Amazon Cloud Sales Demonstration Environment

Sentinel Prime
Jennifer Goldbach, Robert Jones, Anne Markis, Nick Quaranto, Christopher White

Paychex
Duane Gosley, Michael Mosley

Faculty Mentor
Tom Reichlmayr

Project Overview
The Amazon Cloud Sales Demonstration Website involves deploying and running several of Paychex’s payroll and human resources products on Amazon EC2. The company’s current demonstration process is difficult to scale up for hundreds of sales representatives distributed across the country. Team Sentinel Prime will be creating a demonstration platform that will allow untrained sales reps to effectively demonstrate the Paychex product suite. Since deploying Paychex’s actual applications is beyond the scope of this project, team Sentinel Prime will create applications that mimic the basic functionality of the Paychex applications.

The project involves several improvements over the current manual way of setting up demonstrations for customers. Data integration across the various applications will allow sales reps to show customers the benefits of using multiple products to manage their payroll and human resource efforts. Cloud computing will allow Paychex sales reps to easily demonstrate their offerings to customers across many industries. In an effort to make the demonstrations seem fresh and relevant, there must be a method of providing and maintaining a set of realistic data for customers to evaluate. By using EC2, Paychex will save money as salespeople will be able to terminate their EC2 instance when they are finished, and Paychex will only be responsible for paying for the hours the sales demonstration environment was running on EC2.

Basic Requirements
There will be two user roles for this system, a salesperson and an administrator. The majority of the time the salesperson will be using this to schedule and create sales environments for potential customers. They will run their demonstrations in the Amazon EC2 environment by using the URL generated when an instance starts. When they have finished a demonstration, they will shut down the instance. Administrators can view all running instances, and start or stop instances if needed. They are able to update the applications stored on EC2 so salespeople are all using the same version of the software, and they can change the database the sales demonstrations interact with.

Since salespeople do not have a strong technical background, the interface Sentinel Prime is providing to the EC2 environment will mask the details of EC2, allowing the
salesperson to choose details for their sales demonstration, while not worrying about the
technical nature of starting an EC2 instance and pulling the data associated with their
demonstration.

**Administrator User Stories:**

- As an administrator, I want to add a new application in order to allow salespeople to demonstrate it
- As an administrator, I want to be able to sign into the application so I can monitor and alter the sales environment
- As an administrator, I want to be able to set up a new environment tailored to a potential customer so I can verify my changes were successful
- As an administrator, I want to be able to add a new industry to select when creating a new demonstration environment so that new industries can be tailored to
- As an administrator, I want to be able to remove an app from the system and destroy any running instances to keep irrelevant apps out of the sales demonstrations
- As an administrator, I want to update an app on EC2 so that the sales demo is up to date
- As an administrator, I want to be able to update all the baseline data so that the demonstrations are up-to-date and relevant.
- As an administrator, I want to destroy any demo instance so that I can manage active demos
- As an administrator, I want to be able to view all created demos so that I can manage demo instances.
- As an administrator, I want to be able to find outdated versions of running applications so I can deal with them

**Salesperson User Stories:**

- As a salesperson, I want to demo the single-sign on capabilities so that customers can see that integration across applications is possible
- As a salesperson, I want to be able to set up a demo environment tailored to the potential customer so that they can view the products in a more relevant manner
- As a salesperson, I want to demonstrate data integration across apps so that the customer can see the value in multiple apps.
- As a salesperson, I want to be able to log into the app so that I can create a demonstration
- As a salesperson, I want to be able to destroy my own demonstration instance when a specific demo is over so that I can complete the demonstration
- As a salesperson, I want to be able to select an industry category when creating my demo environment so that I can customize the demonstration
- As a salesperson, I want to be able to reset the demo data so that I can depend on unchanged data before a demonstration
Constraints
This project is based around the requirement of using Amazon's EC2 cloud computing environment to deploy and run applications already in existence at Paychex. Therefore, using the Amazon AWS SDK was a requirement for the team's software interacting with the Amazon environment. Other considerations include that of Paychex’s applications’ environment. To facilitate for this, the EC2 application had to be deployed on the cloud on a Windows Server and use SQL Server.

