360° Webcams for Zoos and Aquariums

Final Report

Nathan Cool, Ian Jamieson, Zach Newton, TJ Yacoub, Alan Negrete, Sam Abdeltawab

Table of Contents

List of Figures

List of Tables

General Information

Team Details

- **ID:** sddec18-12
- **Email:** sddec18-12@iastate.edu
- **Website:** sddec18-12.sd.ece.iastate.edu

Team Members

Name	Roles
Nathan Cool	Front-End Development, Project Manager, Webmaster
Ian Jamieson	Computer Vision Development, Graphics Lead
Zachary Newton	Front-End Development Lead, Scrum Master, QA
Tarek (TJ) Yacoub	Microservices Development Lead, QA
Alan Negrete	Front-End Development, Microservices Development, QA
Hosam (Sam) Abdeltawab	Embedded Development
Dr. Henry Duwe	Advisor
Christopher (Chris) James	Client (True 360)

Part 1 - Introduction

Acknowledgments

We would like to acknowledge our client, Christopher James, and our advisor, Dr. Henry Duwe, for their contributions to our project. Chris provided our team with embedded boards and 360° webcams and funded all monthly expenses associated with project development. He also attended team meetings in order to provide feedback on our progress. Dr. Duwe attended team meetings and provided our team with invaluable technical advice and feedback on our progress.

Problem/Need Statement

True 360 is an Ames-based startup founded by ISU business undergraduate Christopher James. True 360's mission is to provide zoos and aquariums with access to immersive 360° digital experiences. Our project, *360° Webcams for Zoos and Aquariums*, is the first phase necessary to bring True 360's mission to reality.

With our project, True 360 hopes to provide solutions to three major problems/needs. First, zoos and aquariums do not have access to an easy-to-use 360° webcam solution for their exhibits. While there *are* zoos and aquariums that currently use webcams (some with 360° capabilities) for various purposes, no solution currently exists to act as a centralized (remote) connection and control hub for all of their webcams. Second, zoos and aquariums are always looking for ways to boost their social media presence and improve their marketing techniques, both to attract new visitors and improve public awareness of issues such as animal health and wildlife conservation. Finally, zookeepers, animal health professionals, and other staff members are currently required to visit each exhibit on a regular basis in order to monitor the animals. This process is time-consuming, which means staff members have less time to focus on other important responsibilities.

Our team—in collaboration with True 360—aims to provide solutions to these problems/needs. By developing a system that allows zoo and aquarium staff members to connect to and remotely control multiple 360° webcams, the necessity of physically interacting with each webcam on a regular basis will be removed. The system will allow for webcams to be installed in both above-ground (indoor and outdoor) and underwater exhibits and controlled via a central web application. Conveniently, this application will help us solve the aforementioned marketing need and animal monitoring problem. With our system, zoos and aquariums will be able to constantly capture 360° ; of animals, which can be extracted for use in educational live streams, promotional content, monitoring animal activity, and even archival purposes.

Intended Users and Uses

We must take into account three user perspectives and their respective uses for the system:

The Zoo (or Aquarium) Marketing Team

The marketing team's role is to promote the facility through various social media outlets. It is essential that our system takes into account their need for easy access to all footage/clips captured by the system, as well as a way to extract the most useful content for promotional uses.

The Zoo (or Aquarium) Curators

The zoo curators are responsible for monitoring and caring for the animals and their exhibits. Knowing the reality of staffing limitations, curators are always on the move in order to keep track of all of the animals. As a result, our system must focus on providing easy access to captured footage, particularly when it involves the presence of animals. That way, in an emergency or undesirable circumstance, curators can review archived footage in order to identify the root cause of an issue.

The Zoo (or Aquarium) IT Department

Marketing team members, curators, and other staff members are required to focus on their own jobs and cannot get caught up in the technical aspects of the tools they rely on every day. As a result, IT staff members are vitally important to the successful adoption of our system. Thus, we must take into account the installation, configuration, and long-term maintenance requirements necessary to keep the system up and running. The system must provide remote access to commonly used functionality, as well as minimize the complexity associated with providing technical support.

Market Survey

Per the Final Report assignment requirements, details regarding market survey information and related products/work have been moved to Section 5 (Related Work), which can be found below or by consulting the Table of Contents.

General Assumptions and Limitations

Assumptions

Our system *will* be responsible for:

- Webcam management
- Video archiving (via a connected cloud-based service)
- User management (multiple user types)

Our system *will not* be responsible for:

- Financial transactions (facilitation, record-keeping)
- Communication-related functionalities
- Stream management

Technical Assumptions:

- Webcams will be powered via standard 120V wall outlets
- Zoos/Aquariums using the system provide wired/wireless Internet access within the facility
- IT department staff members will set up the cameras

Limitations

Our system will assume the following limitations:

- Reliance on manufacturer-provided APIs and SDKs for interacting with webcams
- Limited connection mediums to webcams via WiFi, Ethernet, or USB
- Potential bottlenecks when collecting high-resolution footage (1080p, 4K) due to server storage cost/space and Internet bandwidth at zoos and aquariums

Part 2 - Design

Concept Description

The general design concept for our system revolves around the development of four primary software components: front-end, back-end, embedded, and computer vision. The back-end is comprised of a microservices architecture that various AWS components that help integrate all of the other parts of the project. The front-end client allows the zoos to view and download clips, as well as control the cameras. The embedded program works with the camera and controls how the camera interacts with the microservices and the front-end. The computer vision component was built to lower the AWS cost and improve the quality of videos displayed on the website.

Functional Requirements

Status Reference

- COM: Completed
- DEP_1: Deprecated (no longer required)
- DEP_2: Deprecated (no longer required, per client's request)
- INP: In progress
- BLOG: In backlog (not started)

Modules

- APP: Web application
- CV: Computer vision
- EMB: Embedded board + camera
- MIC: Microservices
- SYS: Combination of multiple/all system modules

Table 1: Functional Requirements

Listing	Status	Module(s)	Description
FR_01	DEP_2	SYS	Stream video in various resolutions.
FR_02	DEP_2	SYS	Capture photos in various resolutions.
FR_03	COM	SYS	Start and stop recordings.
FR_04	DEP_2	SYS	Start and stop streams.
FR_05	COM	SYS	View current camera status.
FR_06	COM	APP	Staff must be able to view archived footage.
FR_07	COM	APP	Staff must be able to download archived clips.
FR_08	COM	SYS	Machine learning to detect activity.
FR_09	DEP_2	SYS	Alerts to staff for abnormalities.
FR_10	DEP_2	SYS	See animal activity over a period of time.
FR_11	BLOG	SYS	Staff must be able to embed logos on streams/recordings.
FR_12	DEP_2	SYS	Sponsors can rent allotted times for certain streams.

Non-Functional Requirements

Status Reference

- COM: Completed
- DEP_1: Deprecated (no longer required)
- DEP_2: Deprecated (no longer required, per client's request)
- INP: In progress
- BLOG: In backlog (not started)

Table 2: Non-Functional Requirements

Listing	Status	Description
NFR_00	COM	Webcams must be able to operate in various types of environments.
NFR_01	COM	Persistent data storage with automatic periodic backups.
NFR_02	COM	The system must scale with user growth and usage.
NFR_03	DEP_1	The system must be remotely maintained via Snappy for Ubuntu Core.

Constraints

Status Reference

- OK: Constraint is still in effect.
- DEP_1: Deprecated (no longer required)
- DEP_2: Deprecated (no longer required, per client's request)

Table 3: Constraints

Listing	Status	Description
C_01	OK	The system must be able to function in different environments with varying temperatures.
C_02	OK	Many of the webcam options have limited APIs/SDKs to work with - if any at all.
C_03	DEP_2	A working system requires high bandwidth to ensure stable and high-quality streams.
C_04	OK	Very limited development time with a physical embedded device and Garmin camera.
C_05	DEP_2	Each zoo will have a local server and storage for archiving footage.

