

Lecture 14: March 5

Deployments



Agenda

- **Demo 3**
- March Sprint
- Deployment Technologies
- Example Industry Setup
- Upcoming Deadlines
- Deployment Tutorials



Demo 3 + Feedback

- This week you'll meet w/ instructors as a team to review February progress
- As next week is Spring Break, we'll send out written feedback instead of meeting to review.
- We're working through February Sprint grades and will share them in the coming weeks as well. Generally things are looking good!



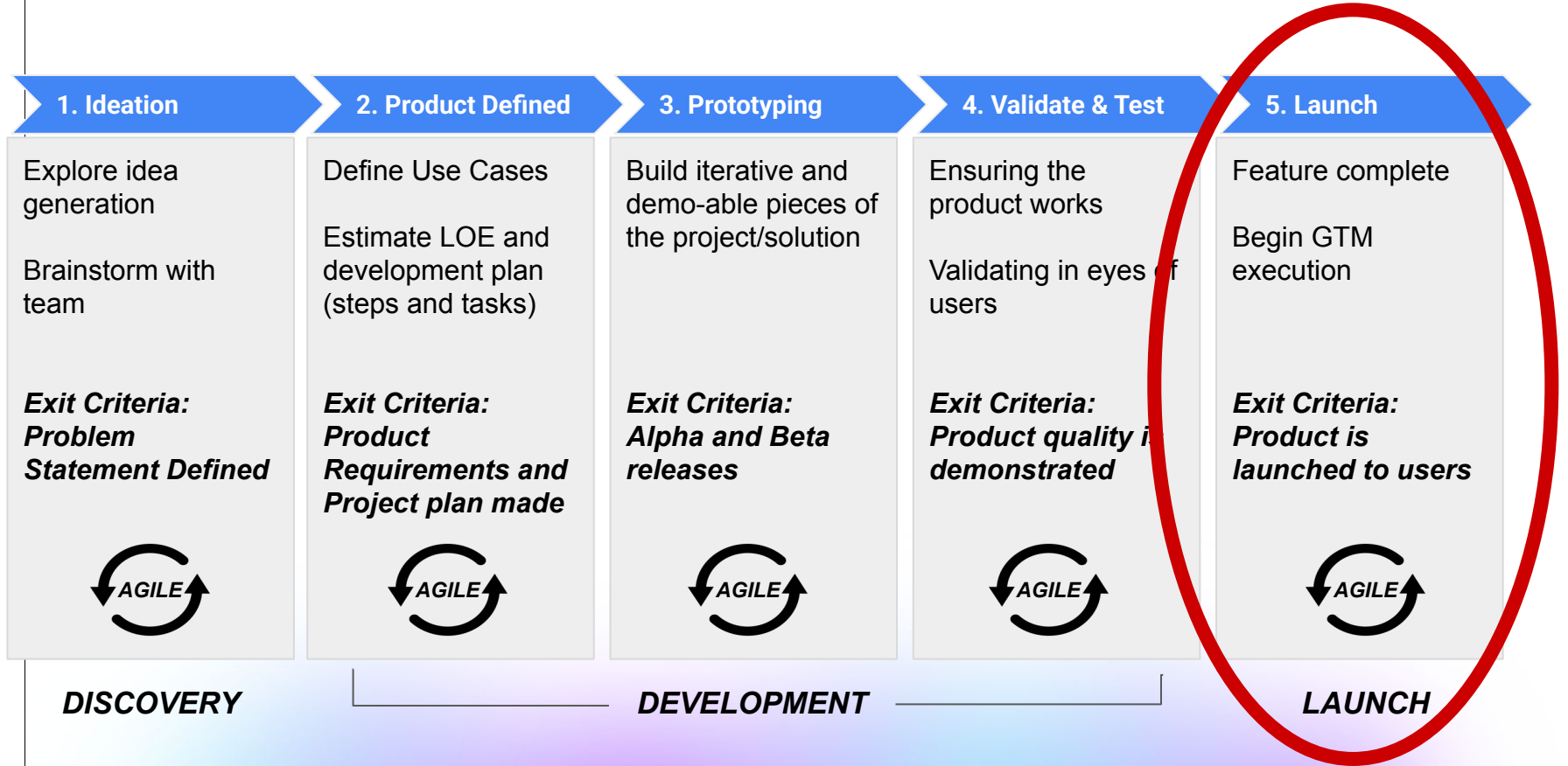
Agenda

- Demo 3
- **March Sprint**
- Deployment Technologies
- Example Industry Setup
- Upcoming Deadlines
- Deployment Tutorials



