CS3505 Final Project, Fall 2020

Team Exoplanets
Edward Barrowes, Corbin Gurnee, Ted Goodell,
Marley Stark, Ivan Burkett, Garrett Keefe

December 9, 2020

1 Overview

Our group project was to develop a web-scraping and data synthesizing tool that could use FFMPEG to create a scaled timeline of data from multiple sources. The project needed to be easily scaled, flexible, run remotely, and require minimal user input to generate timelines for arbitrary time windows. Additionally, we were tasked with making use of cloud storage to the best of our ability to support the architecture of our project.

We made use of multiple technologies in developing our tool, including:

- Amazon Web Services
- Docker
- GitHub
- Python
- Selenium
- FFMPEG
- C++

We were successful in meeting the aims of this project, including the educational aim of practicing AGILE development in a team based environment, all of which is outlined within this document.

2 Architecture

2.1 Organization

Scalability is an important goal for this project, which factors in to the organizational choices our team made. At the top level, all of our code is run on remote environments, and at lower levels is compartmentalized to allow for easy scalability. This structure is illustrated in figure 1

The main operations of the program are run within a virtual machine instance using a service called Elastic Compute Cloud (EC2), which is itself a part of the greater Amazon Web Service (AWS). Within the EC2 instance are two Docker images: one for data collection, and one for data synthesis.

The data collection image uses a web-scraping tool called Selenium, to collect images and data from internet sources. Selenium supports multiple languages, but we chose to use python to interface with its scripts. Excepting the data collected from twitch, all our data, both text based[1] and image based, was collected using Selenium. For twitch, we wrote C++ code which directly interfaces with the Twitch API using a built-in C++ library called `lcurl`. These data are stored as .png images in cloud storage, again provided by Amazon, called the Elastic File System (EFS).

Data synthesis occurs within a Docker image containing code we constructed, framebuilder.cpp. framebuilder assembles individual images collected in the EFS, combining them into 600 time-consecutive frames using FFMPEG. Subsequently, the frames are assembled into a minute long video (again using FFMPEG) showing all of the data.

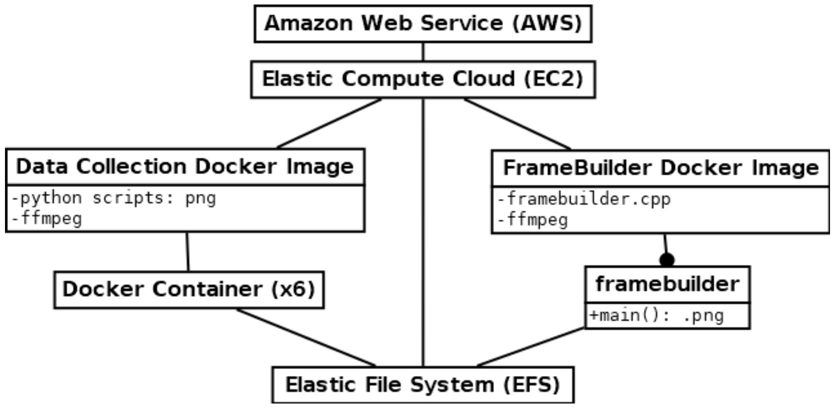


Figure 1: UML Class diagram depicting project structure

2.2 Execution

Execution occurs on both a continuous and instantaneous basis. Data collection occurs autonomously, and continuously, requiring no direct user input. Data synthesis occurs only when queried by the user interacting with the EC2 instance, illustrated in figure 2.

[1]In-between data collection and synthesis is a bit of formatting. Specifically, text based data (i.e. number of viewers on twitch) is converted to an .png file using a python script which interfaces directly with FFMPEG.

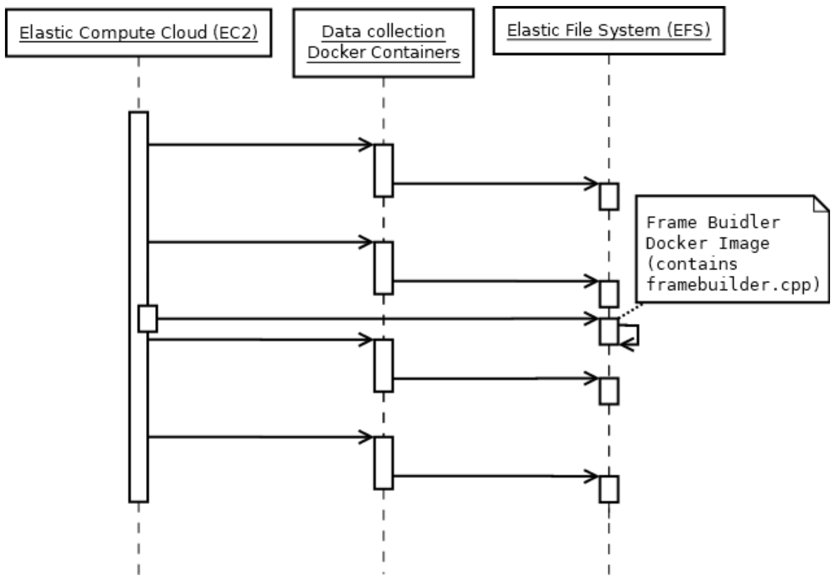

Elastic Compute Cloud (EC2)
Data collection Docker Containers
Elastic File System (EFS)
Frame Buidler Docker Image (contains framebuilder.cpp)

