DigitalCast - A digital signage management platform

Datacenter Scale Computing (Fall 2024) - CSCI 5253-872

INSTRUCTOR: Prof. Eric Keller
Fall 2024

Project Participants:

- Aneesh Khole
- Uttara Ketkar

Introduction

Digitalcast is a cloud-centric digital signage solution designed to streamline the process of media management for businesses and individuals. By leveraging advanced cloud technologies, it enables users to upload, schedule, and display media files such as images and videos seamlessly. The system is built with scalability, reliability, and user-friendliness in mind, ensuring it caters to diverse user needs while handling concurrent workloads efficiently. Digitalcast represents an innovative approach to digital media management by utilizing a robust backend, cloud storage, and message queuing systems to decouple workflows and optimize performance.

Project Goals

The primary goal of the Digitalcast project is to create a cloud-based digital signage platform that offers users a seamless experience for uploading, scheduling, and displaying media. By leveraging advanced cloud technologies, the system aims to provide scalability, reliability, and user satisfaction. Digitalcast focuses on:

1. Building a user-friendly interface for uploading media files, ensuring ease of use for non-technical users.
2. Enabling precise scheduling capabilities so media can be displayed at specific times, tailored to individual or organizational needs.
3. Establishing a robust backend to manage metadata, synchronize media storage, and display operations efficiently.
4. Using Kubernetes and Google Cloud Platform (GCP) services to achieve scalability and handle increased workloads effortlessly.

This project demonstrates the practical application of cloud-based solutions to address real-world challenges in digital signage management.

Software and Hardware Components

Purpose of Components

Each component in the system has a specific role to ensure seamless functionality:

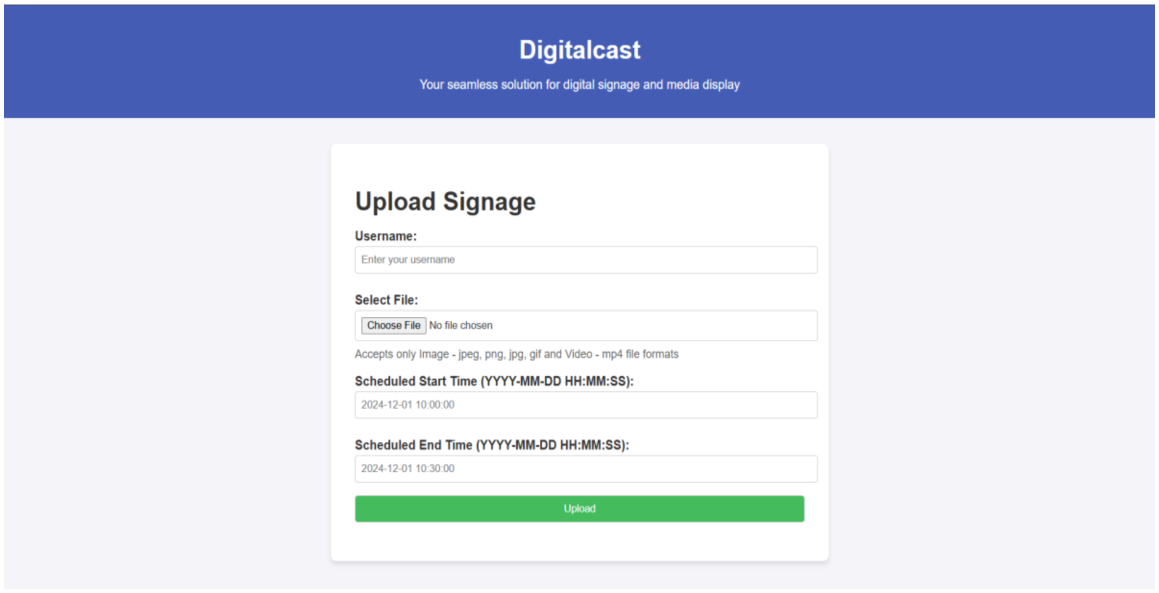
- **Frontend (REST API Client):**
    - The Flask-based interfaces provide a user-friendly way to upload and display media, ensuring accessibility for non-technical users.
    - However, their reliance on periodic updates may lead to minor synchronization delays in real-time environments.
- **Backend (REST API Server):**
    - The Python-based backend enables efficient media validation, storage, and scheduling logic, streamlining workflow.
    - Yet, its heavy reliance on Flask could introduce limitations in handling very high concurrency.
- **Database (PostgreSQL):**
    - Robust metadata storage and real-time query capabilities facilitate seamless media scheduling.
    - However, database throughput can be a bottleneck during heavy concurrent uploads, impacting system performance.
- **Cloud Services (GCS, Pub/Sub):**
    - These services ensure scalability and asynchronous workflows, enabling smooth handling of workloads.
    - The disadvantage lies in the cost implications, especially with large-scale usage of storage and messaging services.
- **Kubernetes:**
    - Dynamic scalability and efficient load balancing ensure the system can handle fluctuating user demands effectively.