Operating Environment

The major physical components of the system include one or more 360° webcams and any central computing hardware necessary for connecting/controlling the webcams and managing the captured footage. In order for the system to maintain a normal operational state, there are many environmental factors which must be taken into account.

Zoos and aquariums will place webcams in a variety of locations, including both above-ground (indoor *and* outdoor) and underwater exhibits. Webcams placed above-ground and outdoors must be able to operate while exposed to the local weather. Webcams placed above-ground and indoors must be able to operate while in glass-covered exhibits or in areas where visitors' devices may tax the network (if public/staff network traffic is not separated). Exposing electronics to underwater conditions inevitably poses many potential problems which will have to be addressed (e.g., cable exposure, underwater webcam temperature regulation).

Regardless of where webcams are placed, there are many universal environmental factors which must be considered in order to maintain a normal operational state. The webcams must have constant access to ample power, a stable and strong (wired or wireless) Internet connection, and a case to physically protect the webcam unit. Fortunately, our client will be responsible for the process of fabricating a case to fit the webcam model used in the system.

One of the most important environmental factors which must be considered is the presence of animals. The webcam unit must be durable enough to withstand the impact and pressure of strikes, bumps, bites, and other animal interactions. Any cables (e.g., power, Ethernet, USB) connected to the webcams must be installed in such a way as to remain out of the sight/reach of the animals.

In addition to the webcams, we must take into consideration the computing/storage hardware and the software (web application) components and their respective environments. Any computers (desktops, servers) used must be stored in a secure indoor and temperature-regulated environment in order to avoid issues such as overheating and unauthorized access. The web application must be developed with digital security in mind.

Overall, these environmental factors represent our basic understanding of and assumptions about how zoos and aquariums will use the system. They do not encapsulate every possible factor, as there will inevitably be some variance between facilities/users.

Risk Identification and Management

Potential Risk: A team member is considering dropping the course.
Management: Maintain open communication amongst team members in order to address and resolve any issues/impediments in a supportive manner. In the event that the situation occurs, communicate with course instructors, our faculty advisor, and the team member in order to mitigate any potential impacts on progress.

Potential Risk: The client loses interest in the project or the project is discontinued.
Management: Team members should think about contingency plans and/or alternative projects. In the event that the situation occurs, immediately contact course instructors and our faculty advisor to determine how to proceed.

Potential Risk: A major hardware component is damaged.
Management: Team members should take care to ensure proper storage, transportation, and usage of all hardware component. In the event that the situation occurs, document it immediately and notify the owner of the damaged component in order to determine whether it can be fixed/replaced under warranty or if complete replacement is feasible.

Potential Risk: Members of the team deviate from the project schedule or team productivity decreases.
Management: Ensure that everyone is on the task at all times by utilizing our communication and planning tools and procedures. In the event that the situation occurs, call a meeting to discuss the deviation and identify the necessary steps to return to the proper project schedule (and mitigate any potential losses).

Potential Risk: Research efforts are consuming significant time.
Management: Research is inevitably part of the learning process; however, if research efforts are not leading to the desired results, consider reaching out to other members of the team, pulling in an external expert/consultant (a service offered by our client), or reaching out to our faculty advisor or another faculty member for support.

Potential Risk: Sudden changes in requirements (scope creep)
Management: In order to prevent scope creep, the best course of action is to define a concrete set of requirements and obtain written agreement (signatures) from all parties. In doing so, potential instances of scope creep can be mitigated.

Resource Requirements

***Note: All resources listed in this section were acquired via client funding.

- Embedded Program
    - Raspberry Pi - $35
    - UP Board - $100
- Cameras
    - Garmin VIRB 360 - $800
    - Ricoh Theta - $400

Functional Decomposition

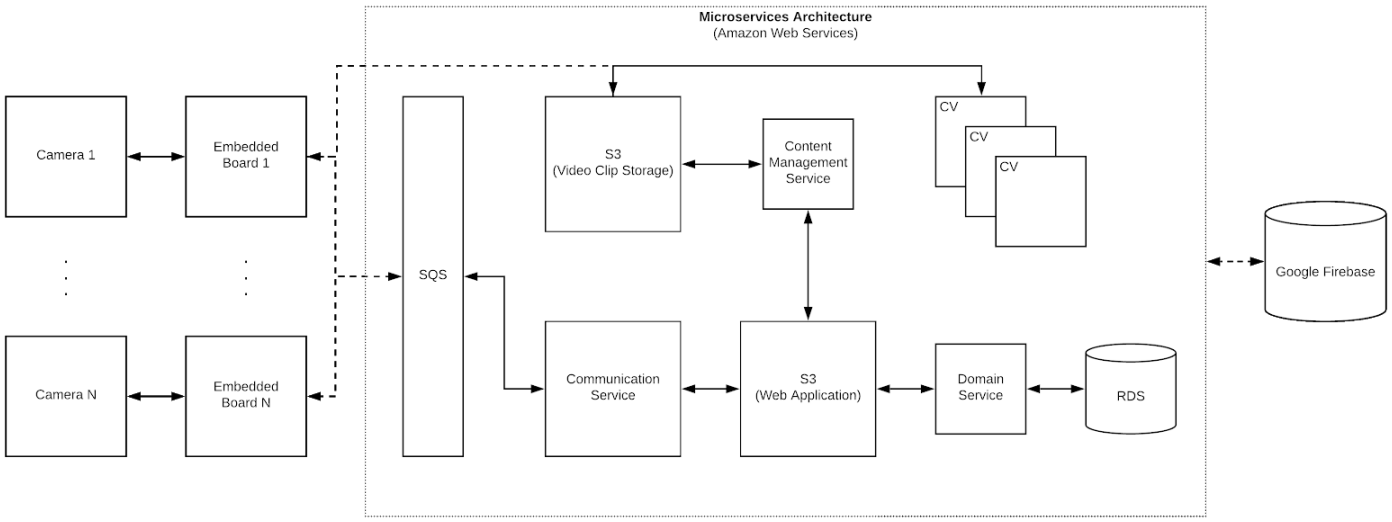

Figure 1: System Block Diagram

Standards

During the design, development, testing, and maintenance phases, our team must adhere to standards in categories such as protocols, design patterns, and documentation. By adhering to these standards, we can ensure that our system will meet industry expectations of safety, security, maintenance, and professionalism.

Many of the project's hardware and software components will be developed around standard communication and connectivity protocols such as IEEE 802 MAC (Medium Access Control), IEEE 802.3 (Wired Ethernet), and IEEE 802.11 (Wireless Ethernet). Each of these protocols is incredibly important to our project, so it's also important that we follow the standards surrounding them. Our team has considered many of these standards and how they affect our project.

When implementing the system's software components, our team will follow the code modularity and readability practices which are relevant to the chosen programming languages. Being that our project utilizes various third-party libraries, APIs, and SDKs, we will follow the coding standards and other

official specifications for those outside resources. Our code is also available on Gitlab so that if a change occurs we can approve or decline it. This is also how we can stay up to date with what everyone is working on.

In regard to our documentation, we have strived to ensure that all our documents have roughly the same layout and are well organized. Our team has organized the documents in such a way that if someone else were to pick up the project, they could easily get caught up within a few hours. This is entirely by design, due to one of our potential risks - a team member considering dropping the course. Should we have a team member that decides to drop the course, and we get a new team member, they will be able to get caught up and contribute efficiently.

Standard Protocols

Programming Practices

- Modular, well-commented code
- Utilize change management software such as Git

Documentation Standards

- A common theme for all documents
- Documented code and Git history
- All documents stored in a central location, available to everyone on the team

Ethical Practices

Based on our work so far, we've been able to follow the ethical practices for IEEE. Our team has done very well in terms of working ethically. Some examples of this are:

- We communicate frequently and discuss issues as they arise,
- Communicate any major issues with the client as soon as possible,
- Ensure we're all aware of safety issues outlined earlier in the Project Plan.