Figure 2: UML Sequence diagram depicting project execution

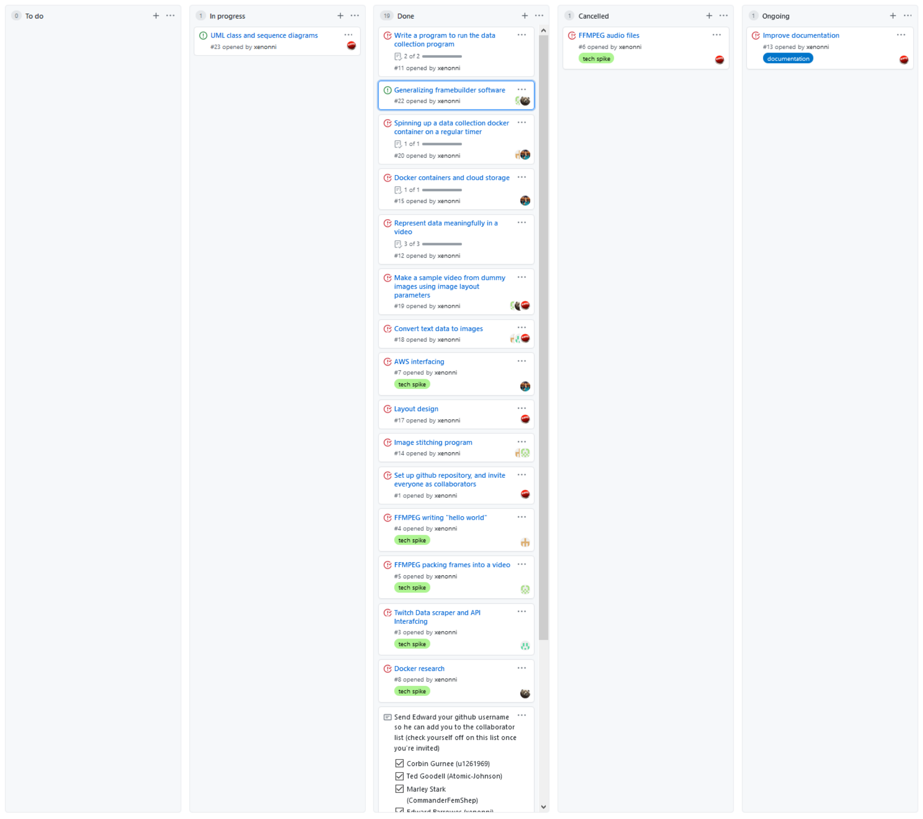
Autonomous data collection occurs at regular intervals, using the Linux command 'cron'. When data collection occurs, six Docker containers are instanced from the same image, each containing python code which collets data from various sources. The image itself contains all the code that all containers could need, however each data source updates at a different frequency. To compensate for this, we instance containers at appropriate time intervals in order to collect data.

A user can query the EC2 instance to produce a video compilation of collected data by running a bash script within the virtual machine. Running the script launches a Docker container, which contains our framebuilder.cpp program, which assembles all the data together into a minute long video. The container itself has FFMPEG installed, since framebuilder.cpp relies on its use.

3 Development

3.1 Organizational Tools

To organize our project, we made use of GitHub and its associated functionality. All our code was backed by a git repository[2], and our development progress was tracked using GitHub Projects and Issue Tracker seen in figures 3 and 4.



0 To do

1 In progress

UML class and sequence diagrams
#23 opened by xenonni

19 Done

Write a program to run the data collection program
2 of 2
#11 opened by xenonni

Generalizing framebuilder software
#22 opened by xenonni

Spinning up a data collection docker container on a regular timer
1 of 1
#20 opened by xenonni

Docker containers and cloud storage
1 of 1
#15 opened by xenonni

Represent data meaningfully in a video
3 of 3
#12 opened by xenonni

Make a sample video from dummy images using image layout parameters
#19 opened by xenonni

Convert text data to images
#18 opened by xenonni

AWS interfacing
#7 opened by xenonni
tech spike

Layout design
#17 opened by xenonni

Image stitching program
#14 opened by xenonni

Set up github repository, and invite everyone as collaborators
#1 opened by xenonni