- Nonetheless, Kubernetes adds operational complexity, requiring skilled management to avoid misconfigurations.

Software Components

Frontend (REST API client)

- **upload.html:** A flask web interface allowing users to upload media files, specify schedule details, and view success messages.



Digitalcast

Your seamless solution for digital signage and media display

Upload Signage

Username:

Enter your username

Select File:

Choose File No file chosen

Accepts only Image - jpeg, png, jpg, gif and Video - mp4 file formats

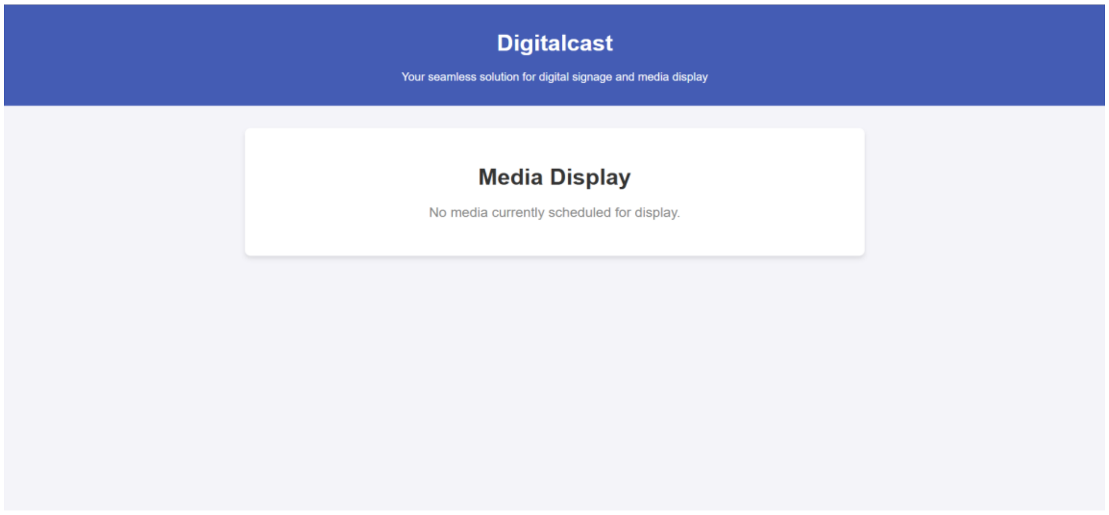
Scheduled Start Time (YYYY-MM-DD HH:MM:SS):

2024-12-01 10:00:00

Scheduled End Time (YYYY-MM-DD HH:MM:SS):

2024-12-01 10:30:00

- **display.html:** A dynamic web interface for displaying scheduled media, with periodic updates to reflect new content.