The technologies currently used by Paychex include a Citrix environment, hosting a WIN32 client, accessing data from SQL and an ASP.NET application using IIS, accessing data from SQL. The team developed applications to mimic these applications and deployed these on EC2 in place of the Paychex applications. Later into the project the requirement for deploying Citrix applications was dropped due to the sheer complexity of the system, licensing issues, and lack of time to for proper implementation.

After these constraints were met, Sentinel Prime was able to choose other implementation technologies and methods. The team is using C# with ASP.NET MVC 2 for a primary implementation technology. The Amazon instances Sentinel Prime created use Windows Server 2008, containing IIS 7.0 and SQL Server 2008 for databases. The instances also use CopSSH, an OpenSSH/Cygwin installer for remote access. Visual Studio was used for development.

Development Process
Scrum, an agile and iterative process, was used as the technical process. Team Sentinel Prime chose to adapt scrum to the project and use short, two-week iterations because of the timeframe of the project. Paychex did not mandate this strategy; however, the sponsors approved this process early on in the project.

Regular meetings with Paychex ensured all issues could be brought up quickly. For urgent matters, Basecamp, a project management tool, was used for communication with Paychex. Paychex approved the user stories, and they have been re-visited and updated as necessary throughout the project. The team used Pivotal Tracker to keep track of these user stories, as well as current, past and future iterations, and team velocity. The team assigned story points to each story based on expected difficulty of completion, and used these values in conjunction with priorities given by Paychex, to plan future sprints.

The team held a sign-off meeting at the end of each sprint to demonstrate the current functionality to Paychex. This was helpful in identifying areas where Sentinel Prime misunderstood Paychex's requests, and brought up questions for future work. Finally, a sprint retrospective was held at the end of iterations to facilitate process improvement.

Project Schedule: Planned and Actual
Using the Scrum methodology, the team created user stories during iteration zero and had those stories approved by Paychex. Points were assigned to each story based on complexity and difficulty of implementing the story, and sprints were planned based on the team's velocity. Half of the first quarter was spent planning, and becoming familiar
with the EC2 environment as it relates to the project. The first milestone identified was manually deploying an application on an EC2 instance, and saving that instance for later use. Paychex's directions to the team were to focus on the salesperson user stories, so those user stories were planned in earlier iterations, leaving most of the administrator stories for late in the project.

**Initial Planned Schedule:**

<table>
<thead>
<tr>
<th>Sprint</th>
<th>Start</th>
<th>End</th>
<th>Story Points</th>
<th>Burndown</th>
<th>RIT Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5-Nov</td>
<td>14-Jan</td>
<td>0</td>
<td>85</td>
<td>Winter 1-5</td>
</tr>
<tr>
<td>1</td>
<td>15-Jan</td>
<td>28-Jan</td>
<td>12</td>
<td>73</td>
<td>Winter 5-7</td>
</tr>
<tr>
<td>2</td>
<td>29-Jan</td>
<td>11-Feb</td>
<td>10</td>
<td>63</td>
<td>Winter 7-9</td>
</tr>
<tr>
<td>3</td>
<td>8-Mar</td>
<td>18-Mar</td>
<td>17</td>
<td>46</td>
<td>Spring 1-2</td>
</tr>
<tr>
<td>4</td>
<td>19-Mar</td>
<td>1-Apr</td>
<td>18</td>
<td>28</td>
<td>Spring 3-4</td>
</tr>
<tr>
<td>5</td>
<td>2-Apr</td>
<td>15-Apr</td>
<td>16</td>
<td>12</td>
<td>Spring 5-6</td>
</tr>
<tr>
<td>6</td>
<td>16-Apr</td>
<td>29-Apr</td>
<td>12</td>
<td>0</td>
<td>Spring 7-8</td>
</tr>
</tbody>
</table>

**Sprint Zero Breakdown:**

Week 1:
- Met with the sponsors
- Produce project synopsis

Week 2:
- Technology/Sales meeting with sponsors,
- Generate questions based off technology meeting to determine scope

Week 3:
- In-depth technology meeting
- Initial revision of the Project Plan

Week 4:
- Researching EC2
- Prototyping based off project synopsis

Week 5:
- Create UI mock-ups of application
- Evolve prototype
- Start iteration 1