Month	Expected Status	Monthly Focus	Deliverables
October	- Project selected & approved by instructors	- Begin technical investigations (services, apis, programming language, etc) - Flesh out project functionality & requirements - Coding should start (scaffolding, ci/cd, prototyping)	- Writing: Executive Summary - Updated Gantt Chart - Teamwork survey - Writing: Technical summary - Presentation: Elevator pitch - Presentation: Project design
November	- Main technologies selected, project is well-defined - Everyone is actively coding	- Answer all questions needed to complete TDD - Lot's of coding for alpha demo	- Writing: PRD
December	- Code complete for alpha demo	- more coding for beta demo - Formalize design discussions into proper TDD	- Presentation: Alpha prototype - Writing: TDD
January	- Continued focus on project development	- continued development for demo 2 - focus on proper testing & integration	- Demo 2 (individual)
February	- Code complete for demo 2	- Refine code from a prototype into a fleshed out project -- testing, integration, polishing - continued development for demo 3 (get as close to finished as you can here)	- Presentation 4 - Demo 3 (team)
March	- Code complete for demo 3	- final code polishing to wrap up project - complete any necessary integration work - add extra features if possible	- Demo 4 (individual)
April	- Code 99% complete for final demo	- finishing touches for final project submission - ideally you are done with coding by this point	- Final Demo - Final Presentation - R&D Showcase
May			- Final package due (team website)

March Work



March Sprint TODOs

- Refine sprint goals (update github, review w/ mentor & instructors)
 - These goals should be easy – code-complete for your project!
- Sprint planning in next mentor meeting
- Peer PR review (before end of month)
 - Each student should author 1 PR
 - Each student should review 1 PR
 - You need to leave comments, so we know who reviewed which pr



Agenda

- Demo 3
- March Sprint
- **Deployment Technologies**
- Example Industry Setup
- Upcoming Deadlines
- Deployment Tutorials



Deployment Technologies

- Modern software deployment has evolved significantly
- Understanding deployment practices are important, you'll likely learn this on the job
- Today: explain a typical industry deployment pattern
- For senior design: do something much simpler



You wrote your app – what next?

- How will users access my application?
- How do I ensure my app runs consistently across different envs?
- How do I handle dependencies?
- How will I update my application without downtime?
- How will the application scale as demand grows?
- How will I monitor performance and troubleshoot issues?

Why Deployment is Challenging

- “Works on my machine” syndrome
 - Environment inconsistencies
 - Dependency management
- Configuration management
- Scaling issues
- Deployment downtime
- Rollback capabilities
- Security concerns



The olden days (before containers)

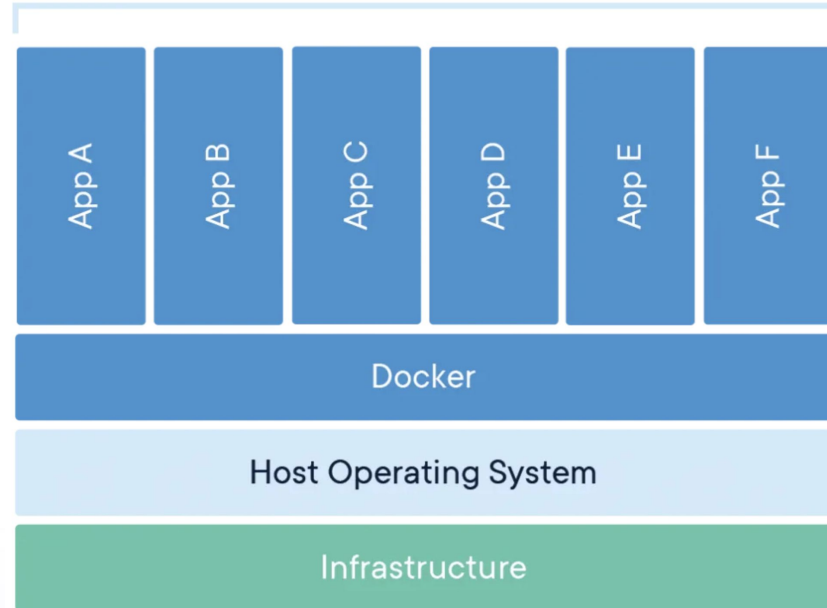
- Physical servers + manual installation and configuration
- Virtual machines – better, but still heavy overhead
- Manual deployment scripts
- “Snowflake” servers (unique & hard to replicate)
- Long provisioning times