IEEE Standards and True 360

Throughout our working relationship with True 360, we've learned a lot about IEEE standards and how they apply to our project. We've identified a few specific areas we're applying IEEE Standards:

Software Quality Assurance Processes

- Defined test cases
- Defined validation scenarios
- Change management software which ensures that when code the is changed, multiple people much approve of the changes

Systems and Software Lifecycle management

- Avoiding short-term solutions and having a future-thinking mindset
- Writing quality documentation and code comments so whoever works on the project next will be able to catch up quickly

- Understanding that technology is fairly new and working through a lot of technical issues

Requirements for Managers about Documentation

- Descriptive documentation for both the Project Plan and Design Document
- Regularly discuss plans and team updates
- Document weekly status reports

Deliverables

360° Webcam System

Our team is delivering a completed system in December 2018. This system consists of all the pieces of hardware and software components integrated together to work as a functioning unit. This system as a whole will capture of all the completed Non-Functional Requirements and Functional requirements listed on Part 2 - Design section. It also ensures that any active constraints have been met. The system is composed of various Microservices that integrate together to meet those requirements. These Microservices and components will be discussed in further detail below.

Web Application

The web application is the main entry point into the system for the most part. It provides a User Interface into the system, that both Zoo Staff members and True 360 Staff members can access. The delivered Web Application will have User Registration that consists of an Invitation System for adding new users, as requested by our client, in order to maintain control over users of the application. The application will also provide an interface for administrative parts of the system, such as creating and updating zoos, cameras, devices, and video recordings.

Computer Vision

The computer vision aspect of the project has changed a lot during the fall semester. This semester was devoted to research and gathering a better understanding of what our steps need to be. We created a foundational program that detects the motion in a given video clip, then saves that video for the Zoo Staff to be able to view. This has allowed the zoo staff to reduce the number of hours that they might have needed to pour through all of the original videos. This foundational system lays the path out to many important Machine Learning use cases.

Part 3 - Implementation

Design Decisions

Local Implementation

Early in the project, our design was focused on a local implementation. This was mostly due to our client's requesting this way. However, we quickly learned that this conflicted with another requirement, perhaps more valuable requirement, scalability. Our team agreed that it would be feasible to create a scalable system, with a local server per zoo constraint. The idea of scalability is what consistently drove our design into a more cloud-based solution as the project progressed. In addition, we agreed that such a system design may have many technical issues when being set up, as setting up a server at each zoo would be a manual task, prone to errors. Having a cloud-based solution would mean that adding the system to a zoo would be mostly an automated or software task, aside from installing the cameras at exhibits.

Web Application to Camera Communication

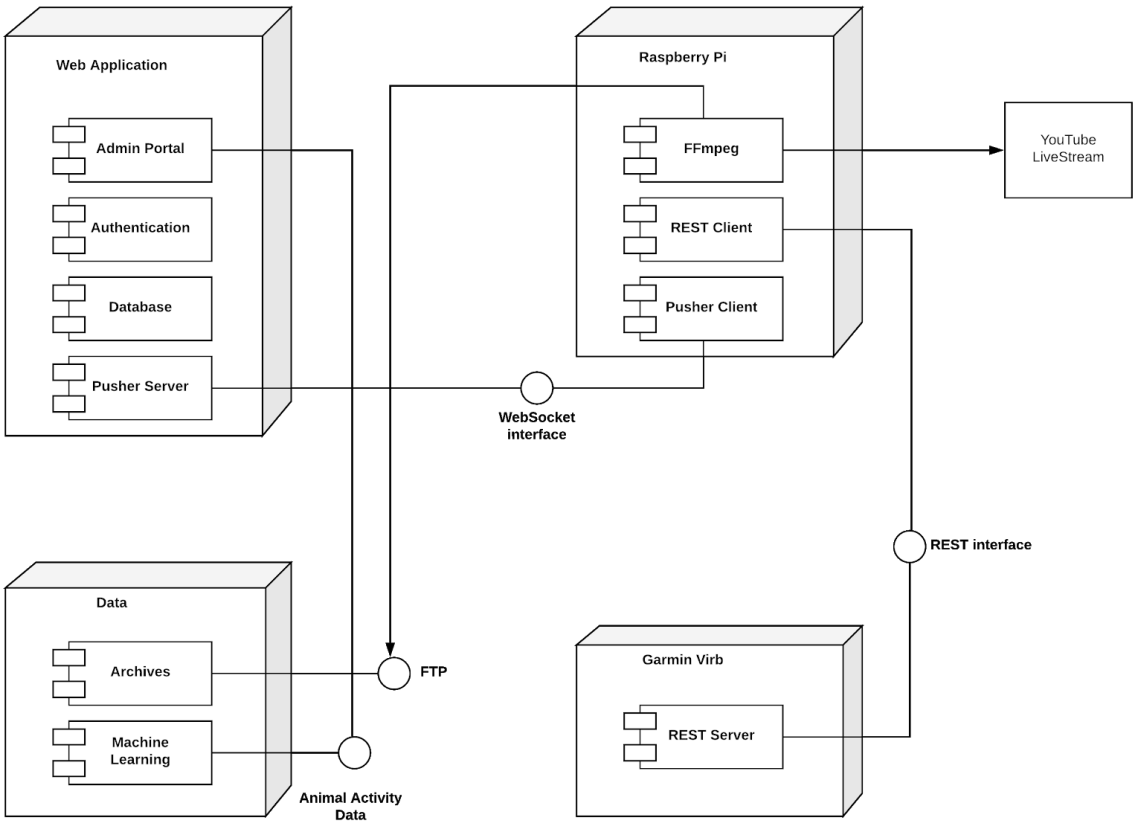

Figure 2: Initial Interface Design

One big design decision change we had to make was how our system handles the communication between the Web Application all the way to each embedded device that interfaces with its own camera. The above figure shows the first feasible design approach that we came up with late in the spring 2018 semester. The key service that will, later on, go to be replaced is the Pusher Server and the Web Socket interface that is provided to the embedded board - which also later changed, see the I/O Design section for more details. In summary, because of the changes to the overall design to a more cloud-based approach, it was simply a better design choice to go with one main provider for our services, which was AWS, instead of having various services scattered throughout different providers. This allowed the system to communicate better and took on the cost-saving advantages that AWS provides when internal services communicate. So the Pusher Server was replaced with SQS and to an extent, with the combination of other microservices we've created, such as the Communication Service. This gave us a more customizable platform to work on and more control over the end-to-end system communication.

Computer Vision

In the spring semester, we discussed the idea of using Machine Learning to implement an animal activity monitor. The original idea was to use Machine Learning techniques to create a foundation for various use cases that involve animal activity monitoring. We discussed implementing a few different use cases such as alerts to zookeepers if there was an abnormality in the animal's behavior, animal health tracking, and trash detection.

When planning a Machine Learning/Computer Vision implementation, it's incredibly important to keep the end result in mind. One of the largest reasons for failure in Machine Learning projects is forgetting the end result and building the wrong product. While this wasn't the case for our project, but as the client learned more from his clients, we were able to get a better idea of what needed to be accomplished.

One of the things we learned from the client was that zoos aren't actually interested in animal health monitoring, as previously mentioned. Zoos prefer to physically check and care for their animals, and creating this feature wouldn't provide them with any actual value. Going from there, it was difficult to come up with specific feature ideas that would provide value to the client.

As for what to do next, we considered how we could improve back-end processes for the end user. One of the processes that we considered was how to improve the process of storing and using vast amounts of video created by the cameras. Just to give you an example, if one camera recorded the video for eight hours straight, it would likely produce around 200GB a day. This isn't exactly practical to upload, store, or view.

Our thought here was if we can't develop the original use cases, we could develop something to help reduce the amount of video we would need to store and view. We came up with the idea of creating a computer vision aspect of the project that allows us to filter out uninteresting parts of the video.