Week 6:
- Finalize Prototype
- Set up scrum board
- Start initial backlog

**Actual Schedule:**

<table>
<thead>
<tr>
<th>Sprint</th>
<th>Start</th>
<th>End</th>
<th>Story Points</th>
<th>Burndown</th>
<th>RIT Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5-Nov</td>
<td>14-Jan</td>
<td>0</td>
<td>85</td>
<td>Winter 1-5</td>
</tr>
<tr>
<td>1</td>
<td>15-Jan</td>
<td>28-Jan</td>
<td>12</td>
<td>73</td>
<td>Winter 5-7</td>
</tr>
<tr>
<td>2</td>
<td>29-Jan</td>
<td>11-Feb</td>
<td>10</td>
<td>63</td>
<td>Winter 7-9</td>
</tr>
<tr>
<td></td>
<td>Mar 18</td>
<td>Mar 17</td>
<td>Apr 28</td>
<td>Apr 12</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>8-Mar</td>
<td>18-Mar</td>
<td>17</td>
<td>46</td>
<td>Spring 1-2</td>
</tr>
<tr>
<td>4</td>
<td>19-Mar</td>
<td>1-Apr</td>
<td>18</td>
<td>28</td>
<td>Spring 2-4</td>
</tr>
<tr>
<td>5</td>
<td>2-Apr</td>
<td>15-Apr</td>
<td>16</td>
<td>12</td>
<td>Spring 4-6</td>
</tr>
<tr>
<td>6</td>
<td>16-Apr</td>
<td>29-Apr</td>
<td>12</td>
<td>0</td>
<td>Spring 6-8</td>
</tr>
</tbody>
</table>

**Discrepancies**

The project scope changed significantly from the initial draft from Paychex to the redefined project at the end of RIT week 3. Although the changes were beneficial for the team, it did add time to Iteration 0. As it was clear from the first meeting with Paychex, the project scope would need to be scaled down. Therefore, the team did not plan sprints until the new project proposal was delivered and user stories could be extracted and approved. The team experienced unexpected difficulties in learning the new EC2 environment, and many questions about data integration and the use (or lack of use) of a Citrix environment on an EC2 instance. When the team realized they were behind schedule, they prioritized user stories with Paychex to avoid skipping some of the highest priority features, and planned more story points for future iterations. Since much of the time spent by team members in the early iterations was spent learning and setting up environments, the team was confident they could handle future sprints with more story points.

**System Design**

**Architecture**

*Web Application (Demolition)*

A custom-written ASP.NET MVC 2 application is the heart of our architecture. It is a user-authenticated web portal that allows salespeople to start up demos, manage custom data sets, and provide a simulation of a live Paychex application environment to potential clients.

System administrators are also able to perform multiple tasks with the web application. They can upload new versions of applications for salespeople to boot, act as any salesperson in case they need support, and perform basic management of EC2 instances.

*Console Applications*

There are two console applications that perform the bulk of work behind the scenes that the Demolition web application cannot do in the HTTP requests. Originally these console applications ran as Windows Services, but we ran into several problems trying to reinstall them on existing instances. This problem was only evident on the instances, and no one on the team was able to reproduce the failure on their local machines. It was enough of a blocker to our releases that we decided to simply tear out the services and just run the console applications by themselves.
The first application is a multi-threaded job queue, which runs different jobs (called “Workers” internally). These workers are in charge of creating new instances, terminating existing ones, and generating SQL that is inserted on instances once the database is ready. It was necessary to use a queue like this because instances commonly take 10-30 minutes to boot up and respond to SSH. It also took several extra minutes for the DNS to resolve. Without the queue, salespeople would only be able to start one instance at a time, and only one salesperson could be starting demonstrations at a given time.

The second application is a periodic instance checker, which currently pings each instance every minute. It is checking the Amazon EC2 instance that each specific demo is running on. The database belonging to that demo is hashed to a value. This value is compared to the stored value for a clean database. If the database hashes do not match, then the demo is flagged as Dirty on the main Demolition database. This will show on the demo information page of Demolition, so that the user knows if the data has been changed. There can be a delay of up to one minute after a change happens for the demo to be flagged due to the checker running on intervals.

