Spring | 17

An Analysis of Heroku and AWS for Growing Startups

Colton Stapper

This project presents information on the architecture of modern cloud-hosting platforms, and gives an analysis of two common Platform-as-a-Service companies: Heroku and AWS.

California Polytechnic State University

Table of Contents

Introduction

The Problem

A local San Luis Obispo startup, PolyRents, aims to streamline the rental housing application process by having both tenant applicants and landlords use a single, organized online platform. They launched their beta with landlords a few weeks ago and plan to have several thousand users in the next year. PolyRents already has a React-based web application running with a Python Django REST API that is deployed on the Heroku cloud-hosting platform.

In preparation for their expansion, PolyRents reached out asking for help with their back-end infrastructure needs. PolyRents wants help exploring cloud-hosting alternatives like Amazon Web Services (Amazon), to learn how to make a robust backend development and deployment pipeline. They would like to know about modern infrastructure techniques to optimize scalability, response times, and to reduce overall cost to the company. PolyRents wants to know at what point the use of an abstracted infrastructure service is no longer worth the cost, and when they should switch to a cheaper alternative.

Scope of Work

The project requires four main components: research, metrics and criteria, implementation, and evaluation. First, the project requires research on the cloud compute architecture. How do Heroku and Amazon's pricing models, implementations, and user-configurations differ? To what extent does each cloud-hosting platforms differ, and what metrics and criteria should be chosen to measure the costs and advantages? The project will consist of a technical specification that outlines the different pieces of software that will be implemented to evaluate the metrics. After analysis of the results, final recommendations will be made to the PolyRents development team.

Solution

The project will analyze the cost-benefit of Heroku and Amazon's features and predict if and when PolyRents should switch to AWS based on the evaluation of the criteria. If the project is able to answer their questions about the future of the development of their backend software, then it can be considered a successful solution to the company's problem.

Background

To understand the scope of the project requirements, some general technical details must be defined. These terms will help to explain which services Heroku and AWS are trying to cater, and which components are user-configurable.

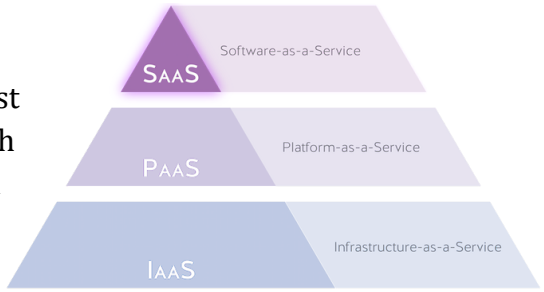

Figure 1 - Cloud Compute Pyramid. The higher the pyramid goes the more abstraction there is, but the less control there is over the lower layers (Sandoval).

Infrastructure-as-a-Service (IaaS)

Infrastructure is the foundation for the cloud compute model. "IaaS provides hardware such as CPUs, memory, storage, networks, and load-balancers" (Hassan). In order to run applications in a scalable, reliable, and secure way, developers cannot just run their program locally; they need to deploy their software on a cluster of trusted servers. The servers should have regular maintenance, like "examining folder permissions, ensuring adequate redundancy of systems, installing security patches, and reading server logs for security alerts" (ECC IT Solutions). Companies like Amazon and Microsoft are IaaS providers (Richman) because they setup and maintain data centers for developers to use.

Platform-as-a-Service (PaaS)

Platform-as-a-Service is a cloud-computing environment that helps developers write, run and manage the lifecycle of their applications. "The service provider not only is responsible for provisioning and managing the lower level infrastructure resources, but also for providing a fully managed application development and deployment platform (Joshi). Both Amazon and Heroku are PaaS providers because they give developers tools to deploy their applications on the necessary infrastructure.

Software-as-a-Service (SaaS)

Software-as-a-Service is the product that end users interact with. Usually in the form of a subscription, clients pay to use a SaaS product online that is hosted on centralized servers (Sandoval). "SaaS services…may be designed to access hardware resources only through a PaaS layer" (Hassan). PolyRents can be considered a SaaS in this instance, since the product is a scalable service that interacts with a PaaS to host the site and store data in a centralized way.