FFMPEG writing "hello world"
#4 opened by xenonni
tech spike

FFMPEG packing frames into a video
#5 opened by xenonni
tech spike

Twitch Data scraper and API Interafcing
#3 opened by xenonni
tech spike

Docker research
#8 opened by xenonni
tech spike

Send Edward your github username so he can add you to the collaborator list (check yourself off on this list once you're invited)
- ☑ Corbin Gurnee (u1261969)
- ☑ Ted Goodell (Atomic-Johnson)
- ☑ Marley Stark (CommanderFemShep)

1 Cancelled

FFMPEG audio files
#6 opened by xenonni
tech spike

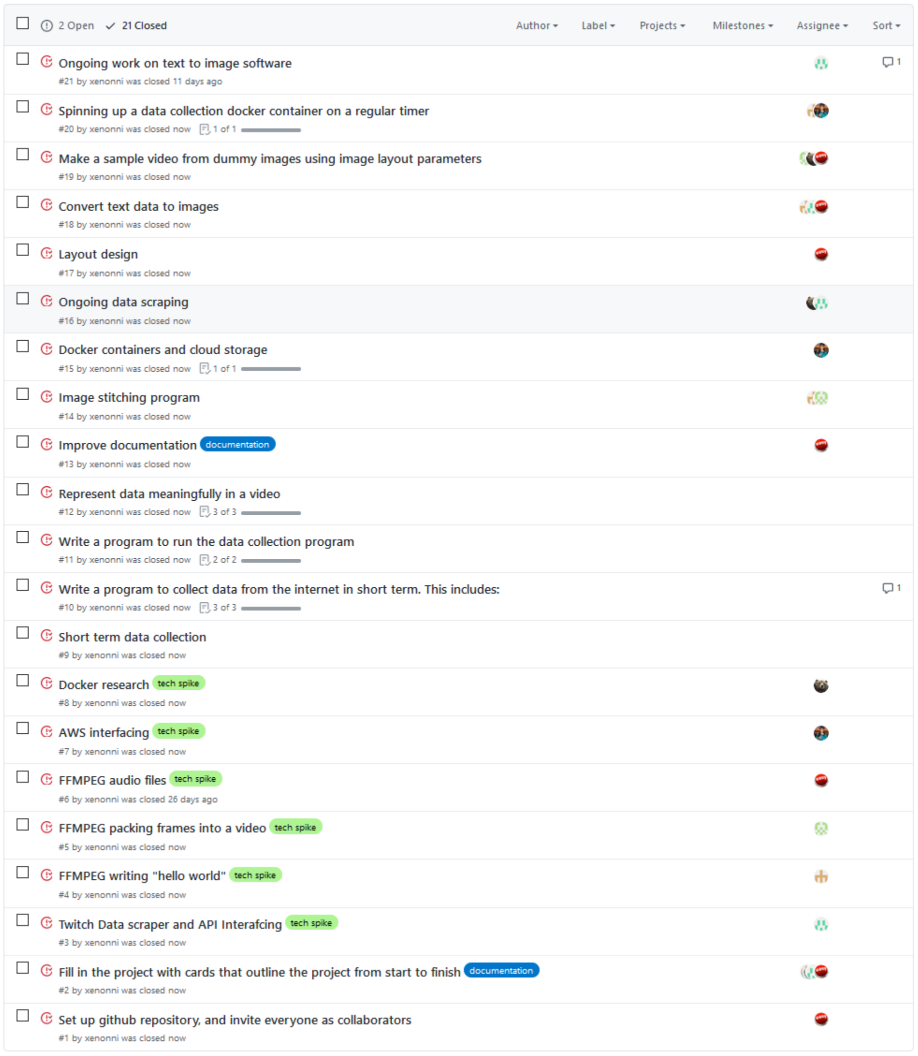
1 Ongoing

Improve documentation
#13 opened by xenonni
documentation

Figure 3: A screenshot showing the tracker, GitHub projects, which we used to organize development.

Additionally, we took regular project development meeting notes, which were also stored in the repository and have been attached in appendix A.

[2]https://github.com/xenonni/cs3505_space_data



2 Open ✓ 21 Closed | Author | Label | Projects | Milestones | Assignee | Sort

Ongoing work on text to image software
#21 by xenonni was closed 11 days ago | 1

Spinning up a data collection docker container on a regular timer
#20 by xenonni was closed now 1 of 1

Make a sample video from dummy images using image layout parameters
#19 by xenonni was closed now

Convert text data to images
#18 by xenonni was closed now

Layout design
#17 by xenonni was closed now

Ongoing data scraping
#16 by xenonni was closed now

Docker containers and cloud storage
#15 by xenonni was closed now 1 of 1

Image stitching program
#14 by xenonni was closed now

Improve documentation documentation
#13 by xenonni was closed now

Represent data meaningfully in a video
#12 by xenonni was closed now 3 of 3

Write a program to run the data collection program
#11 by xenonni was closed now 2 of 2

