PARTLY CLOUDY

DESIGN & DEVELOPMENT OF A HYBRID CLOUD SYSTEM
“This project is focused on building and implementing a single course exploration and enrollment solution that is intuitive, interactive, and end-user focused.”

— Enrollment Tools Project Charter
COURSE SEARCH & ENROLL APP
TIMELINE

• Application in development since 2016
• First debuted at Winter SOAR early 2017
  – Handled Summer SOAR 2017, Winter SOAR 2018
• The Registrar’s Office’s request for an AWS account approved in December 2017.
  – Approved for Production in AWS in March 2018
  – Production launch in AWS on April 1, 2018
• Handled part of Summer & Fall priority enrollment in April 2018
  – Plan is to take the place of the Student Center by Summer 2019
COURSE SEARCH & ENROLL APP

Requirement Aspects

• Scalable — Prone to being swarmed by traffic during enrollment periods
• Enrollment workflow relies on slower external services
• Curricular data and class status required to update in near real-time
COURSE SEARCH & ENROLL APP

Platform & Architecture

• Course search using Elasticsearch database
• Comprised of 5 Spring Boot scalable microservices & 2 stand alone Spring Boot applications
• Leverages Spring Cloud/Netflix libraries
  – Zuul, Ribbon, Hystrix, Feign, Config Server
• Shared enrollment workflow state is stored in Redis
• JMS messaging is used to route enrollment workflow processing
COURSE SEARCH & ENROLL APP

API Gateway

- Search
- Planner
- Enrollment
- Scheduler

EACH SERVICE = 1 OR MORE BOUNDED CONTEXTS
COURSE SEARCH & ENROLL APP

REPLICATED SERVICES ORGANIZED INTO POOLS
COURSE SEARCH & ENROLL APP

ONLY ONE SERVICE ACCESSES EACH RESOURCE
COURSE SEARCH & ENROLL APP

JMS & REDIS ARE USED TO SHARE STATE
Deployment Concerns

- Applications are delivered and deployed using Docker images
- Configuration is externalized
- All microservices are stateless
- Connections to campus services
  - Oracle
  - ActiveMQ/JMS
  - CAOS (Curricular, Academic and Operational Store)
  - SIS (Student Information Services)
12 FACTOR APPLICATION CHARACTERISTICS

• I. Codebase
  – One codebase tracked in revision control, many deploys

• II. Dependencies
  – Explicitly declare and isolate dependencies

• III. Config
  – Store config in the environment

• IV. Backing services
  – Treat backing services as attached resources

• V. Build, release, run
  – Strictly separate build and run stages

• VI. Processes
  – Execute the app as one or more stateless processes

• VII. Port binding
  – Export services via port binding

• VIII. Concurrency
  – Scale out via the process model

• IX. Disposability
  – Maximize robustness with fast startup and graceful shutdown

• X. Dev/prod parity
  – Keep development, staging, and production as similar as possible

• XI. Logs
  – Treat logs as event streams

• XII. Admin processes
  – Run admin/management tasks as one-off processes
WHY GO TO AWS?

• Highly variable usage profile
  – Middle of a break vs priority enrollment period vs SOAR
• “As fast as you are willing to pay for”
  – Performance as financial issue instead of technical issue
• Easy to get redundancy
• Built in tooling for microservices and SOA
• Operational Automation
WE BUILT THIS...
TWICE
TEST AND PROD
WHAT ‘THIS’ IS

- VPC
- Route 53
- EC2
- Application Load Balancer
- Elastic Container Service
- AWS Elasticsearch
- Elasticache/Redis
- Elastic File System
VPC

• Virtual private cloud
• Provides a way to provision virtual network interfaces in private, non-routeable subnets (such as 10.x.x.x) with optional public IPs
  – This is taken care of automatically when you launch a VM
  – Most public IP addresses are ephemeral, and will change upon stop and start of the virtual machine
  – Five public static Elastic IPs per account that can be assigned to virtual machines
    • Because ARPAgeddon
• Allows a choice of an internet gateway or a NAT gateway for each subnet
  – Internet gateways allow inbound traffic
    • Outgoing traffic uses the IP address of the VM instance
  – NAT gateways do not allow inbound traffic
    • Outgoing traffic uses the IP address of the NAT gateway
VPC