Cloud Computing Architecture

The cloud-compute model is a combination of IaaS and PaaS. "Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the Internet with pay-as-you-go pricing" (Amazon). Each cloud-hosting provider has different hardware and features, but the cloud-compute architecture typically has the following components:

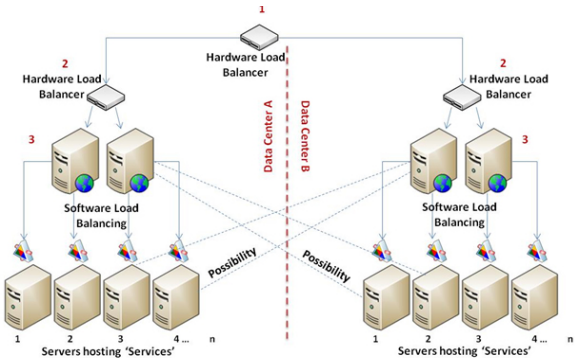

Figure 2 – Cloud Compute Architecture (Alur). Load balancers receive requests from end users and forward them to different data centers. Then, Distributed web, application, and database servers handle the requests.

Database Server

A database server is either a computer dedicated to large-object data store, or a computer that has database software installed so that it can store data while running websites or applications alongside it. PolyRents currently uses Heroku Postgres Database Server to store user information.

Application Server

An application server is a computer that runs developer applications. Commonly written in Python, Ruby or Java, the back-end application takes HTTP requests from the web server to present dynamic web information, perform computations, or store data in a database. PolyRents deploys a Python Django REST application to talk to the web server and database.

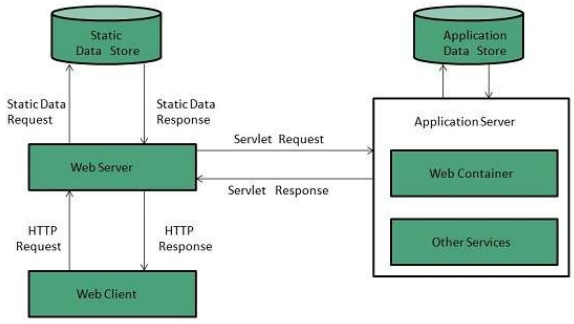

Figure 3 – Web and Application Servers (TutorialsPoint). The client asks the web server for static data in the form of HTTP requests, and the web server makes calls to the application server to access other data stores, like database servers.

Web Server

"A web server stores and delivers the content for a website" (NGINX). A PolyRents client accesses the website on their web browser, asking for data using HTTP requests. The web server sends the client HTTP responses with data, such as HTML documents or images

(Ubuntu). PolyRents uses Gunicorn, a pure-Python HTTP web server, to run their application and take concurrent client requests.

Load Balancer

Once multiple application and web servers are deployed, what happens when all user traffic is sent to the same servers, and there are other idle servers awaiting workload? Load balancers are a piece of hardware or software that "distribute client requests or network load efficiently across multiple servers" (NGINX). Load balancers also detect unhealthy servers and reroute future user traffic to healthy instances (Amazon). Load balancers provide higher availability and reliability in the cluster, and add the flexibility to easily increase or decrease the size of the cluster in response to user traffic.

Cloud Computing Hosting Platforms

Amazon Web Services (AWS)

Services

Amazon provides a cloud compute service platform called AWS to help developers deploy highly available and scalable backend and web applications. They have services like Elastic Compute (EC2), Elastic Load Balancing (ELB), Lambda, Auto Scaling, Simple Storage Service (S3), Relational Database Service (RDS), CloudFront content delivery network, and Elastic Beanstalk (Amazon Web Services). These services integrate with each other and have individual pricing models.

Elastic Beanstalk

Elastic Beanstalk is meant to act as a Heroku competitor. "AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with…Python…on familiar servers such as Apache" (Amazon Web Services). Elastic Beanstalk configures EC2, S3, RDS, and ELB instances for the developer to quickly deploy their application. AWS gives the developer dedicated EC2 instances, meaning that they have root access to the compute server and can SSH directly.