Write a program to collect data from the internet in short term. This includes:
#10 by xenonni was closed now 3 of 3 | 1

Short term data collection
#9 by xenonni was closed now

Docker research tech spike
#8 by xenonni was closed now

AWS interfacing tech spike
#7 by xenonni was closed now

FFMPEG audio files tech spike
#6 by xenonni was closed 26 days ago

FFMPEG packing frames into a video tech spike
#5 by xenonni was closed now

FFMPEG writing "hello world" tech spike
#4 by xenonni was closed now

Twitch Data scraper and API Interafcing tech spike
#3 by xenonni was closed now

Fill in the project with cards that outline the project from start to finish documentation
#2 by xenonni was closed now

Set up github repository, and invite everyone as collaborators
#1 by xenonni was closed now

Figure 4: A screenshot showing a backlog of completed tasks (closed issues) in GitHub projects.

3.2 Individual Reports

Corbin Gurnee - For this project, I assisted with multiple parts. I was assigned to figure out how to work with API's and how to get data with them. Thus I created a script to use the API on Twitch to gather the number of people watching the Science and Technology channel. After this I worked with Garret to create a Web scraper to get images off a webpage online. This turned out to be much easier than interacting with an API, so we changed all other scripts to run off of the web scraper as well. After this, I worked with Ted to convert the data we have compiled from our web scrapers to images. Finally, I worked in limited capacities with most other members of the team to assist in any area they may need assistance with.

Ted Goodell - I worked with Ivan on using the ffmpeg libraries to combine multiple images into a single image. To do this, we wrote code that decoded several images from their files and put them into frame structs that we could then programmatically combine into a larger frame. We then used the ffmpeg libraries to encode that frame and put it into an output image. I also, wrote python scripts that grabbed text data output by the data grabbers, and used the ffmpeg command line utility to put that text into an image. The ffmpeg command used a built in filter called "drawtext" to draw the text on a black template image for that portion of the final output image.

Ivan Burkett - I was assigned to work on the frame builder program for this project. I worked with Ted to create a C++ program called image-smoosh.cpp that used ffmpeg to combine the 6 frame segments into a single frame, then saved that frame as a png image. After finishing image-smoosh.cpp, I refactored the program to make it cleaner and easier to read and saved it as image-smoosh2.cpp. After doing that, I used image-smoosh2.cpp as a base to create framebuilder.cpp with Garrett. As part of making framebuilder.cpp, I also created a wrapper class for the C++ class directory-iterator called peek-directory-iterator, which simplified usage for directory-iterator and added a peek functionality which was necessary for our file traversal algorithm. framebuilder.cpp uses peek-directory-iterators to scan through the data source folders for the one most up to date for the provided date during the frame building process. Due to the way AWS's filesystem works, I ended up scrapping peek_directory_iterator. the final version loads all files in the directory and then sorts them, because AWS doesn't have any useful ordering to the files and directory_iterator doesn't ensure any kind of ordering. loading them all and sorting them was the only way to ensure the ordering.

Garrett Keefe - I first worked on gathering research on docker and how it works. Then I was assigned with Corbin to work on the data scrappers to collect data from our 6 sources. I wrote all of the .py files which used selenium to navigate and grab web resources while Corbin wrote the twitch.out file which uses C++ and the curl libraries to query twitch.tv for info from its API. After that I was assigned to working with Ivan to create the framebuilder program which utilize UNIX timestamps to assemble data from our data sources and compile them together into individual frames, allowing us to generate a list of frames which we use an ffmpeg command to assemble them into a video. I also helped Ted create all of the .sh files which acted as drivers for our data collection and converting to picture formats within the docker containers.

Edward Barrowes - Most of my work on this project was in documentation and organization. I took meeting notes whenever we met and discussed anything of significance. Additionally, I was responsible for setting up the GitHub repository and populating the issue tracker with tasks we had discussed during meetings (though individual team members were responsible for updating progress on tasks they had been assigned). In addition to providing logistical support to the team, I also designed the final layout for the video based on the requirements and constraints the various data sources had. Lastly, as we finished the project, I was responsible for compiling our design discussions and sketches (from weekly meetings) into more professional communication, including this document and the UML diagrams contained within.

Marley Stark - I worked on AWS to get our EC2, EFS, and lambda functions set up so we could have a virtual environment on the cloud to automate and run our entire program and data collection. I made an AWS Educate account so we could explore what AWS provides without having to worry about being charged, and it seemed to have access to everything we needed for the assignment. However the AWS Educate account does not let you schedule EC2 instances so we were unable to set up the Cloudwatch events needed to use lambda functions on a schedule. Our lambda functions work to turn on and off our EC2, we just couldn't schedule them to automate it. Taking advantage of Linux's cron daemon, I edited the crontab file in Cloud9 to schedule our data collection jobs, and it works fantastic! So overall I navigated the immense world of AWS and made a Cloud9 EC2 instance and mounted an EFS to it, learned about Lambda functions and Cloudwatch but was unable to schedule things because of the AWS Educate account restrictions, and instead learned about and implemented cron jobs to do our automated data collection tasks.