Our process was fairly straightforward, we first wanted to create an algorithm that detected motion in a given video clip. Then after we figured that out, we wanted to retrain a model to implement an image recognition model.

Work Breakdown

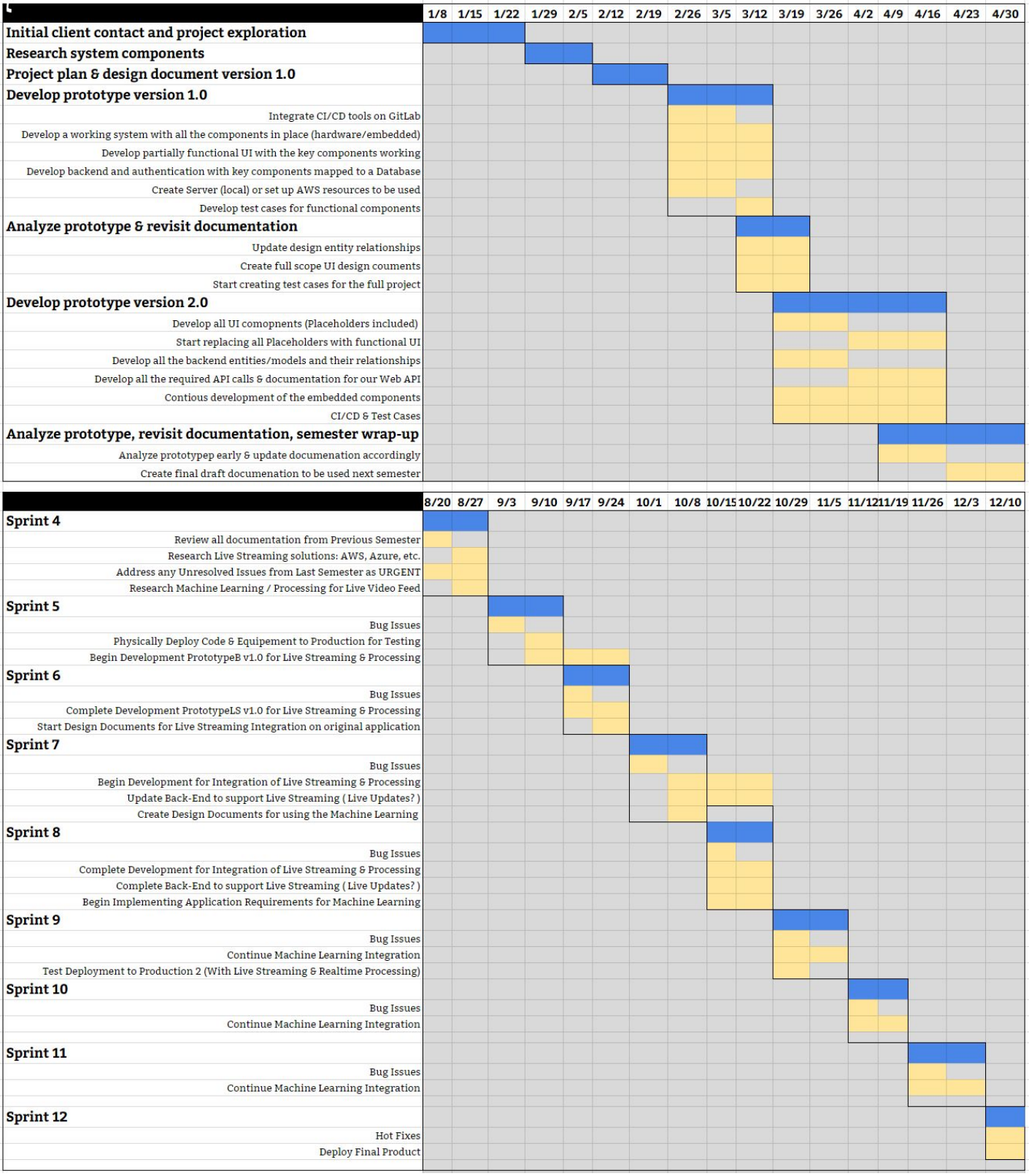

Figure 3: Work Breakdown

Part 4 - Testing

Testing Plan

General

We used GitLab continuous integration and continuous development tools for testing builds of the entire project. Any time changes were committed to a branch, our testing server or virtual machine started a build. Any pull requests also required a successful build before being merged into the master branch.

Front-End

The front-end testing process involved the use of a build server. Whenever code changes were pushed, the server allowed us to build the project and run various types of tests to validate the new features. We focused on both code-based and interaction-based tests.

Back End

Our focus was to ensure a solid foundation so that other teams could continue development. We focused on implementing an API testing framework to run tests during merges, thus ensuring that we didn't break any existing API routes during development, as well as that additional routes, were tested on the fly.

Embedded

Our embedded program testing process involved installation of the embedded program on a Raspberry Pi or other embedded board, connecting to a webcam, connecting to the microservices architecture and other system components, and completing the footage capture process.

Computer Vision

The computer vision component was tested by creating test images to build a dataset to retrain a TensorFlow image recognition model. We used a testing/training split of roughly 60% training and 40% testing, for the Machine Learning model. We were able to test the activity monitor by creating the dataset that was able to detect whether the motion was present.

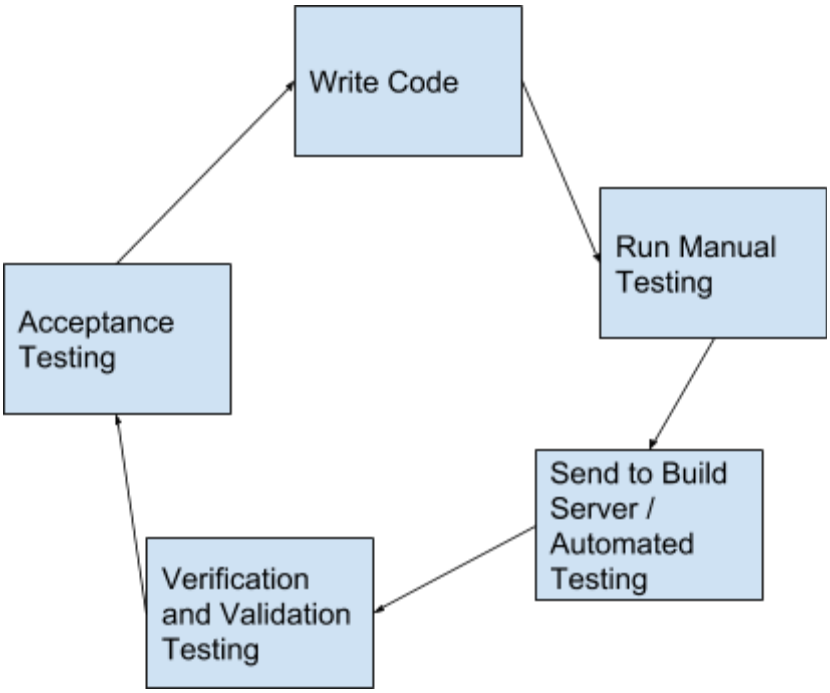

Figure 4: Process Flow Diagram

System Testing

Archiving Video Test

- Goal: Meet functional requirement, " IT staff must be able to view archived live streams"
- Verification Process
    1. Live stream and archive video for 8 hours
    2. Ensure video all video files have been stored in the server at the zoo
    3. Check if all videos are accessible from the Web App
    4. Check if a specified date & time is accessible

Outage Test

- Goal: Meet functional requirement, "IT staff should not have to physically access the devices after outages to get stream back up "
- Verification Process
    1. Disconnect & reconnect power/network from the device or camera
    2. Verify that device restarts and is listening to correct WebSocket Channel
    3. Send a trigger/command from the Web App to the device
    4. Ensure that the device picks up command and executes it

API Server Crash

- Goal: Meet non-functional requirement, "Application Services must be reliable "
- Verification Process
    1. Force restart server or kill the program
    2. NodeJS server should automatically restart
    3. Connection to Database & Pusher should be available via configurations
    4. Hit API Endpoints via Web App or Postman to verify service is back up

Part 5 - Related Work

Market Survey of Related Products

After conducting a general observational survey of several zoos and aquariums, including the San Diego Zoo and Blank Park Zoo (Des Moines), we discovered a common theme: webcams and associated software are usually utilized by zoos and aquariums for hosting low-resolution video streams on their main sites. These streams are meant to provide a simple way to get an up close and personal view of a limited selection of animals. Unfortunately, the functionality provided by these streams is limited to a simple static video pane on the web page. Thus, users and zoo/aquarium staff have no way of utilizing the streamed footage for more useful purposes.