Heroku

Heroku acts as a middleman between the application developer and AWS. Heroku tries to abstract the complexity of DevOps from the developer by offering PaaS software that gives the user a smooth web interface and CLI to easily deploy their applications. Instead of giving developers a dedicated EC2 instance, Heroku deploys many different developer apps on what they call "web dynos", which are shared AWS EC2 instances. Because other developers run their applications in isolated containers alongside each other sharing resources, it is possible that "these dyno types may experience some degree of performance variability depending on the total load on the underlying instance" (Heroku). Heroku has not published the extent that neighbors affect a developer's application in a dyno.

Frameworks and Libraries

ReactJS Library

React is "a JavaScript Library for building user interfaces" (Facebook). PolyRents built their web application using the React library.

Python Django REST Framework

Django is a Python Web framework meant to make programming web applications simpler (Django). PolyRents implemented their API using Django REST Framework, a toolkit built for developers to help write Python REST APIs.

Description

Specification

To understand the advantages and costs of Heroku and AWS, this project will deploy PolyRents' application in an Elastic Beanstalk environment and a Heroku environment:

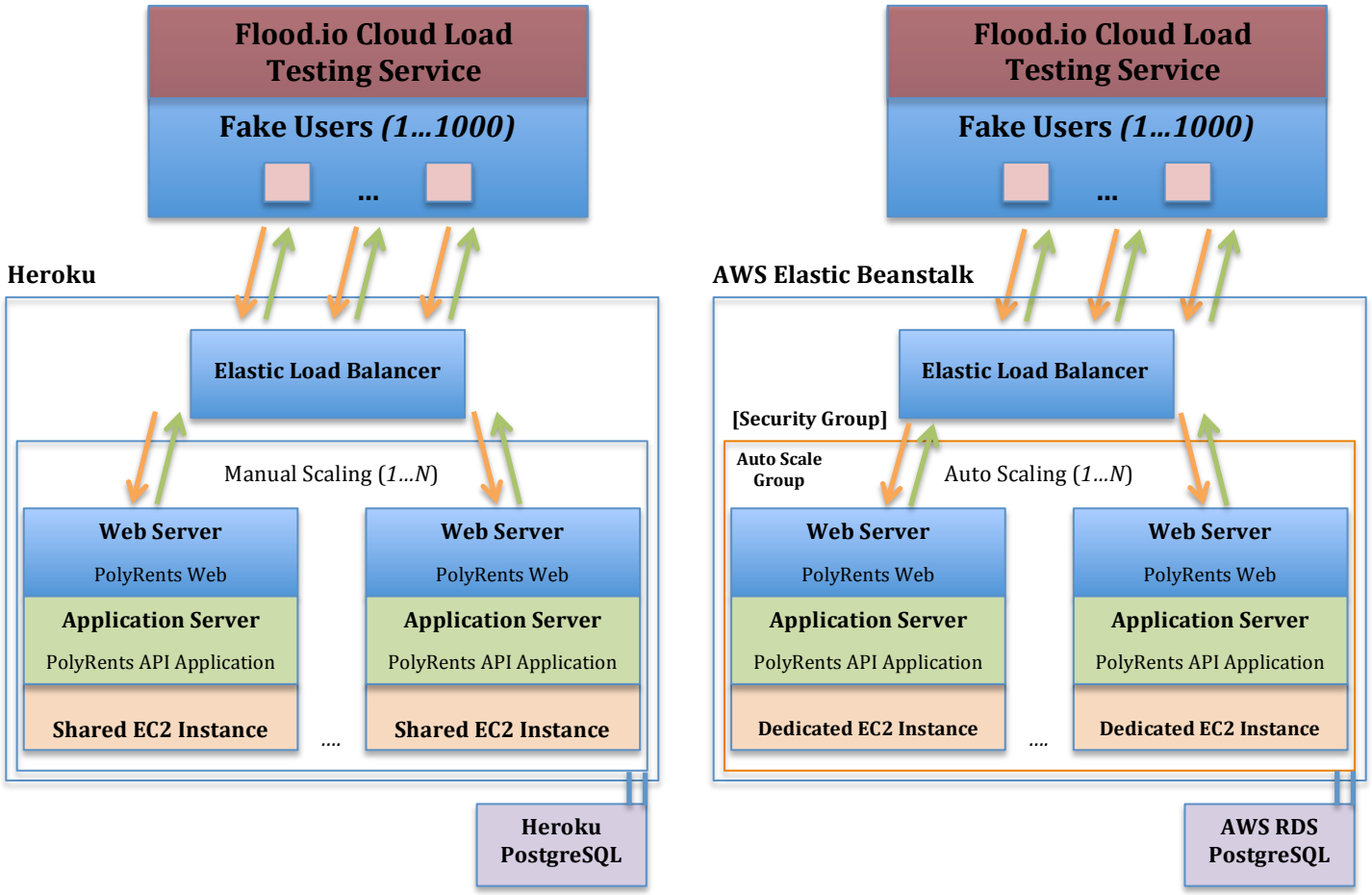

Figure 4 – Project Specification.

Figure 4 shows the individual components of each PaaS architecture and their handling of web requests. Although similar architecturally, Heroku and AWS provide access to and configuration of different components.

Unique AWS configurations

AWS gives the lead developer the ability to configure specific security rules for other developers, labeled [Security Group] in Figure 4. These security rules give specific permissions to members of a team, preserving Fail-Safe defaults and preventing developers from accessing parts of the project that they should not have permission to see. Heroku on the other hand does not give the developer this granularity.

AWS also allows developers to configure Auto-Scaling rules, allowing simple EC2 instances to scale automatically according to traffic demand. Heroku only allows this kind of scaling for their most expensive servers. Costs and tradeoffs will be analyzed in the next section to determine if AWS will be a better option for PolyRents.

Evaluation

Metrics

Heroku and AWS will be evaluated on the following metrics:

- Scalability
- Total cost
- Efficiency
- Availability
- Response times
- Error rates
- Configuration time
- Deployment time
- Maintenance time

The last three metrics are of great importance to the CTO of PolyRents – he is concerned that the time required learning, configuring, and maintaining an AWS environment is not worth the potential lesser costs.

Experiments

Configuring AWS Elastic Beanstalk

Getting an Elastic Beanstalk environment setup takes slightly longer than Heroku because AWS requires the developer to configure additional policies. This is a table listing the time it took to configure AWS Elastic Beanstalk, including the debugging time:

Installing the AWS EB CLI	1 minute
eb init – initialize the application **AIM credentials, setting user permissions and getting keys**	5 minutes
eb create – setting environment and DNS_CNAME names	1 minute
eb deploy – deploy the application to the environment (failed the first time, fixed errors below).	1 minute
Issue with adding other packages **Add .config file in .ebextensions to install git and postgres packages**	30 minutes
Issue with WSGI set to a default path and could not be found. **Had to specify "polyrents/wsgi.py"**	30 minutes
Set the DJANGO_SETTINGS_MODULE and PYTHONPATH in another .config file to point the EC2 instances at the correct settings file (polyrents.settings.development), and the right python path (/opt/python/current/app/polyrents:$PYTHONPATH).	10 minutes
Modify the Heroku DATABASE_URL code to use the new AWS RDS Postgres hostname information.	15 minutes
Stuck on incorrect dj-stripe installation because of invalid egg info (not a problem with AWS)	3 hours
Stuck on incorrect database module import – was importing MySql instead of PostgreSQL (from an incorrect copy and paste from AWS docs).	2 hours
Time re-setting the environment because of modifying the RDS instance from small size to micro size, adjusting configuration settings.	30 minutes
Time trying to modify the number of running processes (could not get this to work on the Apache server).	1 hour

Figure 5 - The time of configuring the AWS Elastic Beanstalk environment.

Total time to learn and configure environment: 8 hours of time.

However, once a developer has learned and gone through the process of launching an Elastic Beanstalk environment, all configurations and setups afterwards are much faster. Many issues arose from the unfamiliarity of working with Elastic Beanstalk before.

Deployment time is relatively quick, between 10 – 20 seconds.