4 Results

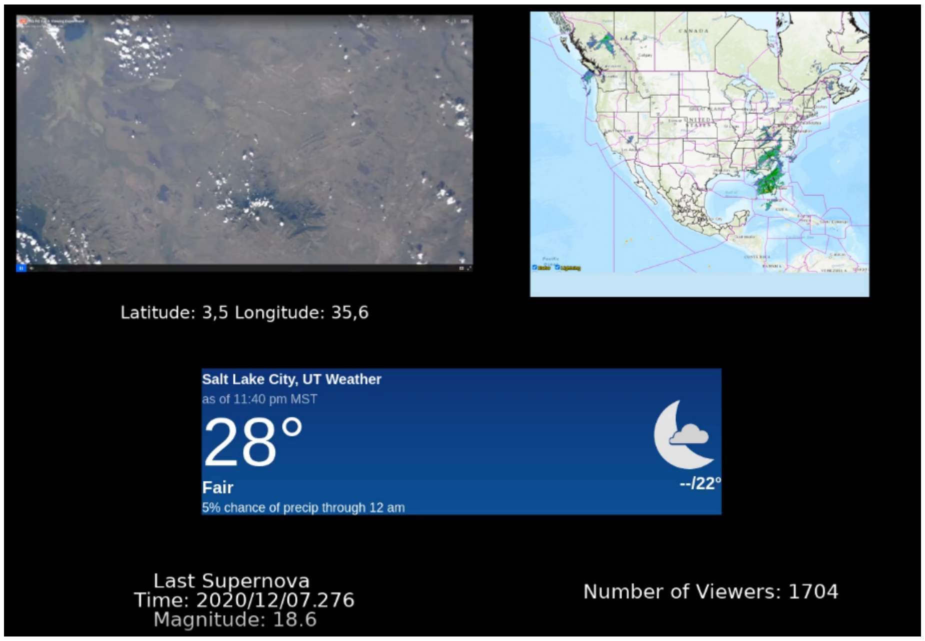
According to the goals of our project, we have complete functionality. Our project works fully as intended because we can leave our EC2 instance running to collect data without having to even be signed into our AWS account. The timeline video is generated with minimal user input.

4.1 Running the Code

To make a video from the data you simply run the command

```
./create_video.sh MM.DD.YYYY.HH.mm MM.DD.YYYY.HH.mm
```

with the desired start and end times.



Latitude: 3,5 Longitude: 35,6

Salt Lake City, UT Weather
as of 11:40 pm MST
28°
Fair
5% chance of precip through 12 am
--/22°

Last Supernova
Time: 2020/12/07.276
Magnitude: 18.6

Number of Viewers: 1704

Figure 5: A screenshot from the completed timeline.

5 Appendix A: Meeting Notes

5.1 Meeting notes 10/29

```
Todo for next checkpoint (1 week from today)
---------------------------

TA: Josh Nelson, Thursday 4:15 PM
Setup github repo
Using the tracking tool of choice (github projects) to setup a
↪  backlog of work, broken into tasks that take 3-4 hours,
↪  leading from start to finish. These can be updated and
↪  revised throughout the entirety of the project.
Plan the first sprint
Assign tasks to people for the first sprint
Recommended: Spike tasks as part of the sprint
  1-2 hour long tasks
    example: draw a smiley face with ffmpeg as a .png
    The idea is so you know how to interface with a a part of the
      ↪  system for future troubleshooting
Plan regularly scheduled standups. Everyone should be there,
↪  everyone must share:
  What they're working on
  What they're stuck with
  What they're doing next
  Suggested one standup per sprint
  Each person gets about 1-2 minutes for this
The timescale of the minute of output data is up to you
  it could be as much as 48 hours compressed into a minute
  another group using 1/day covid data was "pushing it", so probably try not to go lower frequency than that for more than one data source
  Grabbing the most recent data point from a slow updating source would be fine
  The most ideal would be to have data that updates on the scale of hours rather than days
Data seeding: ask the professor
We don't have to use the exact same time slot, but we should try and get the same TA
```