VPC Subnet 192.168.3.0/24

NAT Gateway 35.169.11.26

Internet Gateway

To Internet

Connections to UW Services

VPC Subnet 192.168.12.0/24

VM

SSL

(Inbound Traffic) Load Balancer

VM
VPC

- Make everything private by default
  - Private IP + Subnet with NAT Gateway
  - NAT Gateway has public IP and has outbound access to Internet Gateway
- Set up a bastion host with an Elastic IP for ops access to private subnets
  - Allow firewall access to service team VPN IPs as needed
    - SSH – Just developers and operators
    - 80,443 – RO enrollment support staff
ROUTE 53

• Provides both external and internal DNS services
• 3 private zones
  – *.enrollment.dev, *.enrollment.test, *.enrollment.prod
• Changes this: prod-enroll-app.f91kwb.ng.0001.use2.cache.amazonaws.com:6379
  – To this: redis.enrollment.prod:6379
• Allows swapping service instances without changing configuration
**EC2**

- Provisions and runs VMs
- Provides configurable Launch Configurations
  - Defines configuration of fleets of identical VMs
- Provides autoscaling groups that are used by Target Groups
  - Autoscaling groups provision VMs based on Launch Configurations
- Target groups are comprised of virtual machines providing the same services
  - VMs launched by Autoscaling groups can register with Target Groups
  - Target groups run health checks on their members
AN OBLIGATORY METAPHOR

Pets
- Pets have names
- Pet get taken to the vet if they get sick
- VMs in DoIT’s data center are Pets

Cattle
- Cattle have numbers
- Cattle are killed and replaced if they get sick
- VMs on Amazon are Cattle
EC2

Autoscaling Group

- Registers VMs with Load Balancer
- Performs Health Checks on VMs
- Uses Launch Configuration

Launch Configuration
- t2.medium, no public IP,
- custom User Data script that installs various tools

Target Group

Load Balancer

- Sends requests to VMs

VMs

- Uses custom User Data script to install various tools
APPLICATION LOAD BALANCER

- Application Layer (L7)
- Provides SSL termination for multiple host names
- Provides either an internal or internal facing load balancer endpoint
- Provides some of the functionality of an ESB
- Allows routing http and https requests using port, path and hostname to Target Groups of containers
- Provides the capability to have ‘sticky sessions’
APPLICATION LOAD BALANCER VS Shibboleth IDP

- ALB is not a single server, but several of them
  - One in each availability zone
  - Has multiple IP addresses
- Requests look like they are coming from the load balancer endpoints, not the user's IP address

$ host enroll.wisc.edu
enroll.wisc.edu is an alias for prod-external-lb-957103706.us-east-2.elb.amazonaws.com.
prod-external-lb-957103706.us-east-2.elb.amazonaws.com has address 18.220.222.26
prod-external-lb-957103706.us-east-2.elb.amazonaws.com has address 18.216.14.58
prod-external-lb-957103706.us-east-2.elb.amazonaws.com has address 52.14.65.193
APPLICATION LOAD BALANCER
VS SHIBBOLETH IDP

10.0.5.23
Apache 1

10.0.11.3
Apache 2

10.0.16.41
Apache 3
APPLICATION LOAD BALANCER VS SHIBBOLETH IDP

SOLUTION

- Configuring `mod_remoteip` in apache
  - RemoteIPHeader      X-Forwarded-For
  - This caused the user’s requests to look like they came from that user’s IP address, instead of the somewhat random ALB endpoint they were routed through
ELASTIC CONTAINER SERVICE

- Runs Docker containers organized into Target Groups of instances of the same application.
- Provides scaling controls and an API
- Ensures that the number of running Docker containers of each application that are passing health checks is consistent. Terminates unhealthy instances and replaces them.
- Runs the containers on either EC2 instances that we manage (with an autoscaling group) or in Fargate (a type of serverless server farm….that’s a thing now).
  - Fargate costs more than EC2 VMs on their own
ELASTIC CONTAINER SERVICE