**Tradeoffs**

**Web Application vs. Desktop Application**

From the start we decided to go with a web application instead of a desktop application. The simple reasoning behind this is that it is simpler to maintain, access, and deploy. Instead of trying to install an application on potentially thousands of salesperson laptops or tablet PCs, they can simply access a website with their Paychex credentials. There was no other motivation to use a desktop application, especially since it centralized all configuration and deployment information.

**One vs. Many instances per Demo**

Determining how we would lay out what went on an EC2 instance was a challenge. Originally we thought we could go with a more federated infrastructure, which could help with scaling out the application in the long run. This would involve having each EC2 instance in charge of one main thing: for example, SQL Server would only run on one instance, another could serve up only one type of application, and so on.

After discussing this with the Paychex sponsors, we determined that joining responsibilities onto one instance would make deployment simpler to start, and would help with building out the proof of concept. One instance is now in charge of its own database and applications, and conceptually represents one demonstration. This also reflects how Paychex deploys their applications in real life to their customers, so we hope it will be easier for them to build their infrastructure on top of our work.

**Running Background Tasks**
Another tradeoff dealt with using an existing, open source job queue or messaging system versus writing our own. We wanted to do the bare-minimum, simplest thing that would allow us to run background tasks. Originally we did research into running these as a Windows Service, using threads in ASP.NET, scheduling cache jobs, and more. None of these seemed reliable or easy, and it would have seemed like a “hack”. Existing “enterprise” job queues seemed like overkill for what we needed: just a simple way to run some code outside of the process. The team also determined that we did not need to scale out to a separate server yet, so using something like ActiveMQ or MSMQ to communicate with separate machines was unnecessary.

Instead we wrote a simple LIFO, multithreaded job queue that could run multiple jobs at once as a console application. We wrote the workers for the queue in such a way that it should be easy to port it to another platform if necessary, but it met our core requirements that it was simple to set up and deploy and did not introduce another dependency to our system. The job queue puts each worker onto the ThreadPool.

**EC2 Loadout**

The following diagram aims to give a bird’s eye view of what is launched on each EC2 instance.

![EC2 Loadout Diagram](image)

On a given EC2 instance, all the software necessary for one demonstration is available. IIS serves up a demo’s applications on unique ports starting at 8000. Both applications connect to the same SQL Server database on that machine, which contains specific industry data for that demonstration. There is a database for each application running on
the instance, as well as a shared master database containing any data and changes that have been made after the start of the demonstration.

Outside of this instance, other instances exist that other salespeople may have started. Each is a complete and separate sandbox, so changes to one demo will not affect others. They can be spun up and terminated on demand by the salespeople that own the demo or administrators that deem it necessary. Since they exist within the Amazon cloud, performance is not effected by the number of demonstrations running, nor are any demonstrations effected when EC2 instances are started or stopped.

**Demo Workflow**

From a high level viewpoint it is beneficial to understand the normal flow a user and the application takes in order to get from nothing to a customized demo.

![Demo Workflow Diagram]

The first part of the workflow involves getting the demonstration environment set up. This requires the salesperson to log in and choose the applications that were uploaded previously by an administrator that will be shown off. The salesperson then selects from a set of industries, such as Trucking, Restaurant, or Construction, which contains specific data, such as tips for the Restaurant industry, which can be modified by administrators. Once all of this information has been chosen, the demonstration can begin the boot process.

Behind the scenes, our job queue kicks into action and interacts with the AWS SDK for .NET to launch a new instance from our previously created AMI. The job queue then has to wait until it can actually communicate with that instance, which may take up to 20 minutes. Once it has started up and responds to SSH, the worker then uploads the salesperson’s chosen applications, starts them on IIS, and generates SQL for the given industry and runs it on that instance’s SQL Server.

Once the job queue’s workers have completed, the demonstration is considered “ready” and the Demolition web application’s database is notified. The salesperson can then go into demo mode, which allows them to show off our simulation of single sign on and
jump into the running applications to demonstrate the booted applications with the custom data set chosen to the potential clients.