5.2 Meeting notes 11/3

```
Members Absent: None

Next meeting
TA: Josh Nelson, Thursday 4:15 PM

Tech spikes
  Data scraper & API interfacing - Corbin
    Continuous data monitoring
    API Keys
  ffmpeg
    a program that writes the text "hello world" in a picture - Ted
    take multiple frames and pack them into a video - Ivan
    dealing with audio files - Edward
  AWS or other cloud based computing - Marley
    Somewhere, the docker container must run on a cloud service and use cloud storage to run the program
  More docker research - Garrett
    Running additional docker containers from a container, in order to run additional tasks?

Overall Project steps
  Program to collect data from the internet in the short term
    Identify data sources (complete)
    Identify ways to get the data
      scraper or API
    Unified format to save data in
      might depend on ffmpeg
  Program to run the data collection program
    Involves deployment of docker containers
    Needs to run for a long time (about 24h) calling data
      ↪  collection program at regular intervals
  Store the data
    Persistent and accessible cloud storage where all the data is
      ↪  collected
  Represent data meaningfully in a video
    Visualize data
    Scale the time appropriately for a minute long video
    Output video file

Tasks
  Make github cards - Edward
  Contact TA about moving the meeting time to 3:30 PM, Thursdays
    ↪  - Marley
  Data sources
    Supernovae data -
      ↪  http://www.rochesterastronomy.org/snimages/sndate.html
      Updates once per day, with multiple entries (2-8)
      Does it update at a regular time?
      Might be too slow
    Weather data - https://aprs.fi/weather/a/FW3937?range=day
      API - figure out which of these works best
        https://aprs.fi/page/api

          ↪  https://weather.com/swagger-docs/ui/sun/v3/sunV3CurrentsOnDemand.json
    Lightning data -
      ↪  https://ghrc.nsstc.nasa.gov/hydro/details/isslis_v1_nrt
    Twitch viewer data
      updates very rapidly
      API - https://dev.twitch.tv/docs/api/
    ISS location tracker -
      ↪  https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/International_
      updates very rapidly
    https://api.nasa.gov/
      Garrett will pick a data source from this API

Questions for TA
  Running additional docker containers from one container?
    Also, is this required? or can all the code run from a single
      ↪  docker container? (no)
  Any required documentation, beyond github projects?
    no additional documentation required
  Do we need to pull actual images/audio from the web, or just make images/audio to represent data?
    text to speech could count as pulling audio data from the internet
```

5.3 Meeting notes 11/10

```
Members absent: none

Tech spike updates

 - Edward has done some research on ffmpeg audio, but isn't sure how much more we want to pursue it for our project in light of the most recent TA meeting
 - Ivan had some demo code to show on packing frames into a video. There were some parts of the code that weren't well understood, so we all discussed together. Ivan also showed us a video that was generated by ffmpeg
 - Marley looked into web services to host our project on, and applied for an AWS Educate Starter Account.We should get access in the next few days or so.
Garrett identified some features that we may potentially need from AWS:
   - Elastic container storage
   - Event handling
   - Lambdas (maybe)
 - Garrett reviewed some of the parameters for docker commands
--rm can be used to remove a container once it has run
Garrett also wrote a .bat file to run the docker container he was demonstrating, which was added to a separate branch of the repository
 - Corbin looked through and knows what to do to implement API of twitch, will have it fully working by Thursday at the latest.
 - Ted looked into using libavfilter to draw text on the screen. He is going to modify the video filter example on

Visual layout discussion
Have a circle which displays temperature as text in Salt Lake City, potentially scrape image from weather.com, center bottom
ISS image grab as well as latitude and longitude can go in the top left corner
Lightning strike map image can go in the top right corner
Supernova data on the bottom left
Twitch data on the bottom right
```