Cluster

Service

Task
- Container
- Container

Task
- Container
- Container

Service

Task
- Container
- Container

Task
- Container
- Container

Target Group

EC2 Virtual Machines
Autoscaling group

Fargate
ELASTIC CONTAINER SERVICE
ALB ROUTES REQUESTS TO TARGET GROUPS

- Target group handles health checks of members
- Load balancer routes requests to target groups
  - Hostname
  - Request path
  - Port
OTHER SERVICES

• Elasticache
  – Redis and/or Memcached

• Elasticsearch
  – “You know, for search”

• Elastic File System
  – NFS4, Encrypted

• Code Commit
  – Git, used to store application configuration

• Simple Systems Manager
  – Parameter store: encrypted credential values
  – Patch Manager
FRONT END CODE IN BROWSER INITIATES A REQUEST:
GET /api/enrollment/current
REQUEST ENTERS AWS NETWORK VIA VPC INTERNET GATEWAY
REQUEST ROUTED TO EXTERNALLY FACING ALB MAPPED TO ENROLL.WISC.EDU
ROUTED THROUGH APACHE/SHIB INSTANCES FOR AUTHENTICATION
APACHE PROXIES REQUESTS TO INTERNAL ALB

services.enrollment.prod:8080
GATEWAY SERVICE PROXIES
REQUESTS TO INTERNAL ALB
services.enrollment.prod:8083
ENROLLMENT SERVICE REQUESTS
DATA FROM SIS WEBSERVICES
OUT THROUGH THE NAT GATEWAY
TO THE UW DATA CENTER AND BACK
CURRICULAR DATA REQUESTED FROM SEARCH SERVICE ADDED TO RESPONSE
AND FINALLY BACK TO THE USER
WE LEFT A SERVICE BEHIND
WE LEFT A SERVICE BEHIND

- Indexer service pulls curricular data from the CAOS SOAP API and updates Elasticsearch
- Full refresh in place every night
- Real time updates via JMS messages sent from CAOS
- More traffic between indexer and CAOS than between indexer and Elasticsearch
  - Because SOAP
- Access to AWS Elasticsearch controlled via IP access list and secure IAM token
ALERTING, MONITORING AND LOGGING

• ELK stack for structured application logs
• Metrics aggregated in Grafana dashboards
  – Application level/business rule type metrics captured via Graphite time series server
  – Cloudwatch metrics
  – Grafana alarms sent to Slack
• Cloudwatch Alarms
  – Based on changes in Cloudwatch metrics
  – Cloudwatch Alarms sent to Simple Notification Service (SNS) Topics
  – Lambda function listens to SNS Topic and forwards alerts to Slack
    • And soon to UW Network Operations Center OMi system
• Cloudwatch logs for container and infrastructure logs
• S3 for load balancer logs
• Cloudtrail Logs
SIZE, COST

• Non-peak scaling
  – 5 x t2.small (1 vCPU, 2 GB RAM, $0.023/hour)
    • On Demand: $0.115/hour (total)
    • Reserved for 1 year: $0.07/hour (total)
  – 15 x t2.medium (2 vCPU, 4 GB RAM, $0.0464/hour)
    • On Demand: $0.696/hour (total)
    • Reserved for 1 year: $0.435/hour (total)
  – 4 x t2.large (2 vCPU, 8 GB RAM, $0.0928/hour)
    • On Demand: $0.3712/hour (total)
    • Reserved for 1 year: $0.232/hour (total)

  – Monthly Compute Costs:
    • On Demand: $851.18 (total for 30 day month)
    • Reserved for 1 year: $530.64 (total for 30 day month)

• Other significant costs
  – Elasticsearch
  – Elaticache
LESSONS LEARNED

• Do not become attached to your virtual machines
• Application load balancer for sticky sessions (for Shib)
  – Configure mod_remoteip to substitute the client’s IP address from X-Forwarded-For header
• Use a private subnet with a fixed NAT gateway IP address to deal with UW on-premises firewalls
• Be ready to request higher limits from AWS support if you need them
  – Don’t forget other limits (database connections, etc…)
• Careful with your health checks!
  – 200, 302
  – Defaults to 5 seconds
• Autoscaling is extremely challenging
• The more AWS does for you, the more they charge you for it
  – EC2 vs Fargate vs Lambda
  – Know your usage profile to figure out which suits the task
• There are many, many ways to do the same thing on AWS
QUESTIONS?

brian.hill@wisc.edu