The Rationale for Our Project

The primary argument in favor of our project is that it expands upon the capabilities of the streaming methods currently being used by zoos and aquariums. It provides a robust and intuitive user interface, higher-resolution footage capture, footage management features, archiving, and “likeability” curation using the computer vision component. As a result, the footage captured in animal exhibits can be utilized for a wider array of applications such as creating educational content, marketing material, and providing archives of animal behavior.

The Rationale for Related Products

The primary argument in favor of the implementations currently used by zoos and aquariums is fairly trivial: simplicity. Setting up a webcam and connecting it to a static webpage provides a hassle-free and maintenance-free way to add an element of interactive content to the facility’s website.

Part 6 - Summary and Concluding Remarks

Summary

Webcam setups currently used by zoos and aquariums provide limited functionality with respect to interconnectivity, footage quality, and software features. Many of the solutions that are currently on the market do not have the same 360° footage that we provide in our system. In collaboration with True 360, our team aimed to provide zoos and aquariums with an easy-to-use 360° webcam system for educational, animal health, and marketing purposes, each of which will provide new kinds of business value to zoos and aquariums. Our solution removes the need for physical interaction with each webcam and provides a wide array of software features, including centralized webcam control, 360° footage capture/archiving and live streaming, animal activity monitoring, and content curation. While the underlying technology has been around for years, our utilization of new tools to build a new 360° webcam system for zoos and aquariums will enable them to interact with their exhibits like never before.

Appendix I - Operation Manual

Getting Started

The purpose of this appendix is to guide developers in setting up the system. The system was designed with dependencies on AWS SQS as the message broker, and AWS batch as the batch processor management. As such we decided to host all our services, except authentication, in AWS. If desired the system could be deployed elsewhere, but this appendix will not cover that. Finally, CI/CD is an important part of our system, we decided to use Gitlabs builtin CI/CD service, which is provided for free up to 2000 CI minutes per group per month (https://about.gitlab.com/pricing/). The setup of CI/CD will also not be covered as this is dependent on what version control system the client will use in the future. We strongly suggest that Gitlab be used, as our CI/CD scripts will be provided, and avoiding Gitlab will require the development of new CI/CD scripts.

System Requirements

All of our services use docker to avoid any OS dependencies. Three of our services (domain-service, communication-service, content-management-service) have the same CPU units and memory requirements.

In Amazon ECS each CPU core (vCPU) is equal to 2048 CPU units. The three services at a minimum will require 256 CPU units and 512 MiB of memory. These minimum requirements were sufficient for our testing load and will require adjustments for production. We think it is preferable to find a medium between scaling and hardware specs.

AWS batch requires the configuration instance type, maximum vCPU, desired vCPU, and minimum vCPU. The fourth service, activity-monitor-service uses the optimal option for instance type. This option will choose from the latest AWS C (Compute optimized), M (General purpose), R (Memory optimized) instance families. We have decided to set the following ranges:

- Desired vCPU: 1, the number of vCPUs that each activity-monitor process will use.
- Maximum vCPU: 6, the maximum number of vCPU to create at a given point. For the following settings, this translates to a maximum of 6 activity processes at a given point of time.
- Minimum vCPU: 0, this ensures that all unused vCPU is deleted if there are no jobs in the queue.

Embedded Environment Setup

Running Embedded on the Board

To run the embedded on the board go to the scripts repo, where there will be a “raspi_setup” script that will need to be executed before running the program. This script will install all the necessary packages with the right versions.

At the end of the script, you will notice a group of export commands. These export commands need to be copied and pasted, as they won't be persisted outside the context of “raspi_setup” script.

The following is the list of the environment variables and their usage:

- S3_BUCKET_NAME: this is this name of the root bucket in AWS

- JOB_QUEUE: this is the name of the AWS Batch job queue
- JOB_DEFINITION: this is the name of the AWS Batch job definition
- AWS_DEFAULT_REGION: AWS region
- AWS_ACCESS_KEY_ID: the programmatic user key id
- AWS_SECRET_ACCESS_KEY_KEY: the programmatic user secret
- CAMERA: camera interface to use in the embedded program
    - For value (1) the embedded program will use the Garmin Virb class
    - For value (2) the embedded program will use the Mock class
- STREAMER: streamer interface to use in the embedded program
    - For value (1) the embedded program will use the FFMPEG interface
- CONNECTION:
    - For value (1) the embedded program will use the Pusher interface
    - For value (2) the embedded program will use the SQS interface

Running Embedded outside Board

There exist a “run_local” shell script that is in the embedded root dir, that will build and run the Docker image. We decided to use Docker because it is a lot easier to test since all the packages will be deleted when the Docker image is destroyed, thus not cluttering the developer's computer. As such before you use the “run_local” script, you will need to download Docker to run the script successfully.

Embedded Board Registration

Regardless of where the embedded program is running (outside or inside the board), if the camera is not registered in domain-service the program will log its registration AWS queue URL, this URL includes the camera unique id as such (https://queue.amazonaws.com/456445135125/registration_**00000000-0000-0000-0000-0242ac110002**_queue) where the bolded content is the unique id. This unique id would then be used on the web application to register the camera to a zoo and exhibit.

The zoo and camera id will be persisted in true360/config/config.ini in the embedded directory. At the moment, the only way to de-register a camera would be to set all values in the config.ini file to None.

Local Environment Setup

Running one of domain-service, communication-service, content-management-service locally requires that frontend also is run locally. The front-end proxy should be configured to point to one or all the service running locally. These routes are configured in src/lib/domain/index.js under the frontend directory, where there exist three different variables for each service, domainEndpoint, communicationsEndpoint, contentManagementEndpoint.

All services have a run_local.sh and an entry.app shell scripts provided in the root of the project to help developers run those service quickly and correctly. The run_local.sh will first build the docker image, and then run the image and set the environment to local. Entry.app will be executed after the docker container is running, ensuring that overriding variables will persist.

Since all services are required to be running in AWS, the entry.app will export the following environment variables if ENV is “local” (ENV is set to “local” in run_local.sh):

- AWS_DEFAULT_REGION: This is set to “us-east-1” since all the other running services running in “us-east-1”.
- AWS_ACCESS_KEY_ID: AWS user access key id.
- AWS_SECRET_ACCESS_KEY: AWS user secret key.

When running in AWS the above three variables are set automatically. Moreover, all four services use the same AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, which belong to the AWS user “LocalUserTest”. This user only has programmatic access only to AWS and is allowed to use all the services that we need in AWS (SQS, S3, …).

In the case that these keys are lost, or developers want to create a user for each service instead of sharing one user,

1. Refer to https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html on how to create a user, make sure it has programmatic access only to AWS.
2. Refer to https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_change-permissions.html on how to change and add user permissions.
3. Refer to https://aws.amazon.com/blogs/security/wheres-my-secret-access-key on how to retrieve user key and secret
4. Go to entry.app in the service that you want to run locally, and change AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY with the new key and secret that were retrieved in step 3.

Running the Frontend requires less setup than running the other services, follow the README in docs/README.md under the frontend directory.