See meeting_notes_image_11_10.png for more detail

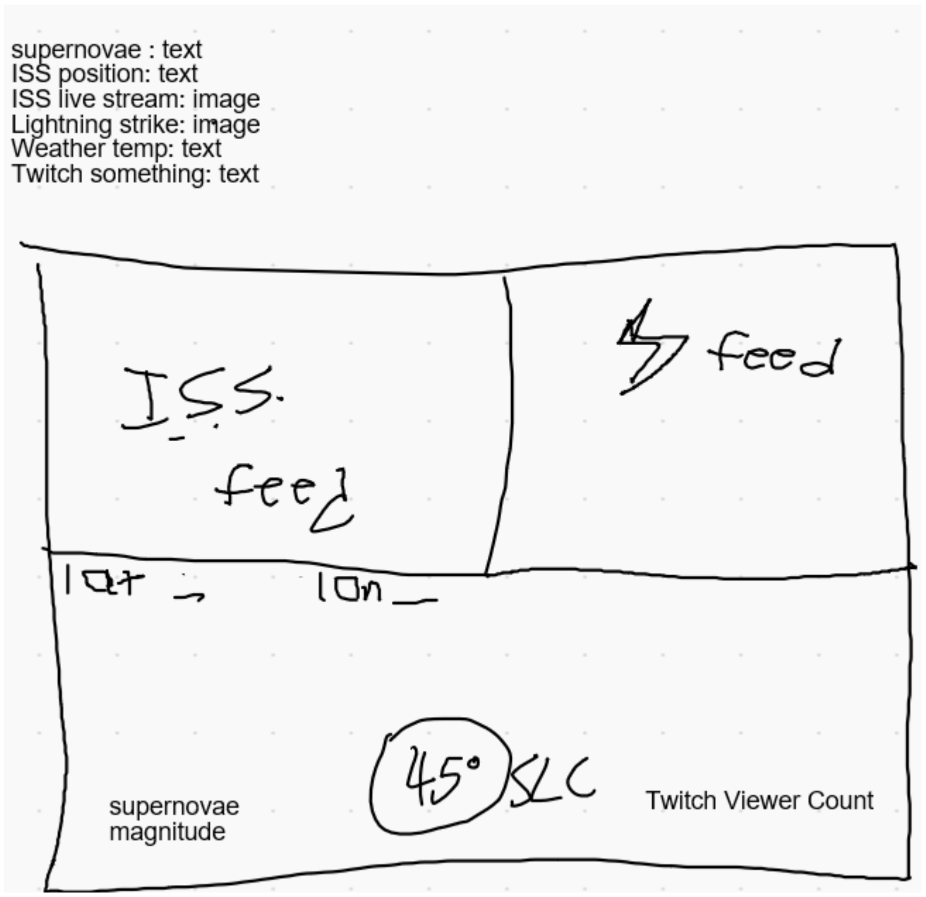


Figure 6: Image from meeting notes for 11/10

5.4 Meeting notes 11/12

One of the requirements is to write C++ code that interfaces with FFMPEG at some point. Displaying weather data using a color drawn by ffmpeg (orange for increase, blue for decrease) could easily satisfy this requirement.

We discussed turning all data into images and storing the images in the elastic file storage, which is how we will store data going forward.

```
By giving each data source a fixed section of the overall video frame, we can display all the data in the final video by copying a given data source's image into its designated region. These regions would be hard coded.
For all tasks, reach out if anyone needs any help
Tasks for this week:
 - someone should write a program that will assemble all the separate images (data sources) into one frame. Expect a progress update on tuesday on whether or not the task will need more time. Assigned to Ted and Ivan
 - ongoing preparation of the AWS environment, especially so that it's completely ready for other parts of the project. Specifically, spinning up a docker container on a regular timer. Make sure to have the E2 and elastic storage set up.
 - Garret will help Corbin with ongoing web scraping
 - Edward will design the layout of all individual data sources, i.e. how big each box should be, where it should go, as concretely as possible.
```

5.5 Meeting notes 11/17

```
Members Absent: None
Next meeting: from 3:20 to 4:15 on Thursday, 11/19

Garrett and Corbin have finished with the web scraper, but some data (i.e. supernovae) are stored in plain text right now.
    Python scripts and selenium have been added to the git repo
    TODO store text data as images, using the ffmpeg command (Ted worked on this). This should be relatively easy
Ivan and Ted completed their task of image stitching.
Their program requires input images to be .png using rgba pixel format.
ffmpeg -i <filename.png>
The above command can be used to check the pixel format
Edward is continuing work on finalizing the layout, including specific pixel measurements and placement
Marley has Cloud9 setup, as well as the elastic file storage, but there was some difficulty linking them together
    The professor has said that he will make a tutorial on this process as many students are struggling with it, but no materials have been released as of yet

Tasks for next time
Corbin, Ted and Edward - Convert text data to images
Garrett, Ivan and Edward - Make a sample video from dummy images using image layout parameters
```

```
    Time scaling: make sure that data are displayed for an appropriate amount of time, and at the correct time
Edward - finish the layout measurements by thursday
Marley - continue work on AWS related issues

Questions
Can we have help and/or a tutorial on Linking EFS with EC2? There seems to be some difficulties with security specifically.
    See announcement from today
```

5.6 Meeting notes 11/24

```
cron and/or lambdas?
  Cron requires continuous running of a machine
  Lambdas do not
Split this into its own issue on the issue tracker
We would like to use lambda statements
AWS and EFS related tasks are complete
Marley posted instructions for working with EFS in the discord server: make sure to read and follow
EFS structure and naming conventions - directory framesegments has 6 subdirectories
  issfeed
  isscoords
  lightningmap
  supernovae
  twitch
  weather
  frames

Ted demonstrated completed work on the text-to-image task. Due to power outages, sample output was not available.
Some supernovae data points occur with identical timestamps, which conflicts with the requirement that files be named after the unix time at which the data point occurred. We discussed and corrected this by having the script overwrite data points with identical times, so that only the last read data point is actually saved persistently.
Corbin is still working on parts of the text-to-image task.

When and how often should we start collecting data?
  Supernovae: once per day
  Other data sources: about once every 30 minutes
  Start: ASAP
Where should docker files be stored?