Testing

The team chose to use an EC2 instance as a Hudson continuous integration server for the duration of this project. The team created both unit and acceptance tests to run each time a commit was made to the project. Several utility classes were written to make this testing easier, as the testing involves uploading files, and there were several tricky problems with automated browser testing. These tests did not start or stop EC2 instances, as issues that arose with those test cases were beyond the team's control, and that would make each test suite run take an extra 10-30 minutes. All of the testing with instances had to be done manually. Logging was added to all of the Worker classes so anyone testing could monitor the Demolition Queue progress during testing and have a record of any errors that occurred. Bugs encountered during testing were recorded in Pivotal Tracker, and assigned to a team member to fix and re-test.

Process and Product Metrics

Process Metrics:

Two process metrics were tracked: release burndown and team velocity.

The SCRUM process breaks up tasks for each iteration, or sprint, into separate backlogs. Each backlog item has a set number of hours that the team has estimated that the task will take. This chart provides a view into how many hours are left in the sprint according to how many tasks have been completed so far. As it is updated each day, it is easy to gauge when the sprint will be done.

Velocity measures how much work was done over an iteration, and it is usually counted by the number of story points assigned to each feature. This metrics helps with determining how much can be completed in one iteration and helps with planning future work. Also, tracking sprint task and how long it takes to complete it will be essential in determining how the team is progressing with implementation and testing.
Project Metrics:

Two project metrics were tracked: code coverage and code complexity.

Metrics on how much of the code is covered by tests from both unit and integration levels will show how committed the team was to solid testing of the implementation and provide insight into problem areas that may be less reliable than others. The team will not be shooting for 100% code coverage since that is usually a goal that provides little benefits for too much effort, but instead will be seeking a maintainable level of coverage that still assures reliability and dependability for the codebase. At the time of completion, the following code coverage was observed:

Demolition.Models: 54%
- Job: 98%
- Model: 94%
- App: 79%
- Instance: 71%
- Industry: 70%
- User: 57%
- Demo: 45%

The Code Complexity metric was considered to ease the handoff of the code to Paychex. Unfortunately, this added little value to development, especially due to the fact that ASP.NET MVC 2 added a significant amount of complexity at the start. At the time of completion, the Code Complexity is as follows:
One of the main requirements the project sponsors require is that the applications can withstand a certain amount of concurrent users, and this can be proven by using performance testing tools on the website the team creates. A community accepted tool to do this is Apache Bench, which can give an accurate rating of how many requests per second the server can handle. Although we did no formal performance testing, EC2 has a hard limit on the number of instances you can boot unless if a sales contact is made with Amazon. The Amazon contact provided to the team to manage this problem was on vacation and did not respond in time for the end of the quarter. If the Demolition website runs on its own EC2 instance, this would leave 19 other instances, each running a single demo. There’s no limit to the amount of applications that are running on each demo, so once Paychex contacts Amazon to raise that limit, they could potentially have thousands of demos running at once, allowing hundreds of salespeople across the country to show off their products.

**Product State at Time of Delivery**

At the time of delivery, the product was finished: all promised user stories were completed, including documentation. While there will likely be many additions to the project after the handoff, these were not part of the scope of the project, and were not the concern of Sentinel Prime. Included in the documentation was a list of known issues.

**Project Reflection**

What we feel went right:

- **Good grasp on scope**
  - The first thing that was tackled was the scope. After a series of talks with the sponsor, the initial scope was reduced to something more manageable for the time allotted. It also opened up communication channels between the team and the sponsor, creating an open environment to discuss issues. Regular meetings assured that scope matters were constantly addressed, and feedback on progress was readily available.

- **Good sponsor communication**
  - The team received feedback from the sponsor after each two-week iteration. Additional meetings were scheduled as needed, often towards the end of quarter to discuss administrative and book-keeping matters. These regular meetings, along with active and immediate discussions on
Basecamp, allowed for the quick reaction to problems encountered, and adoption of suggested changes.

- **Good selection of tools**
  - Using Hudson for continuous integration was a good idea, as it provided immediately feedback on new code committed. WatiN for acceptance testing contributed largely to the robust test suite. Pivotal, although not used to its full extent, was very effective in providing an easy way to keep a backlog, and assign task owners. Basecamp was a good communication tool.