Intro to Docker

- Lightweight containerization platform
- Packages applications with all dependencies
- Provides consistent environments
- Isolates applications from one another
- Efficient resource utilization compared to VMs
- Solves the “works on my machine” problem

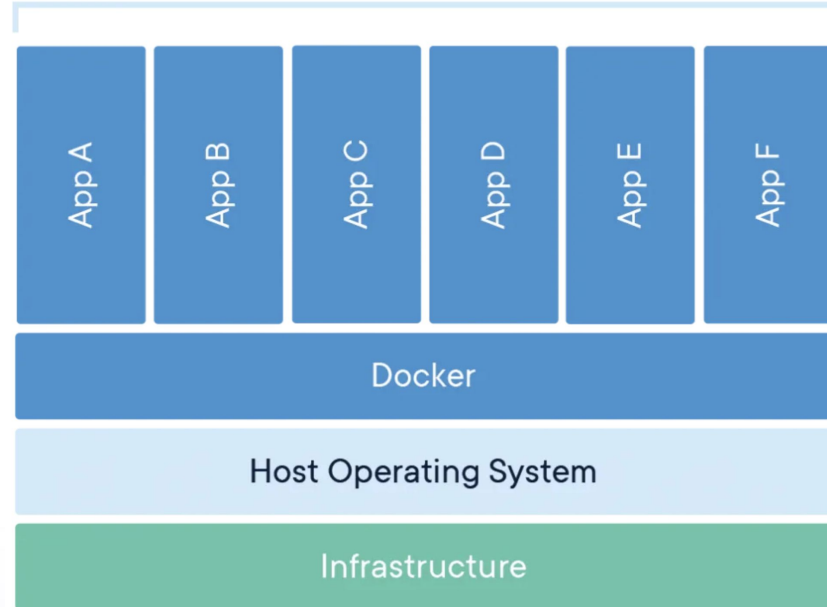
Containerized Applications



Docker Benefits

- **Consistency:** same env for development & production
- **Isolation:** applications run independently
- **Portability:** run anywhere that has docker installed
- **Efficiency:** Lightweight compared to VMs
- **Version Control:** single version representing your application

Containerized Applications



Container Orchestration Challenges

We used docker to create a container for our application, but...

- Managing multiple containers becomes complex
- Need for automated deployment, scaling, and management
- Load balancing requirements
- Service networking
- Storage management
- Rolling updates & rollbacks

Intro to Kubernetes

- Container orchestration platform
- Automates deployment, scaling, and management
- Originally developed by Google, now open source
- Abstracts away underlying infrastructure
- Enables microservices architecture at scale
- Industry standard for container orchestration



Deploying to Kubernetes – The Helm Challenge

- Complex YAML configurations
- Difficult to manage application versions
- No built-in templating
- No release management



Introduction to Helm

- Package manager for Kubernetes
- Simplifies application deployment and management
- Provides templating for Kubernetes manifests
- Manages releases and enables rollbacks
- Industry standard for Kubernetes deployments



Putting it all together

- **Docker:** Application packaging & containerization
 - Solves environment consistency
- **Kubernetes:** Container orchestration and management
 - Solves container orchestration at scale
- **Helm:** Kubernetes application deployment
 - Simplifies kubernetes deployments
- **CI/CD:** automate the whole workflow

Benefits

- Consistency across environments
- Infrastructure as code & config
- Reproducible deployments
- Scalability & resiliency



Agenda

- Demo 3
- March Sprint
- Deployment Technologies
- **Example Industry Setup**
- Upcoming Deadlines
- Deployment Tutorials



Agenda

- Demo 3
- March Sprint
- Deployment Technologies
- Example Industry Setup
- **Upcoming Deadlines**
- Deployment Tutorials



For Next Week

Weekly Focus

- Demo 3
- March Sprint Planning
 - Refine goals (**update github readme**)
 - Create sprint tickets

Mentor Meetings

- [Team]: Sprint planning + review sprint goals – work to pair down functionality if necessary

