Project Plan

Vinz

Team May14-32

Michael Davis

Maxwell Peterson

Jacob Hummel

Zach Heilman

Eric Feldman

Ario Xiao Qin

Advisor

Dr. Mitra

Client

Dave Tucker, WebFilings

Table of Contents

Problem Statement

Background

System administrators use the SSH (Secure Shell) protocol to administer linux servers. Authentication to these servers is rarely password based, and instead use public/private keypairs for added security. The SSH protocol has key pair authentication built-in, so properly configured SSH clients can seamlessly connect to a server without the user having to even see the authentication. In order for this to work, the user (client) must have a private key and the server must have the accompanying public key in its “authorized_keys” file.

Problem

With the advent of cloud computing, it is not uncommon for companies and organizations to have hundreds of linux servers running. If they are using public/private key pair authentication for all of these, it becomes a huge pain to manage access. Oftentimes system administrators find themselves writing custom shell scripts to SSH into each server and edit the authorized_keys files. Generally not all of these servers are doing the same things, so different people need access to different servers, making scripting these access changes difficult. Developers and Incident Response personnel sometimes require temporary access to debug production issues. Access requests and removals become tedious and inefficient.

Objective

Our goal is to remove this headache from the system administrator’s daily tasks. We will create a system that will manage access to a fleet of linux servers via private/public key pair authentication. The system will be easy to setup and install, and will allow users to easily manage access to logical groups of servers.

Existing Solutions

As mentioned in the problem portion, some organizations make custom shell scripts to remove or add public keys from/to authorized_keys files or simply do it by hand. This does not scale well and quickly breaks down once you have more than a few different logical groups of servers or access rules.

Larger organizations opt for a central directory server to handle authentication over protocols like LDAP. LDAP authentication and the management of a central directory server is much more complicated and difficult. Many organizations find that they do not have enough resources to devote to this kind of solution.

Deliverables

1. Github repository containing the source code of the project, as well as documentation surrounding the installation and use of the product.

2. A Personal Package Archive to automatically install the project on Debian based systems.

Functional Requirements

- Users must be able to upload SSH keys
- Users must be able to delete SSH keys
- Admins must be able to create other users
- Admins must be able to add new servers
- Admins must be able to put servers in logical groups
- Admins must be able to manage the groups of users and servers
- Admins must be able to control user access to servers and groups
- Admins must be able to view access reports

Optional Functional Requirements

- Admins must be able to set up systems to automatically add new servers

Non-Functional Requirements

- Users must be able to sign in using two-factor authentication
- Installation must be easy to install.
- Account deletion and removal of access should be as quick as possible for the case of a user being fired.
- Access restrictions must be enforced by scanning servers periodically
- Adding new APIs should be modular and easy to add

Project Management

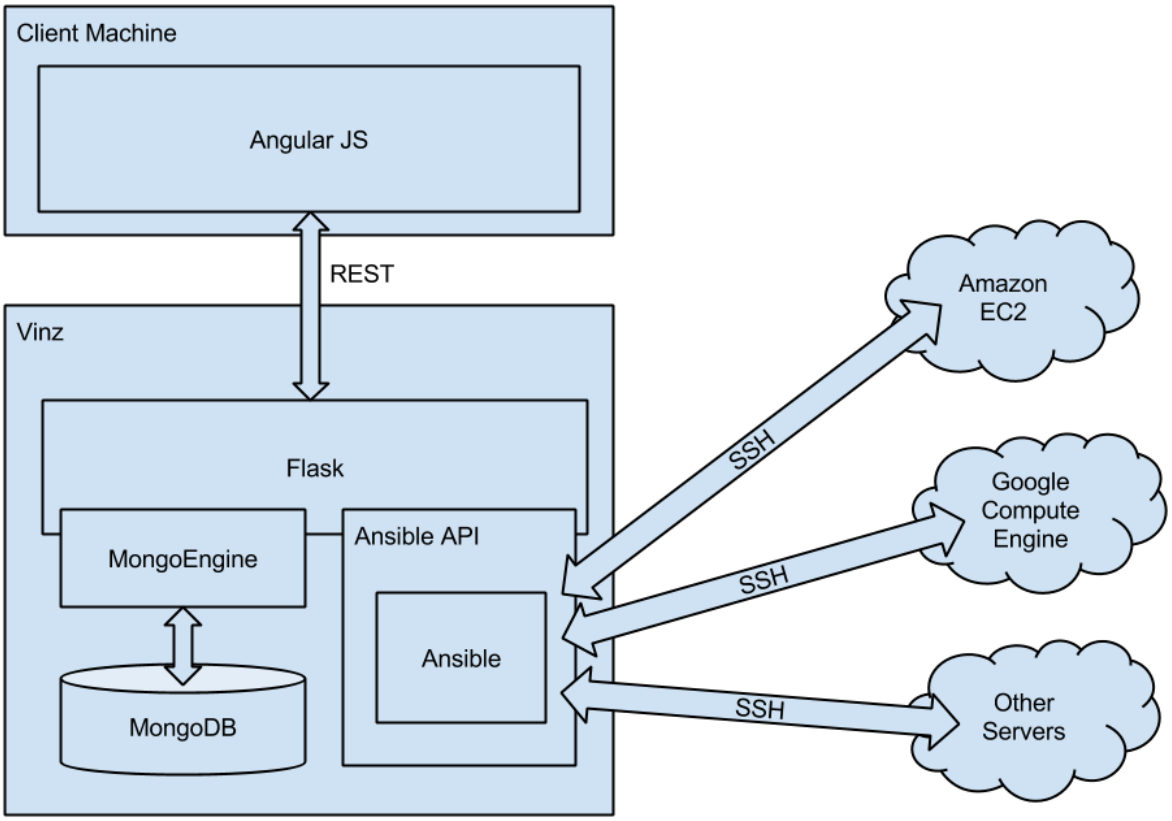
Work Breakdown Structure

We have assigned the tasks to each person with timeline on Smartsheet.
The project is one year based, and below is the major tasks for this semester. Plan may change as we go through the project and add more requirements and features into it.