Cloud Environment Setup

We made sure that running the environment on the cloud should be an easy process. Thus, we have decided to use Terraform. Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers which includes AWS. In our Gitlab group for true360, there is a repo designated for infrastructure, all that is required from the developer is to run the build script, and the infrastructure will be automatically set up.

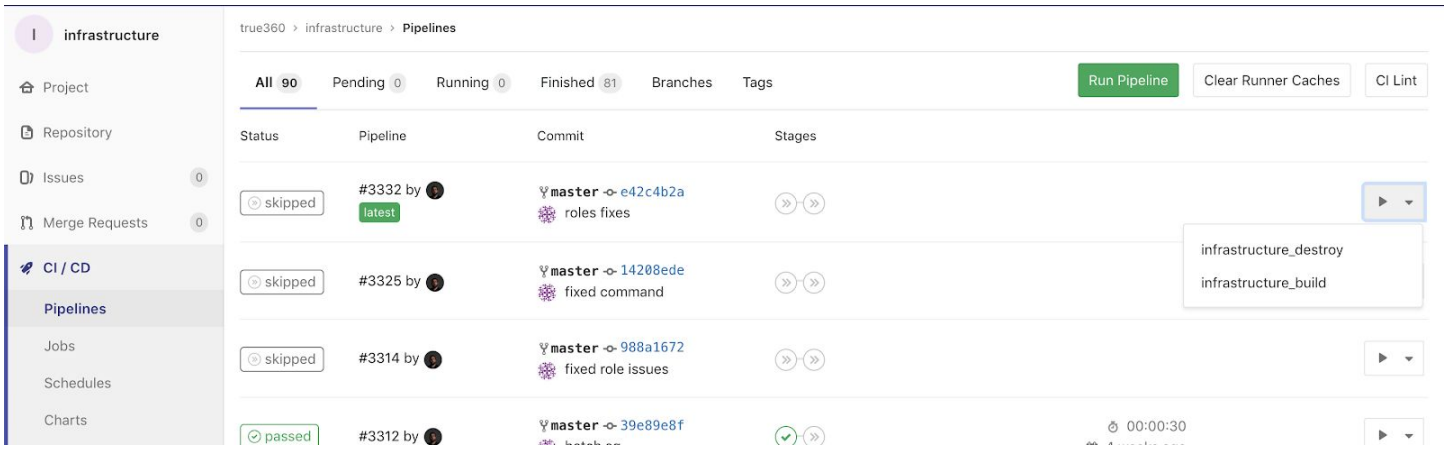

Figure 5: Gitlab CI/CD Pipelines

Is indicated in the picture, below the play button, there are two commands infrastructure_destroy that will destroy all the AWS infrastructure, and infrastructure_build to build all AWS infrastructure.

This process depends on some variables that need to be configured in Gitlab variables.

1. Head to the true360 Gitlab group
2. Go to setting
3. Go to CI/CD
4. Expand the variables section
5. Ensure you have the following variables

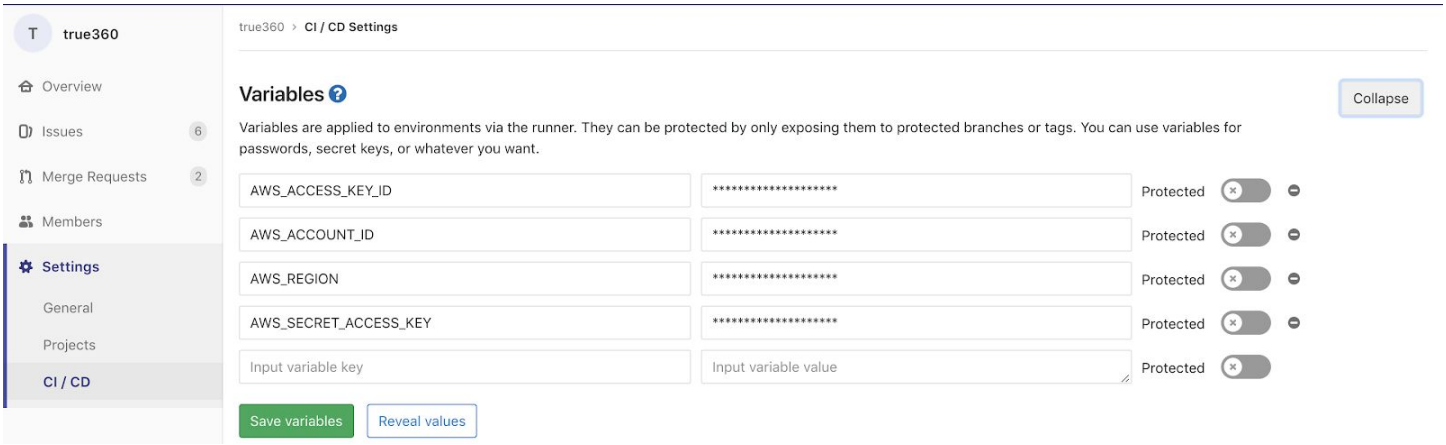

Figure 6: Gitlab Group CI/CD Variables

All the Gitlab continuous integration scripts depend on the variables above.

The sequence of building the project from nil is the following:

1. Execute Infrastructure **infrastructure_build** CI
2. Execute Frontend CI
3. Execute **ALL** other services CI (no order necessary)

The sequence of destroying the project from a running state is the following:

1. Execute Infrastructure **infrastructure_destroy** CI
2. Go to https://console.aws.amazon.com/ecs/home#/repositories and delete all the repositories

The main entry point of the infrastructure code is terraform/main.tf under infrastructure directory. The current terraform script will create all services under public subnets.

The resulting architecture is pictured in the diagram below.

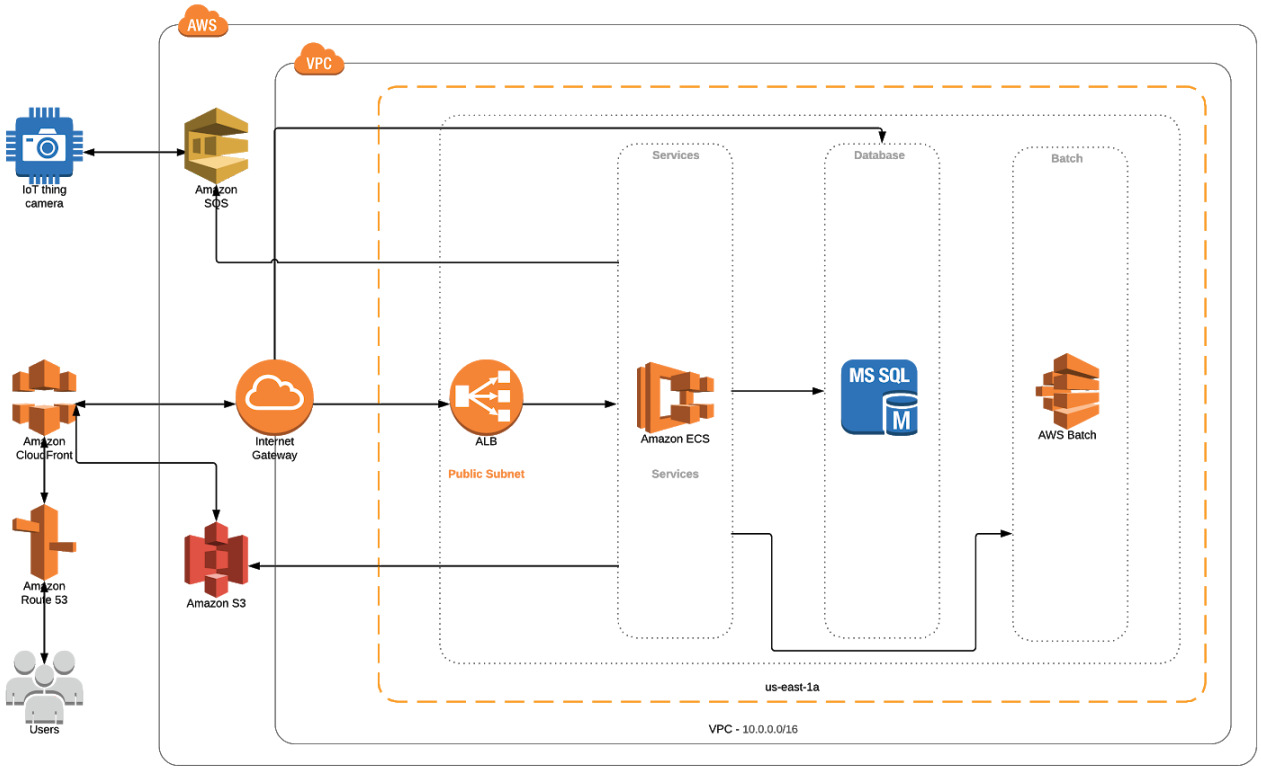

Figure 7: AWS Cloud Architecture

Accessing the Source Code

Currently, all of the Source Code is located in a GitLab Group that contains all of the repositories for the entire project. Valid authentication to git.ece.iastate.edu is required to be able to view the source code for the project.