Deadlines

- [Individual]: [Teamwork survey](#) (3/9)
- [Individual]: Peer PR review (3/30)
- [Team]: End of March Sprint (3/30)
- [Team]: Demo 4 (**week of 3/31**)
- [Team]: [R&D Showcase Application](#) (4/4)
- [Team]: [R&D Showcase Poster Submission](#) (4/18)

Example R&D Showcase Poster



BikeBuddy
 Ethan Cohen | Claes Boillot | Matt Gouvin | Adham Popal
 The George Washington University
 Department of Computer Science



The Problem

Despite having some of the best bicycle infrastructure in North America, Washington still sees cars hitting and killing cyclists at an alarming rate. BikeBuddy aims to route cyclists to their destination along safe, efficient, and reliable routes.

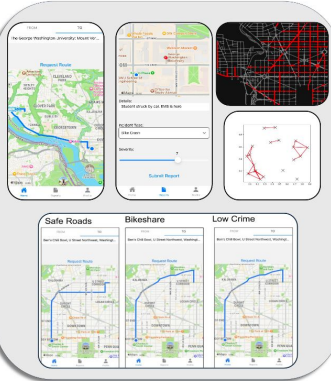
The Algorithm

Our app pulls from OpenStreetMap as well as Open Data DC, an API provided by city government that contains data about cycling infrastructure, road safety, and crime across the District. The red dots in the map to the bottom left show the majority of data points collected.

The Architecture

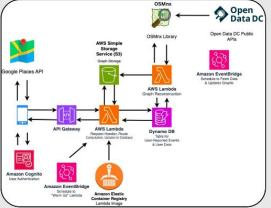
BikeBuddy fundamentally lives in an AWS Lambda function that connects our front and back ends. From the mobile app, users can request routes as well as report unsafe conditions they encounter. An AWS EventBridge scheduler refreshes the data every morning so the map is always up to date. Our graphs and lookup structures are stored in S3.

The Application



We utilize the OSMnx in Python for graph creation, specialized for maps. Construction of graph data structures works by using our ingested data to add specific bike nodes and their attributes to the base graph for D.C. We utilize a cKDTree (plot on the left) for efficient lookup of bike nodes that will be added.

Routing works by modifying the weights of edges of our custom graphs. Thus, the shortest path from the graph's point of view will generate based on the weight modifications we have put in place. This can include user preferences, user reports, and bike attributes from our data ingest.




Department of Computer Science
 School of Engineering & Applied Science



Engineering

Example R&D Showcase Poster



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC


RTX CAPSTONE COMPETITION

MAE: BRENDAN HUMPHREY, IFTAKHAR ALAM, NITHA PAULUS, RYAN RAFATI

CS (UAV): KARL SIMON, LEO PHAN, JUSTIN PARK

CS (UGV): MANUE ALAIMO, DANIA ABDALLA, KAYLA BERNE

ADVISORS: JARICK CAMMARATO, STEVEN SHOOTER, TIMOTHY WOOD



Motivation

- The drone competition integrates advanced tech like AI, vision, autonomy, and 3D printing. This year's "Water Blast" mission pushes the boundaries of unmanned vehicle technology
- Real-World Applications: By solving the "Water Blast" challenge, teams contribute to the development of drone tech for tasks like package delivery and firefighting
- The competition is a high-energy event where months of hard work culminate in showcasing creations and competing against other top teams.

Objective

- Engage students in the RTX drone competition, comprising seeker and evader challenges
- Develop algorithms for UAV navigation and object detection to meet competition requirements
- Design, prototype, and optimize water deployment and detection systems for UAVs and UGVs
- Execute precise coordination and strategic decision-making to overcome competition challenges
- Enhance students' understanding of engineering principles through hands-on application and experiential learning

Overall Approach




Figure 1: UAV Circuit Architecture

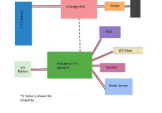


Figure 2: UGV Circuit Architecture

Implementation of Approach

Unmanned Aerial Vehicle (UAV)

- UAV is a hexacopter (6 propellers)
- Nvidia Jetson Orin Nano for image processing and route planning
- Flight controller equipped with three IMU sensors and a barometer
- 14.8 V 4000 mAh battery to power all UAV systems, capable of ~12 minutes flight time
- Camera – Full HD, 60 fps, 128° FOV – for ArUco marker detection
- GPS receiver for navigation
- Rangefinder for altitude measurement
- RF receiver for remote control

