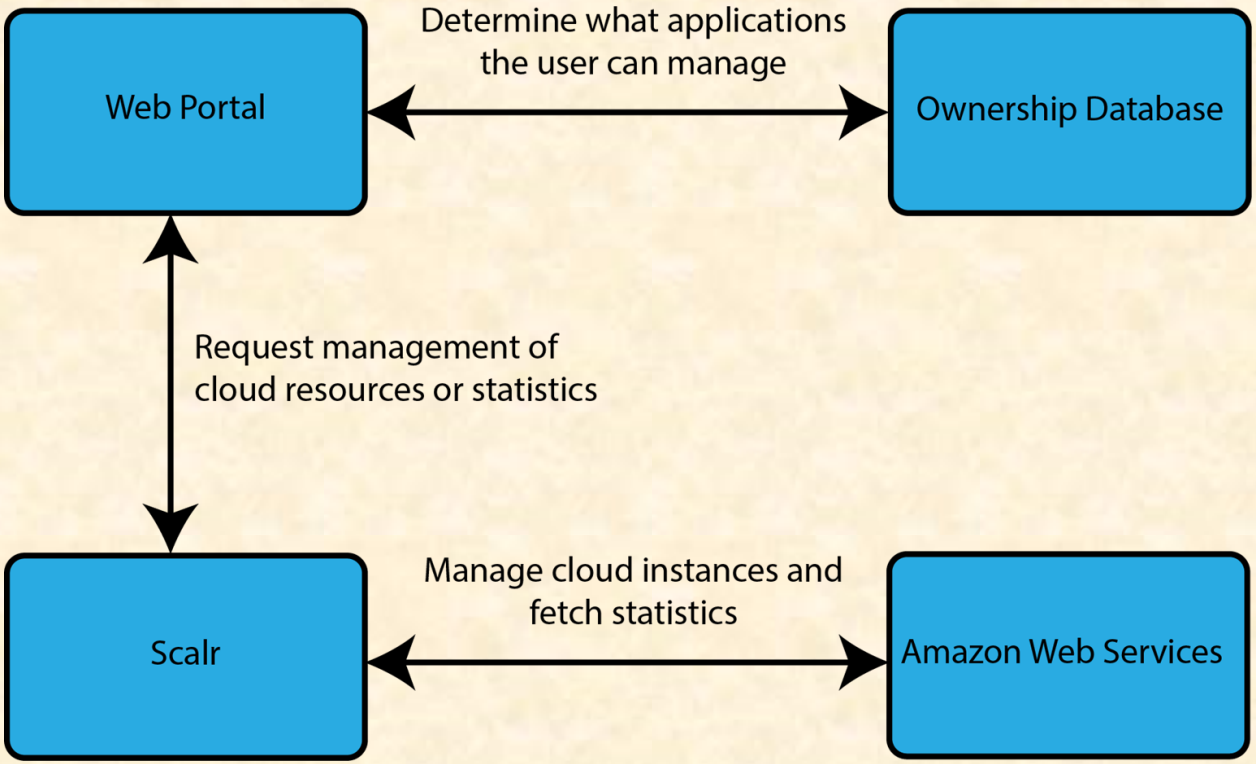
Screen Mockup: New Farm Modal

Screen Mockup: Farm Information Modal

Technical Specifications

- Web Portal
  - Django-based server, Python interface to Scalr API
- Ownership Database
  - PostgreSQL database on AWS
- Scalr
  - Platform-agnostic cloud management server
- Amazon Web Services
  - Cloud services provider managed by Scalr

System Architecture



System Architecture

Web Portal - Django-based server that renders the user interface. Uses a wrapper around the Scalr API for python to execute user commands and queries against the ownership database to determine what Scalr Projects a user can control.

Ownership Database - PostgreSQL database server that maintains the relationship between users and what Scalr projects they have access to.

Scalr - Cloud Management Server, exposes a frontend and API for managing cloud resources. Scalr is platform agnostic which enables us to interact with cloud resources from different platforms using 1 interface. In the prototype Scalr will only be managing AWS instances

Amazon Web Services - Cloud Provider that provides scalable application and database servers as well as load balancing tools.

System Components

- Hardware Platforms
  - Amazon Web Service
- Software Platforms / Technologies
  - Scalr: Existing webapp, manages server “farms” on the cloud
  - Python/Django: Back-end, interaction with Scalr API
  - CHEF: Automation of server setup
  - AngularJS or JQuery: Front-end
  - PX: GE proprietary front-end framework

Testing

- Manual Testing
  - Most testing will be manual
  - Emphasis on access policies and error checking
- Unit Testing
  - Leverage Django's unit testing features
  - Identify high-traffic areas and prioritize them in the testing

Risks

- Lack of access to GE internal network
  - No access due to HR and internal IP policy
  - Mitigation: Creation of a similar external
  - Status: Using resources available outside of GE
- Ambiguity of Requirements
  - No clear feature requirement at the beginning of the project
  - Mitigation: Continuing communications with GE
  - Status: First mockup has been sent
- Scalr configuration and Mismatch APIs
  - GE internal version of APIs is different from the external version
  - Mitigation: Communication with GE personnel familiar with role definition in Scalr will help mitigate that concern
  - Status: Issues are not yet resolved but communication is ongoing.

Risks

- AWS free account limitations
  - current AWS accounts are limited in the amount of servers/space/time
  - Mitigation: Communication with GE has led to the promise of additional resources to move beyond a free AWS account.
  - Status: Resources have been requested from GE and the project team is currently awaiting on the disbursement of these resources.
- High number of systems interacting
  - GE's current system uses vary of technologies
  - Mitigation: Team has been focused on learning the new technologies. Communication with knowledgeable GE personnel and online documentation
  - Status: Team is currently involved in learning technologies. 70% to 80% of required technologies have been suitably mastered.