The link for the group is https://git.ece.iastate.edu/true360. In addition, our client will also be provided with zipped & up to date source code for all projects.

Accessing AWS Services

Relevant resources in AWS:

- https://console.aws.amazon.com/ecs/home: contains all the clusters in the system. A cluster is a collection of similar services. There should be four clusters,
    - True360-dev-domain-service-cluster
    - true360-dev-activity-monitor-batch_Batch_4c025800-0202-355f-9f90-0765a37560d1 (managed by AWS, therefore, this cluster is empty)
    - True360-dev-comm-service-cluster
    - True360-dev-content-mgmt-cluster

  Each cluster only contains one running service for testing
- https://console.aws.amazon.com/ec2/v2/home#LoadBalancers: true360 load balancers
- https://console.aws.amazon.com/ec2/v2/home#TargetGroups: Running services status

- https://s3.console.aws.amazon.com/s3/home: true360 storage that contains the following:
    - True360 WebApp client
    - True360 Archives
    - Terraform remote state
- https://console.aws.amazon.com/rds/home: relational databases page

Associated Costs

ECS:

For 0.25 vCPU and a memory configuration of the following 0.5GB, 1GB, and 2GB.

Total vCPU charges = # of Tasks x # vCPUs x price per CPU-second x CPU duration per day (seconds) x # of days

- # of Tasks: 3 tasks, 1 for each service
- # vCPUs: 0.250 x 3 services
- price per CPU-second: 0.00001406
- CPU duration per day (seconds): full day

Total vCPU charges = 3 x 0.75 x 0.00001406 x 86,400 x 30 = $81.99792

RDS:

For a db.t2.micro instance, the cost per hour is $0.017

S3:

First 50 TB / Month $0.023 per GB
Next 450 TB / Month $0.022 per GB
Over 500 TB / Month $0.021 per GB

Load Balancer:

$0.0225 per Application Load Balancer-hour (or partial hour)
$0.008 per LCU-hour (or partial hour)

For more information: https://aws.amazon.com/elasticloadbalancing/pricing/

Our average monthly cost has been ~100$ per month for the months November 2018 and October 2018.

Troubleshooting

For troubleshooting services running on the cloud,

1. Go to cloud watch page (https://console.aws.amazon.com/cloudwatch/home)
2. Select logs on the left-hand side
3. Select the relevant Log Group

After selecting “Log Group” logs are bundled according to a date and time, and when clicking on that particular log, all log output is shown.

For troubleshooting service running locally, log output is always outputted to the terminal. Therefore, no extra steps are necessary to troubleshoot locally.

Known Bugs and Other Issues

There is no ability to edit some existing or user-created information. For example, once a User has been invited into the application, via the user invitation process, the created user currently does not have a “User Profile” page. This means that an existing user is not able to go back and change any of his profile information.

In our activity-monitor-service, we were only able to encode the processed videos in MPEG-4 codec. Unfortunately, Chrome does not support the MPEG-4 codec, but we were able to view the processed videos on Safari. As a workaround, we added a download button for each video that will enable users to download the footage to view it on there client machine.

Appendix II - Alternative Designs

Alternative System Design I (CPRE 491): Local Implementation

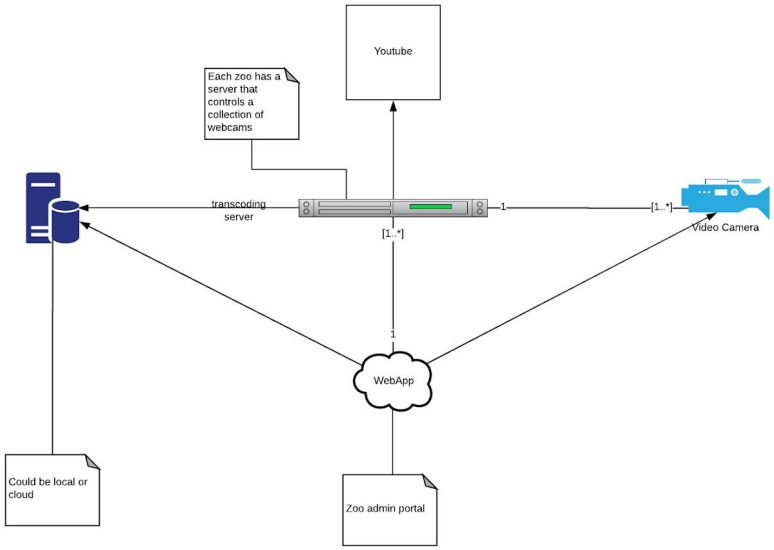

Figure 8: Localized Design Approach

Alternative System Design II (CPRE 491): Initial Cloud Architecture

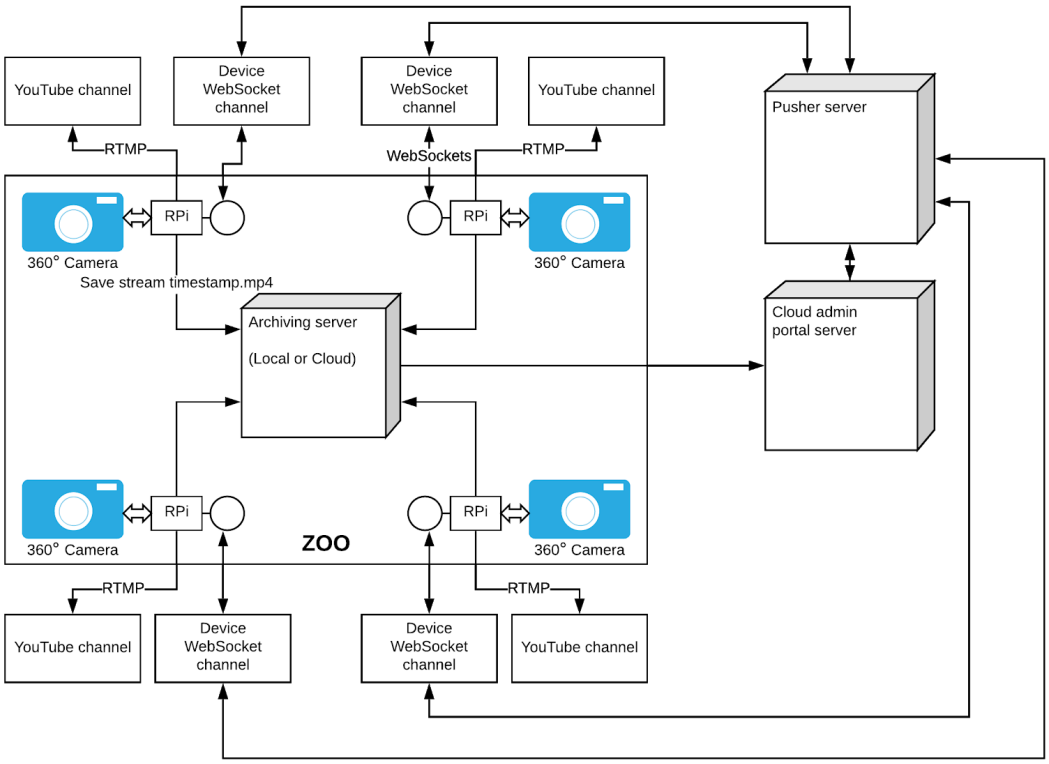

Figure 9: Initial Cloud Design

How It Works

Each 360° camera is behind an interface provided by the Raspberry Pi. Each Raspberry Pi will have its own WebSocket channel that it is subscribed to, where the backend can trigger events on, which in turn the Raspberry Pi will send the command related to that event to the 360° camera. The Raspberry Pi will run the FFmpeg library (transcoding software) which streams videos to YouTube as well as an archive to a given path.