Water Deployment System

- Self-priming pump delivers a consistent flow rate of 8mL/second at 12v, controlled by a normally closed solenoid valve connected to the onboard computer.
- Adjustable nozzle optimizes spray pattern, while quick-connect tubing allows for fast assembly, disassembly, and future modifications.
- Custom-designed 3D-printed structures (ABS plastic) provide secure and reliable component placement on the drone frame

Unmanned Ground Vehicle (UGV)

- Basic frame with patterned holes for variable mounting
- 2 motor drive with 0.69 Nm, 294 RPM motors powered by a 7.4 V 5200 mAh battery
- Raspberry Pi 4 Model B powered by a separate 5 V battery
- ROS workspace with nodes and topics to facilitate communication between all sensors and the Pi

Water Detection System

- Hydrophobic-coated ArUco marker placed on an angled (3.5 degrees) funnel allows water to flow seamlessly into the funnel while remaining easily detectable by the sensor
- The water detection chip, angled to prevent water build-up, triggers a dual response upon water contact. First, a signal alerts the team of water detection, and second, LED lights and an alarm on the UGV visually confirm the detection

Current Design




Figure 3: Full CAD Assembly




Figure 4: Photographs of current assembly

Algorithms and Testing

UAV

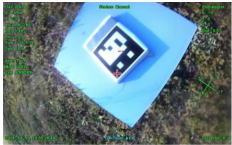


Figure 5: Camera footage from camera on UAV

- We tested our UAV at GW VSTC and UMD campus, outside DC light restricted zone
- Users can view real-time camera footage and sensor readings from the UAV
- The UAV can be controlled remotely via remote controllers and computers
- The UAV flies autonomously 3 meters in the air to scan for ArUco Markers
- Once detected a non-ally marker, the UAV flies toward it and releases water
- When all non-ally markers have been targeted, the UAV flies back to its base

UGV

- We tested our UGV following each of the challenges
- Motor controls UGV at 3 constant speeds
- When UGV has been hit, alerts with lights and buzzer and logs the hit with time
- When hit UGV will immediately disable the motors and the UGV will stop traveling
- UGV uses on board IMU sensor to make precise 90 degree turns and stay on a straight linear path

Acknowledgments

Special thanks to Jarick Cammarato, Professor Steven Shooter, Professor Timothy Wood, the GW 3D Printing Lab, the GW Machine Shop, & the Drone Lab