Load and Scale Testing

PolyRents will eventually have thousands of users but do not have that amount of traffic at the moment. Load tests are required to measure the response times and scalability of the two platforms. To model the influx of thousands of users, a cloud load testing service called Flood.io will be used to perform these large-scale tests. Flood.io takes as input a list of REST API requests to call and the number of users, and simulates user requests by sending thousands of requests per second to the Heroku and AWS PolyRents applications. The flow of requests that Flood.io will send and receive can be seen in Figure 4, and the variables involved in the experiment are in Figure 6:

AWS Elastic Beanstalk URL: *http://polyrents-api-loadtest1.us-west-1.elasticbeanstalk.com*
Heroku base URL: *http://polyrents-api-loadtest.herokuapp.com*

Number of concurrent users: *1000 users*

REST APIs to hit:

- GET */api/tenant/application/bio*
- GET */api/tenant*
- PUT *a long, new bio to /api/tenant/application/bio*
- PUT *a short, new bio /api/tenant/application/bio*

Controlled variables:

- *Time of experiment (8pm)*
- *Length of experiment (25 minutes)*
- *Location of requests (same node region – US N. California)*
- *Same maximum number of concurrent requests (1000 users)*
- *Same ramp-up interval of concurrent requests (ramp-up every 30 seconds)*
- *Tear-down and re-create the environment before every test*

Figure 6 – Load and scale tests. Flow.io will hit the Heroku and AWS endpoints with 1000 concurrent simulated users in controlled tests.

Results of the tests are shown in Figures 7 and 8.

Load Test Results

Heroku Results

Figure 7 – Gunicorn WSGI server running on Heroku.

AWS Elastic Beanstalk Results

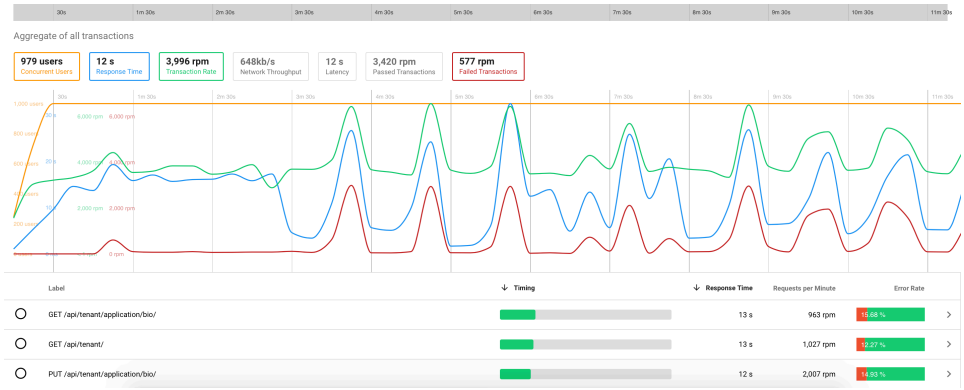

Figure 8 – Apache WSGI server running on AWS Elastic Beanstalk with one EC2 instance.

Heroku dynos had inconsistencies – every test produced a different Flood.io graph. AWS performed in a sinusoidal way, responding to a lot of requests quickly and then more slowly. In contrast, Heroku would slow down suddenly at large intervals – this appears to be part of the resource management that Heroku forces upon developers since the applications are running on shared EC2 instances.

Heroku had considerably worse response rates – AWS responded to requests after about 13 seconds on average, and Heroku responded about 26 seconds on average. The two platforms did successfully complete about the same amount of requests during the experiments.

AWS and Heroku Costs

Heroku and AWS offer different tiers for servers, databases and load balancers that have different pricing.

Heroku

Type	Cost per hour	Cost per month
Heroku Postgres (Hobby add-on)	N/A	$9
Heroku Web Dyno (Standard)	N/A	$25
Load Balancer	N/A	N/A

Figure 9 – cost of components under the Heroku platform (Heroku).

During development, PolyRents is using a cheaper database and server, so their monthly costs come to about $7 instead of the projected $34.

AWS

Type	Cost per hour	Cost per month
RDS PostgreSQL (t2 micro)	$0.024	$17.28
EC2 instance (t2 small – 2GB Ram)	$0.031	$22.32
Elastic Load Balancer instance (Classic)	$0.028	$20.16
S3 instance	$0.026 per GB	About $0.026

Figure 10 – cost of components under the AWS platform (Amazon).