- **2 week iterations**
  - Having 2-week development cycles was a natural pace for the team. It allowed for the selection of a predictable number of story points that the team could usually commit to. It was also easy to put in 2 weekends per iteration, which greatly helped the success of the iterations.

**What we feel could have gone better:**

- **Lack of focus on task dependencies**
  - Some tasks that depended on other tasks to be completed were scheduled to be completed before, or at the same time. This created a significant amount of rework and delays for otherwise straight-forward tasks.

- **Task ownership miscommunication**
  - After the first iteration, the team decided that a better task-assignment strategy should be applied, as there as regularly chunk of a project that some team members weren’t familiar with. It was decided that there should be ‘task owners’ – a person assigned to be the one responsible for knowing the status of the task, not necessarily being the one to complete it all the way through. This concept broke down as task owners worked as though they were solely responsible for finishing and testing the task. Some task owners did not ask for assistance, creating bottlenecks and delays.

- **Communication with Amazon**
  - Paychex provided us contact information for Amazon tech support, but we have yet to hear back from anyone regarding the problems we ran into. Since we ran into the biggest problems very late in the project, there was not a lot of time for us to wait for Amazon to get back to us and potentially solve the 'insufficient capacity' error.

- **Decision to use ASP.NET MVC 2**
  - This created some friction in starting development. Most of the team was familiar with C# and ASP.NET WebForms, but MVC was clearly a different animal. It’s a much cleaner approach to web development and definitely the most modern web framework to come out of Microsoft in recent years, and with the RTM getting strong feedback we wanted to try it out. The hardest part was educating the team about where code is placement conventions, which were not always obvious. After a few weeks of working with it, most of the issues were ironed out.
EC2 Lessons Learned

This project was unique in that it will be delivered to Paychex as a proof-of-concept that they will use as a starting point for future work with Amazon EC2. Since the work done during the project involves the mock-up applications, when Paychex receives the work from Sentinel Prime, they will be replacing those with their own applications. Because these applications were developed to Paychex's specifications, the transition is not expected to be overwhelming. A main concern of management at Paychex is to keep costs of these demonstrations low, so an analysis of cost savings using Amazon EC2 versus another technology is part of the schedule. Paychex requested the team keep track of problems encountered with using EC2 and the AWS SDK.

The lessons learned document can be found in “Amazon Cloud Lessons Learned.docx” in this submission.

References

Apache Bench http://httpd.apache.org/
Basecamp http://basecamphq.com/
Pivotal Tracker http://www.pivotaltracker.com/
ASP.NET MVC 2 http://www.asp.net/(S(d35rmemuono1wvm1gsp2n45))/mvc
TestDriven.Net http://www.testdriven.net/
Amazon AWS SDK http://aws.amazon.com/sdkfornet/
AWS EC2 Technical FAQ http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1145&categoryID=100
EC2 Bootstrapper http://github.com/qrush/ec2bootstrapper
Setting up SSH http://www.digitalmediaminute.com/article/1487/setting-up-a-sftp-server-on-windows
Hudson Continuous Integration
http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson

Wizardby database migration
http://code.google.com/p/octalforty-wizardby/

Wizardby command line reference

Wizardby MDL reference
http://code.google.com/p/octalforty-wizardby/wiki/MdlReference

SQL Last Changed Row data reference
http://stackoverflow.com/questions/2423909/is-there-meta-data-i-can-read-from-sql-server-to-know-the-last-changed-row-table/2424035#2424035

Citrix
http://www.citrix.com/English/ps2/products/subfeature.asp?contentID=1689905

Deploying .NET Apps on EC2
http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1791

Other EC2 References
http://learn.iis.net/page.aspx/161/an-introduction-to-windows-powershell-and-iis-70/
http://blog.crowe.co.nz/archive/2006/05/24/633.aspx
http://www.experts-exchange.com/Programming/Languages/Scripting/Powershell/Q_23676100.html
http://support.microsoft.com/kb/968929
http://www.15seconds.com/Issue/041223.htm
http://stackoverflow.com/questions/777396/asp-net-deployment

MVC Authentication
http://forums.asp.net/t/1398778.aspx