Digitalcast

Your seamless solution for digital signage and media display

Media Display

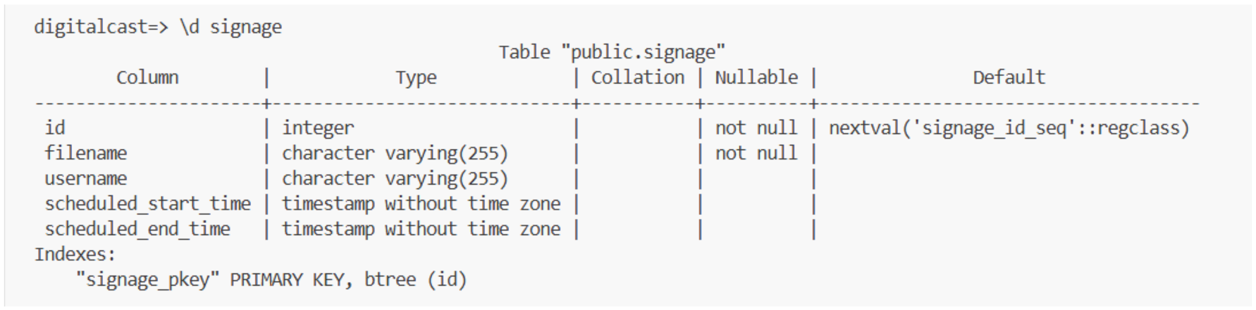
No media currently scheduled for display.

Backend (REST API server)

- **app.py:** Handles file uploads, validates media types, and interacts with the database and Google Cloud Storage. It also publishes metadata to the Pub/Sub topic for further processing.
- **display.py:** Queries the database for scheduled media and fetches files from GCS for display during their specified times.
- **worker.py:** Subscribes to the Pub/Sub topic to process metadata and ensure media readiness for scheduled display.

Database

- **PostgreSQL:** Stores metadata such as filenames, usernames, and schedule times. Provides robust querying capabilities to support real-time media scheduling and retrieval.



```
digitalcast=> \d signage
                                          Table "public.signage"
       Column         |            Type             | Collation | Nullable |                Default
----------------------+-----------------------------+-----------+----------+-------------------------------------
 id                   | integer                     |           | not null | nextval('signage_id_seq'::regclass)
 filename             | character varying(255)      |           | not null |
 username             | character varying(255)      |           |          |
 scheduled_start_time | timestamp without time zone |           |          |
 scheduled_end_time   | timestamp without time zone |           |          |
Indexes:
    "signage_pkey" PRIMARY KEY, btree (id)
```

Cloud Services

- **Google Cloud Storage (GCS):** Stores uploaded media files securely and efficiently.
- **Google Pub/Sub:** Decouples upload and display workflows, enabling asynchronous communication and scalable processing.

Hardware Components

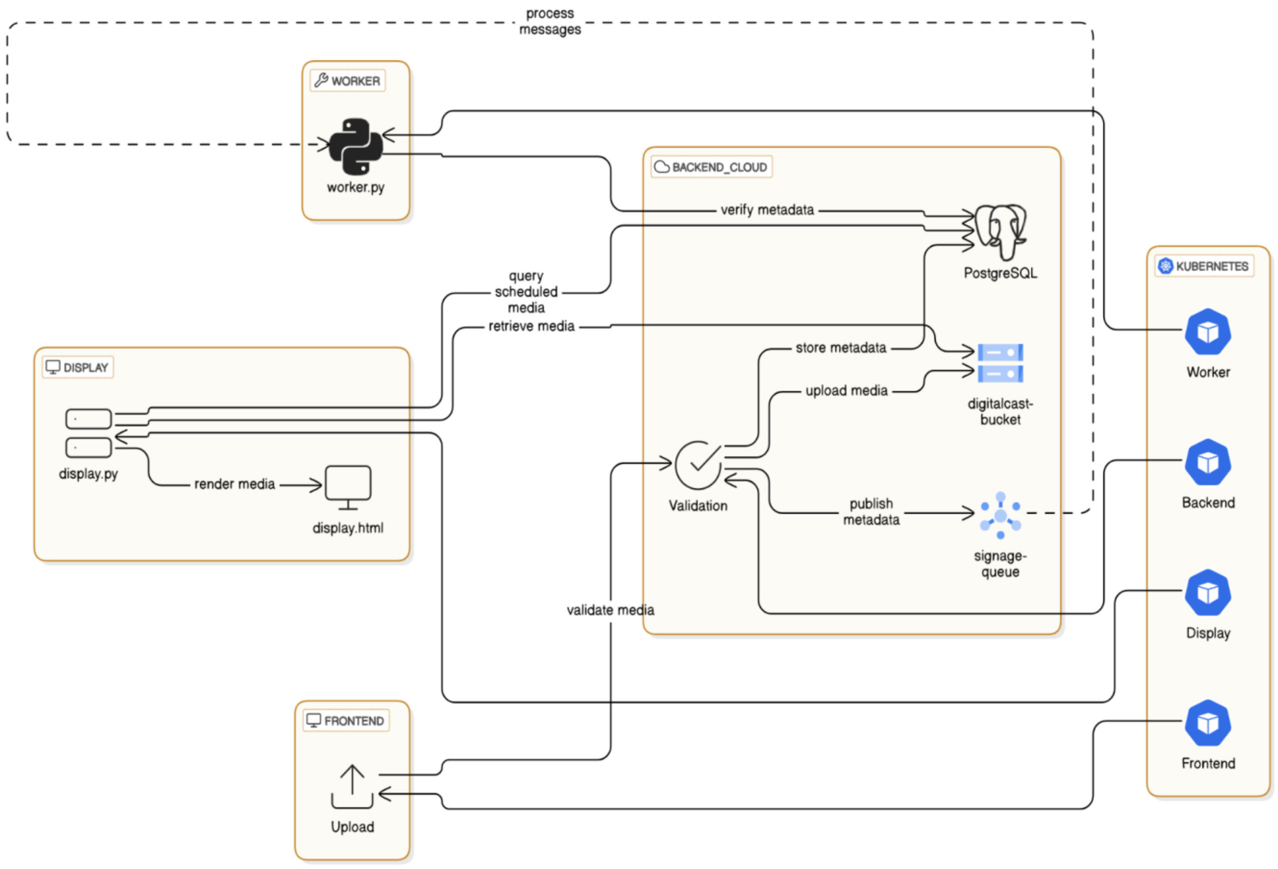
- **Kubernetes Cluster:** Manages containerized microservices, ensuring scalability and load balancing.