The AWS costs are under the assumption that the compute cluster is running 24 hours a day, when realistically the service will be less utilized at night and in the morning. All of these costs are covered under AWS Free Tier for a year, so like Heroku, there is little to no price during the early development of the application.

AWS allows the developer to configure Auto-Scaling, giving the compute cluster the ability to scale out during high traffic points of the day. When PolyRents has more users, Auto-Scaling would cut down the wasteful spending that they occur now on the Heroku dynos at night when there is less traffic. This will keep their service highly available and cheaper with little added maintenance or configuration.

Recommendation

PolyRents should switch to AWS as soon as their user traffic gets near 200 requests a second. At that traffic, Heroku response times will become slow (1 – 5 seconds), and their service will become nearly unusable. Once spending about 5 – 8 hours of configuration time and migrating their databases, their service would be completely usable on AWS and PolyRents could Auto-Scale their platform during peak usage times.

This project can be considered a success because it answered PolyRents' core question wondering when they should be concerned about switching to alternative cloud hosting platforms.

Conclusion

The project was an incredibly educational experience – I chose the topic of cloud infrastructure and hosting because it is becoming increasingly more important in an expanding IoT and SaaS-based economy. In fact, California Polytechnic State University just announced that it will switch its technologies to use an AWS cloud-hosting environment.

It is important to consider the security of these applications. It is not ethically responsible to let the cloud-hosting platform abstract the infrastructure away to the point that developers cannot control the access permissions of their backend applications and servers. AWS maintains this control by giving users the ability to define clear security rules around every service.

A future direction that is more complex would be finding the best combination of all types of Heroku dynos and AWS components by tweaking configurations. This work would find the optimal combination for growing startups to deploy their applications in a relatively cheap but available and scalable environment.

Works Cited

Alur, Sandeep J. Enterprise Integration and Distributed Computing: A Ubiquitous Phenomenon. September 2008. Microsoft Corporation. <https://msdn.microsoft.com/en-us/library/cc949110.aspx>.

Amazon. Amazon Web Services (AWS) - Cloud Compute Services. 2017. <https://aws.amazon.com>.

Amazon Web Services. Overview of Amazon Web Services. April 2017. <https://d0.awsstatic.com/whitepapers/aws-overview.pdf>.

Django. Django Documentation. 2017. Django Software Foundation. <https://docs.djangoproject.com/en/1.11/>.

ECC IT Solutions. Server Maintenance and New Server Installation. 2017. <https://eccitsolutions.com/services/server-maintenance-and-new-server-installation/>.

Facebook. React - A JavaScript library for building user interfaces. 2017. Facebook. <https://facebook.github.io/react/>.

Hassan, Qusay. "Demystifying cloud computing." The Journal of Defense Software Engineering 1 (2011): 16-21.

Heroku. Dynos and the Dyno Manager | Heroku Dev Center. 2017. Heroku. <https://devcenter.heroku.com/articles/dynos>.

Joshi, Sunil. What is Platform-as-a-Service (PaaS)? 17 February 2014. <https://www.ibm.com/blogs/cloud-computing/2014/02/what-is-platform-as-a-service-paas/>.

NGINX. NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy. 2017. <https://www.nginx.com/resources/glossary/>.

Richman, Dan. Microsoft Azure just behind Amazon Web Services in Gartner's new IaaS rankings. 4 August 2016. GeekWire, LLC. <https://www.geekwire.com/2016/microsoft-azure-just-behind-amazon-web-services-gartners-new-iaas-rankings-google-distant-third/>.

Sandoval, Kristopher. Living in the Cloud Stack – Understanding SaaS, PaaS, and IaaS APIs. 8 July 2015. Nordic APIs. <http://nordicapis.com/living-in-the-cloud-stack-understanding-saas-paas-and-iaas-apis/>.

TutorialsPoint. Web Server. 2017. TutorialsPoint. <https://www.tutorialspoint.com/internet_technologies/web_servers.htm>.

Ubuntu. Web Servers. 2017. Ubuntu. <https://help.ubuntu.com/lts/serverguide/web-servers.html>.