```

```
Tasks for next week:
Scheduled automatic data collection via lambda statements - Marley and Ted
Ask Prof. Jensen about meeting next week - Edward
UML class and sequence diagrams - Edward
Current framebuilder program is hardcoded, and needs to be generalized to take arbitrary time ranges - Ivan and Garrett
Ongoing work on
An output video

Next meeting 1 PM sunday - let's keep it short please! (absolutely less than an hour)
Please report if you finish anything early
Review submission requirements
```

5.7 Meeting notes 11/29

```
Members absent: Ted

We commiserate over a lack of replies from Prof. Jensen...

Marley:
We really need prof. Help on this.
EC2 doesn't work on a schedule, so we use a cron job. Docker commands are run every 2 hours. It seems the amazon account logs us out at unpredictable times.
Automation of AWS stuff would require a new account without restrictions and some code refactoring
Lambda functions do work, but account restrictions prevent us from fully implementing it
Which commands need to go into the cron job file (to be run every 2 hours)?
    Supernovae can be run once every 2 hours
    Other sources can be run once every 30 minutes
For now, let's run data collection on a local machine using cron so we have something to show.

Garret and Ivan:
Generalizing the framebuilder software is going well and should be done tonight or tomorrow.

Edward: a bit behind on task. professor hasn't posted any instructions for assignment submission and documentation, which makes completing that requirement ambiguous at the moment.

```

```
Next meeting: tuesday at 2
```

5.8 Meeting notes 12/1

```
Garrett and Ivan are nearly done, scales correctly with time and
 ↪  grabs images. Just needs command line arguments to specify a
 ↪  time window.
Turn in date: let's set an appointment for 1 PM on the 10th
Edward has been busy and hasn't worked more on documentation yet,
 ↪  but should be good moving forward.
Marley will send Edward a paragraph explaining troubleshooting
 ↪  AWS problems to include in the documentation
Continued editing of the cron tab file
Text conversion python scripts should output to /dataStorage
We discussed some of the details of when docker containers should
 ↪  run and how often.
Supernovae makes multiple files, so maybe run it only once every
 ↪  two hours.
What Ted needs is a list of shell commands for each docker
 ↪  container and how frequently each of them should be run.
 ↪  Garrett can provide this.
Once the .sh files are done (Garrett and Ted), Marley can
 ↪  schedule them with cron tab.
Goal: start data collection by tomorrow night
What path did we decide to mount EFS to?
    Marley posted a screenshot in chat
Edward submitted times to meet with the professor
Edward posted on piazza asking for clarification on the documents
 ↪  to be submitted
Next meeting: 2PM thursday
```

6 Appendix B: Layout design

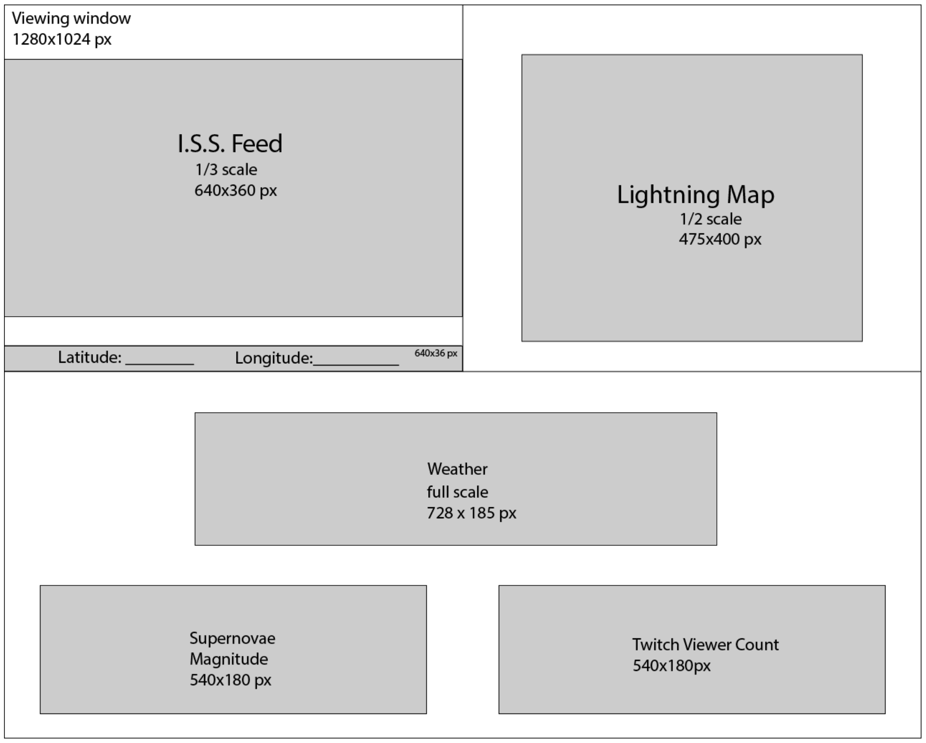


Figure 7: A finalized version of the layout from 11/10