- **Virtual Machines:** GCP-provided instances hosting the application components and database.

Requirements satisfied

1. REST API - Communication between different components
2. Google Cloud Storage (GCS)
3. Cloud SQL - PostgreSQL Database
4. Google Pub/Sub - Message queueing
5. Kubernetes - Scalability

Architectural Diagram



Component Interactions

The Digitalcast system consists of multiple interconnected components that work together to ensure seamless media scheduling and display process. These interactions are as follows:

Upload Process

1. **User Interaction:** Users access the /upload endpoint via the upload.html interface to provide their username, media files (images or videos), and scheduling details (start and end times).
2. **Validation:** The system validates the uploaded files to ensure they conform to the supported formats.
3. **Cloud Storage:** Once validated, media files are uploaded to a Google Cloud Storage bucket (digitalcast-bucket).
4. **Database Entry:** Metadata, including the filename, username, and schedule times, is stored in PostgreSQL.
5. **Message Publication:** A Pub/Sub message is generated with the media details, triggering downstream processing for scheduling and display.

Display Process

1. **Metadata Query:** The display.py Flask application continuously queries PostgreSQL to fetch metadata for scheduled media.
2. **Media Fetching:** Media files are retrieved from GCS and displayed via the display.html interface during their scheduled times.
3. **User Attribution:** The username of the media uploader is displayed alongside the media, ensuring clear attribution.
4. **Real-time Updates:** The display interface refreshes periodically to reflect changes and update the displayed media dynamically.

Pub/Sub Workflow

Google Pub/Sub decouples the upload and display workflows, ensuring efficient and asynchronous communication between components:

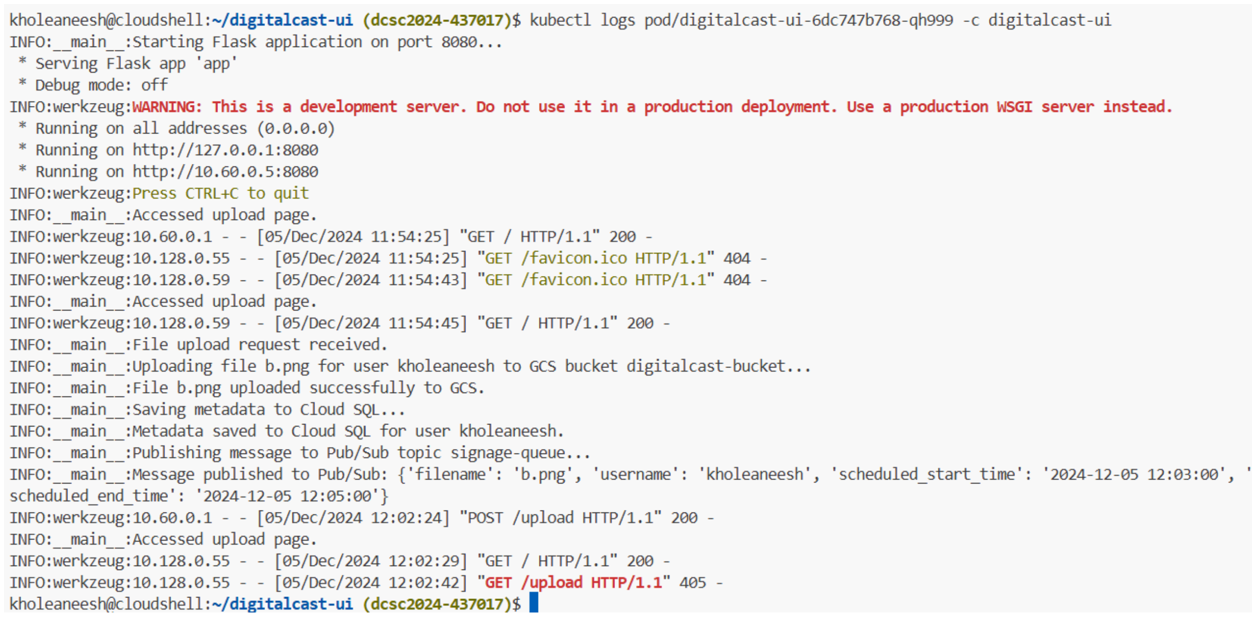
- Upon successful upload, metadata is published to the signage-queue topic.
- The worker.py script subscribes to the signage-queue-sub, processes incoming messages, and ensures media readiness for display at the scheduled times.
- This decoupling enables scalability and reduces processing latency.

Debugging and Testing Mechanisms

Debugging

To ensure the system operates smoothly, several debugging strategies were implemented:

- **Comprehensive Logging:** All Flask applications and the worker script include extensive logging to capture system events, errors, and user interactions.



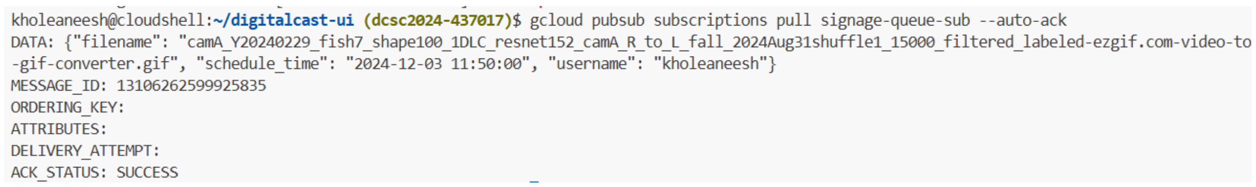
```
kholeaneesh@cloudshell:~/digitalcast-ui (dcsc2024-437017)$ kubectl logs pod/digitalcast-ui-6dc747b768-qh999 -c digitalcast-ui
INFO:__main__:Starting Flask application on port 8080...
 * Serving Flask app 'app'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://10.60.0.5:8080
INFO:werkzeug:Press CTRL+C to quit
INFO:__main__:Accessed upload page.
INFO:werkzeug:10.60.0.1 - - [05/Dec/2024 11:54:25] "GET / HTTP/1.1" 200 -
INFO:werkzeug:10.128.0.55 - - [05/Dec/2024 11:54:25] "GET /favicon.ico HTTP/1.1" 404 -
INFO:werkzeug:10.128.0.59 - - [05/Dec/2024 11:54:43] "GET /favicon.ico HTTP/1.1" 404 -
INFO:__main__:Accessed upload page.
INFO:werkzeug:10.128.0.59 - - [05/Dec/2024 11:54:45] "GET / HTTP/1.1" 200 -
INFO:__main__:File upload request received.
INFO:__main__:Uploading file b.png for user kholeaneesh to GCS bucket digitalcast-bucket...
INFO:__main__:File b.png uploaded successfully to GCS.
INFO:__main__:Saving metadata to Cloud SQL...
INFO:__main__:Metadata saved to Cloud SQL for user kholeaneesh.
INFO:__main__:Publishing message to Pub/Sub topic signage-queue...
INFO:__main__:Message published to Pub/Sub: {'filename': 'b.png', 'username': 'kholeaneesh', 'scheduled_start_time': '2024-12-05 12:03:00', '
scheduled_end_time': '2024-12-05 12:05:00'}
INFO:werkzeug:10.60.0.1 - - [05/Dec/2024 12:02:24] "POST /upload HTTP/1.1" 200 -
INFO:__main__:Accessed upload page.
INFO:werkzeug:10.128.0.55 - - [05/Dec/2024 12:02:29] "GET / HTTP/1.1" 200 -
INFO:werkzeug:10.128.0.55 - - [05/Dec/2024 12:02:42] "GET /upload HTTP/1.1" 405 -
kholeaneesh@cloudshell:~/digitalcast-ui (dcsc2024-437017)$
```