Example R&D Showcase Poster

CHAT GPT TRAVEL

SIDRA HUSSAIN, EVAN FRIES, COLIN RANCK, SARAH JAGERDEO

<p>PROBLEM STATEMENT</p> <p>Our project aims to bridge the gap between knowing where you want to visit and having personal travel preferences, but not knowing what activities align with those preferences or how to optimize your itinerary at your travel destination.</p> <p>Currently, many travel services create itineraries for users based on their preferences. For example, Wonderplan.ai creates itineraries based on factors such as budget, number of travelers, and activities of interest. However, these products often do not optimize their itineraries with objective measures like cost, ratings, or travel time.</p> <p>Our project addresses this gap by generating travel itineraries based on user preferences, using activity suggestions from sources like Yelp and Foursquare. We then filter these suggestions with ChatGPT to align them with personal travel preferences, while also optimizing for objective measures such as cost, ratings, and travel time.</p>	<p>SYSTEM ARCHITECTURE</p> <p>Our application is hosted on an AWS EC2 server running an ubuntu operating system. Therefore, the architecture design is all contained within AWS. The purpose of chatGPTTravel is to take a destination, user preferences and an optimization metric from the user and return an itinerary for said destination using generative AI and the corresponding optimization algorithm. First, the user must log in to the application using the system UI. We maintain and update a user database. A logged in user can view their saved itineraries and generate new itineraries through the user interface.</p> <p>The core of our application is the itinerary generation functionality. When the user enters a destination, dates, preferences and optimization metric, we acquire activities in the specified destination city from the Yelp and Foursquare APIs to yield activities with specific information, such as location, phone number, operational hours, rating and cost. We next filter the activities from the APIs based on the user's specified preferences using the openAI API and a set of natural language queries designed to yield the most accurate results. Finally, the system needs to decide which optimization algorithm to apply: fastest, cheapest, or best rated, which was provided during the user input period. The appropriate algorithm is applied to the activities to generate an itinerary for the specified number of days and then the itinerary is returned to the user.</p> <div style="text-align: center;"> <p>The diagram shows the system architecture. It starts with a 'Users Database' and 'User Interface' (UI) connected to 'AWS'. The UI sends 'Destination, Preferences, & Optimization Metric' to a central box containing a user query: 'I want to go to X, I am interested in Y, I will optimize by Z.' This query is processed by 'Activities' (represented by a speech bubble icon) and 'Restaurants' (represented by a fork and knife icon). The 'Activities' and 'Restaurants' feed into a 'Filter Activities by Preferences' step, which then feeds into a 'ChatGPT' icon. The output is a list of 'Activity 1', 'Activity 2', 'Activity 3', and 'Activity 4'. These activities are then filtered by 'Cheapest', 'Best Rated', and 'Transit Time' metrics, which are represented by icons for a dollar sign, a star, and a location pin respectively.</p> </div>	<p>COST OPTIMIZATION</p> <p>The cost optimization algorithm filters through a list of activities and then sorts them from cheapest to most expensive. The algorithm adopts a greedy strategy, iteratively choosing the cheapest restaurant for each meal: breakfast, lunch, dinner, and non-meal category. The selection process continues until the top five cheapest activities are chosen for the day within their appropriate category. The algorithm continues until there is an itinerary for each day. Cost data is obtained using an approximation of costs, based on the dollar (\$) rating system provided by Yelp and Foursquare.</p>	<p>COSTS AND RISK</p> <p>Our project has a few costs associated with it. Particularly, we pay for an API key to use chatGPT, Google Maps directions API, AWS database services and AWS Web hosting services. We also use the free Yelp and Foursquare APIs. Both of these APIs have usage limits, therefore we have created multiple free keys to cycle through in instances when the usage limit is hit.</p>	
	<p>RATINGS OPTIMIZATION</p> <p>Initially, the ratings optimization algorithm filters through a list of activities and selects clusters of top-rated restaurants with ratings above 4 stars for breakfast, lunch, and dinner. The algorithm adopts a greedy strategy, iteratively choosing the highest-rated restaurant within the specified cluster for each meal: breakfast, lunch, and dinner and non-meal category. The selection process continues until an itinerary is created for each day. We obtain rating information from the Yelp or Foursquare API when the activity is initially generated.</p>	<p>FUTURE WORK</p> <p>In the future, we hope to improve our project in a few ways. First, we would like to improve the runtime of our application because it can sometimes take 5+ minutes to generate an itinerary. Additionally, we would like to add more features to improve our users' experience, such as the ability to customize generated itineraries by adding and removing to it, links to direct booking services, and better filtering of activities. Most importantly, we would like to improve our error handling when the application crashes or an API key limit is hit.</p>		
	<p>TRANSIT TIME OPTIMIZATION</p> <p>The transit time optimization algorithm adopts a greedy approach; it iteratively finds the activity that is closest to the hotel, then finds the activity that is closest to the first activity, second activity, and so on until the itinerary is completed to create an itinerary that is optimized based on transit time for each day. The transit time between two locations is found by finding the directions between two locations using the google maps api.</p>			
	<p>YELP API</p> <p>Generates restaurants located near the user's hotel address</p>	<p>FOURSQUARE API</p> <p>Generates activities near the user's hotel address</p>	<p>OPENAI API</p> <p>Filters the restaurants and activities based on users food and activity preferences</p>	<p>GOOGLE MAPS API</p> <p>Calculates the amount of time it takes to travel between activities, restaurants and hotels</p>

Agenda

- Demo 3
- March Sprint
- Deployment Technologies
- Example Industry Setup
- Upcoming Deadlines
- **Deployment Tutorials**



Deployment Tutorials

- **EC2**: Setup a VM in aws & run your code like you would on your computer
- **Elastic Beanstalk**: Provide source code, aws manages the platform and deploys it
 - <https://www.youtube.com/watch?v=FaKlKCicyKQ>
- **Amplify***: Provide source code/link with github, aws manages deployment. Primarily focused towards frontend devs
- **App Runner***: Provide a docker image, deploys without kubernetes

*Cannot be completed in the aws academy learner lab