	At Risk	Task Name	Start Date	End Date	% Complete	Assigned To	Comments
1							
2		**Create Project Plan**	**10/01/13**	**10/11/13**	**36%**		
11		**Software Development**	**10/01/13**	**03/31/14**			
12		**Create Design**	10/01/13	10/20/13			
21		**Prototyping and Implementation**	**10/21/13**	**01/31/14**			
22		Initial Module Implementations	10/21/13	10/30/13	0%	Michael	
23		Framework Construction	10/21/13	11/30/13	0%	Max, Ario	
24		Database Architecture Structure	10/21/13	11/30/13	0%	Max	
25		Frontend UI	10/21/13	11/30/13	0%	Jacob, Eric	
26		Auth & Security	10/21/13	11/30/13	0%	Michael	
27		Ansible API	10/21/13	11/30/13	0%	Michael, Zach	
28		Installation	10/21/13	11/30/13	0%	Zach	
29		Module Intergration	11/30/13	12/03/13	0%	All	
30		First run web app and do simple Module Test	12/04/13	12/10/13	0%	All	
31		Simple Testing	12/10/13	12/15/13	0%	All	
32		Semester report and Presentation	12/15/13	12/20/13	0%	All	
33		**Module and Prototyping Completion**	**12/21/13**	**04/30/14**			
34		Version 1.0 Releases	12/21/13	12/21/13	0%	All	
35		Feedback and Advise	12/21/13	12/30/13	0%	All	Get feedback from last semester test, res

Link to view detailed work break structure
https://app.smartsheet.com/b/home?lx=JLc5raZR13CzM40z6BcqnQ

Block Diagram



Resources

Infrastructure

Google Compute Engine: WebFilings will provide time on GCE

Software dependencies

A Linux system - primary target is Ubuntu Server
Python 2.7
Ansible
MongoDB
Flask
mongoengine
AngularJS

Development tools

Vagrant
VirtualBox
Text editor (Sublime Text 2) or Python IDE (Eclipse+PyDev)

Risk and Mitigation Strategies

WebFilings

WebFilings has offered time on Google Compute Engine for testing purposes. Although this would be beneficial, if WebFilings needs to withdraw its offer Vinz can theoretically be implemented on virtually any Linux system; other computing resources will suffice, even virtual machines on personal laptops.

Licensing

At the end of the project we would like to open-source the code - this would greatly improve its maintainability. However, we will need to consult with both our client (WebFilings) and the university to determine what rights we have with the project.

Security

Since Vinz's purpose is to grant and revoke access to systems, it is imperative that it correctly revokes access to critical systems swiftly if lingering access would pose a security risk to the environment (for example, if an employee is fired, it is paramount that his or her access is revoked to prevent retaliatory actions by that employee). We will address this through rigorous testing on multiple systems.

Loss of a team member

It's always a possibility that a team member is unable to complete the project. To mitigate this, we will use a combination of through documentation and a modular design that allows components to be developed in isolation from each other.

Bottlenecks

Through both feedback from our client and software testing, we will identify and remedy any bottlenecks in UI, computation, and database access.

Scalability

It is conceivable that Vinz could be implemented on networks with hundreds of users and thousands of systems. Beyond bottlenecks in database queries and computation, we will need to ensure that our project is massively parallelizable to deal with swift access revocation across a large number of systems.

Testing and Evaluation

Testing is an important aspect of this project. In order to guarantee quality, we will test the code. At all times, the master git branch will be production ready.

Automated Testing

Eventually, automated tests will be run upon each build/deploy of the system. Automated tests will run all of the integration and unit tests. A successful build means that all of the integration and unit tests have successfully passed. A failed build means that any of the unit or integration tests fail.

Unit Testing

Everything crucial will be unit tested. Crucial is determined by the developer, at their discretion, based upon the perceived complication and importance of each unit. Unit tests must be submitted with the code they are testing, during the same task check in.

Integration Testing

Everything crucial will be integration tested. Crucial is determined by the developer, at their discretion, based upon the perceived complication and importance of each module. Integration tests must be submitted with the code they are testing, during the same task check in.

Code Review

Whenever a task has been deemed complete, the developer may submit their code for a code review. Code reviews will be completed using GitHub. Any code that is checked in must undergo a code review by at least one other member of the team. The automated tests must have been run, and successfully passed in order to check in.

Beta

We will release a beta of Vinz to be used by our stakeholders. Once we reach beta stage, there will be constant interaction between our development team and our users. This stage will provide us with valuable feedback. This feedback will allow us make appropriate enhancements, bug fixes, and discuss additional features.

Maintenance

With enough support, we would like to continue this project beyond the duration of this course. We believe it to be solving a real issue in any company that makes use of cloud environments. Since we are open sourcing this solution, we would like to allow continued enhancements from the community. GitHub provides us with the necessary tools for community involvement.