- **Database Integrity Checks:** SQL queries are designed to ensure metadata consistency and proper storage.
- **Error Handling:** Robust error-handling mechanisms are in place to:
  - Log GCS errors when files are not found or fail to upload.
  - Retry Pub/Sub message processing in case of failures.
- **Validation:** Uploaded files are rigorously validated to prevent unsupported formats from entering the system.

Testing

Testing focused on validating individual components and the overall system workflow:

- **Unit Testing:** Flask endpoints were tested for both valid and invalid inputs to ensure robust error handling.



```
kholeaneesh@cloudshell:~/digitalcast-ui (dcsc2024-437017)$ gcloud pubsub subscriptions pull signage-queue-sub --auto-ack
DATA: {"filename": "camA_Y20240229_fish7_shape100_1DLC_resnet152_camA_R_to_L_fall_2024Aug31shuffle1_15000_filtered_labeled-ezgif.com-video-to
-gif-converter.gif", "schedule_time": "2024-12-03 11:50:00", "username": "kholeaneesh"}
MESSAGE_ID: 13106262599925835
ORDERING_KEY:
ATTRIBUTES:
DELIVERY_ATTEMPT:
ACK_STATUS: SUCCESS
```

- **Integration Testing:** End-to-end tests simulated the entire upload-to-display process, verifying each step's correctness.
- **Kubernetes Testing:** Deployment configurations were tested to ensure seamless scaling and reliability under different workloads.

System Capabilities and Bottlenecks

Capabilities

Digitalcast leverages cloud technologies to deliver several key capabilities:

- **Scalability:** Kubernetes ensures the system scales automatically based on workload demands, handling high user concurrency efficiently.

- **Reliability:** Google Pub/Sub guarantees reliable message delivery, ensuring scheduled media is processed and displayed as intended.
- **Flexibility:** The system supports overlapping schedules and allows multiple users to schedule media with clear attribution.

Bottlenecks

Despite its robust architecture, the system has a few limitations:

- **Database Throughput:** High concurrent uploads can strain PostgreSQL, potentially affecting performance.
- **Latency:** Pub/Sub message processing introduces minor delays, impacting real-time updates.
- **Storage Costs:** Large media files in GCS increase storage expenses, especially for extended usage.

Conclusion

Digitalcast successfully demonstrates the integration of cloud technologies to build a scalable, reliable, and user-friendly digital signage solution. By leveraging GCP services such as Kubernetes, Pub/Sub, and Cloud Storage, the system provides an efficient platform for managing media uploads, scheduling, and displays. The decoupled architecture ensures scalability and flexibility, meeting diverse user needs while maintaining robust performance.

While there are challenges related to database throughput and storage costs, these can be mitigated with optimization techniques such as database indexing and implementing media compression. Digitalcast serves as a strong foundation for further innovations in cloud-based media management and highlights the potential of cloud computing in solving real-world challenges.