360° Webcams

The Garmin Virb is compacted live streaming webcam that provides good accessibility for the developer to develop software around it. The webcam provides a REST API through WIFI. The Garmin Virb is able to stream and take still pictures in different resolutions that the developer can change through the provided REST API commands.

Storage Solution

The local or cloud-based solution that will archive 72 hours of the previous live stream from each webcam. Consequently, zoo staff will be able to go back at most 72 hours to address any issues related to animal exhibits. Finally, once a live stream is complete the machine learning algorithm will be executed here.

Admin Web Application

Since different zoos will be using the same admin portal, the web application will be hosted on the cloud. Users will be associated with a zoo, therefore, when a user logs in the Admin Portal they will be able to manage (depending on the user permission) the system related to the zoo he/she works in. Finally, users will be able to request a previous live stream if needed.

Node JS Server

Our first prototype on the back end was using Node JS. We have completed setting up the project with a fully functional API. The project was broken down into an MVC structure. We also implemented an ORM design to ensure modularity of the project going forward and to allow us to expand on our API functionality quickly and efficiently. Currently, we have set up the migrations, models, seeders, and controllers based on our Database Diagram below.

Database

Our initial prototype is using MySQL for the database. Currently, our design only includes the core aspects of the application. We begin expanding as the project progresses. This is something that we plan to do going forward. For now, we want to get the core functionality of the application and ensure there is a good communication between our software and hardware components. From there, we will expand on the React side of things, along with the necessary additions to the back end and database.

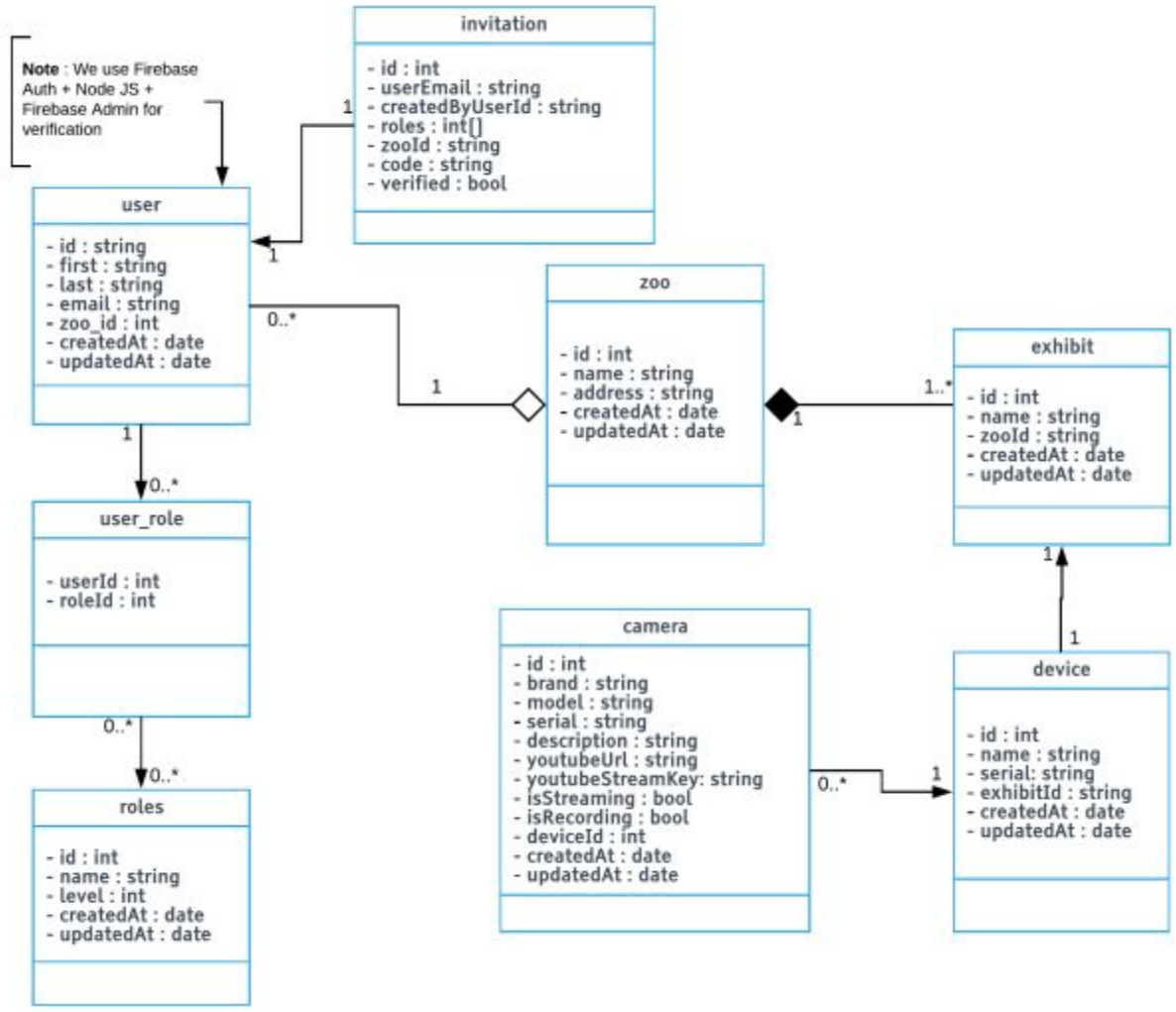

Figure 10: Initial Database Design

Other

The Garmin Virb 360 is able to stream in 4K resolution. Additionally, the Raspberry Pi handles synchronous streaming and archiving by using the FFmpeg library. The web application will use Firebase Authentication to prevent people from messing around with zoo's stream.

Appendix III - Other Considerations

Lessons Learned

Programming Languages

- Python
- Javascript
- Terraform
- NodeJS
- HTML
- SASS

Front-End Technologies

- React
- Redux
- CSS Zen Garden
- Webpack
- Javascript prettier

Back-End (Microservice) Technologies

- AWS SQS
- AWS S3
- AWS Batch
- AWS ECS
- Docker

Embedded Technologies

- FFmpeg
- Pusher

General Development Tools

- Git
- Gitlab version control
- Gitlab CI/CD

Administrative & Project Management

- Gitlab Boards
- Slack

Appendix IV - Definitions

Table 4: Definition Listings

Term	Definition(s)
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Delivery
CV	Computer Vision
ECS	Elastic Container Service
ML	Machine Learning
SQS	Simple Queue Service
UI	User Interface

Appendix V - References

“Build Something with Us.” *Home | Garmin Developers*, developer.garmin.com/.

Garmin, and Garmin Ltd. “Action Cameras | VIRB 360.” *Garmin*, buy.garmin.com/en-US/US/p/562010.

“Product.” *RICOH THETA*, theta360.com/en/about/theta/v.html.

RICOH THETA Developers, developers.theta360.com/en/.

“Transport Your Audience: VR in 8K.” *Insta360 Pro - Transport Your Audience: VR in 8K*, www.insta360.com/product/insta360-pro.

“True 360.” *True 360*, true-360.com/.