CLOUD APPLICATION AND DEVELOPMENT(CAD)

PROJECT: Media Streaming with IBM Cloud Video Streaming

Phase 5: Project Documentation

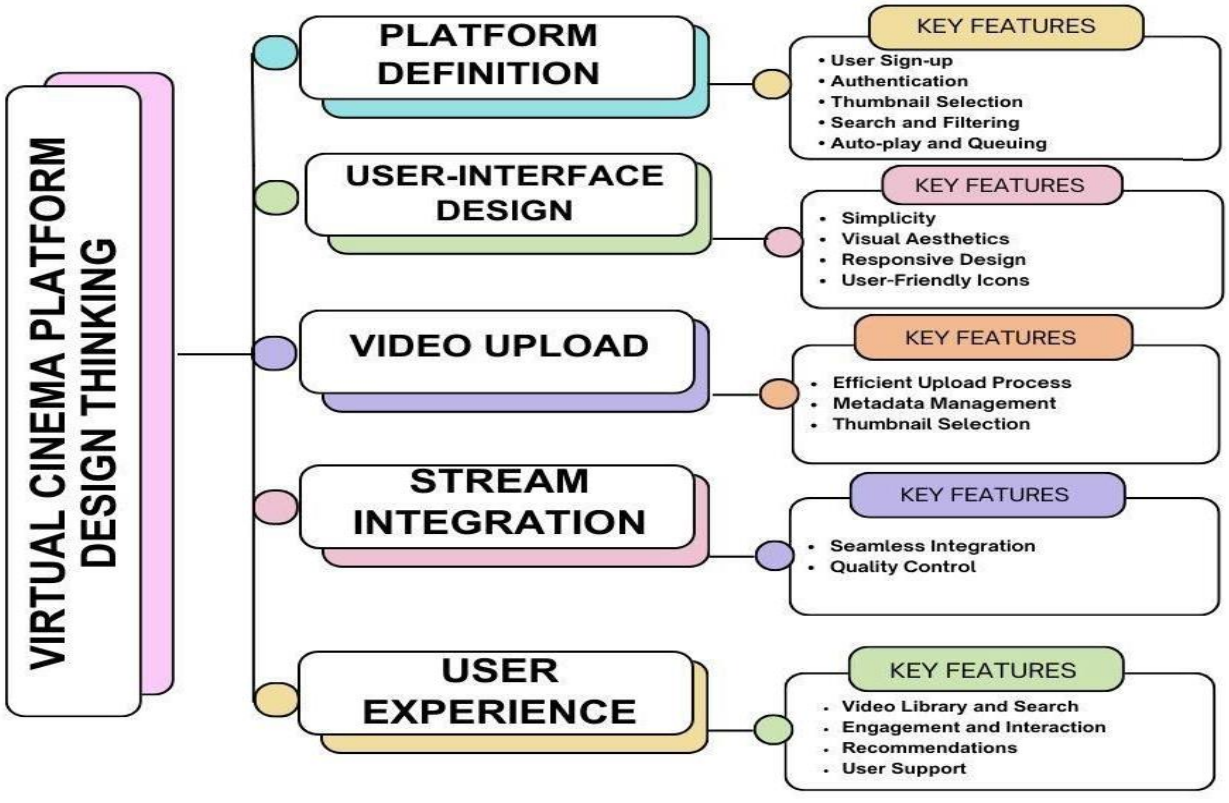
Problem Objective:

Create a virtual cinema platform using IBM Cloud Video Streaming for on-demand movie and video streaming. Define the platform, design a user-friendly interface, integrate IBM Cloud Video Streaming, enable on-demand playback, and prioritize a seamless and immersive cinematic experience. Ensure compliance with legal requirements and optimize for various devices.

Design Thinking:

- Platform features: user registration, video upload, on-demand streaming.
- Prioritize secure authentication, efficient video upload, and a seamless playback experience.
- Design a user-centered interface emphasizing simplicity and responsiveness. Optimize video upload for efficiency and metadata management.
- Seamlessly integrate IBM Cloud Video Streaming, ensuring quality control and adaptive streaming.
- Enhance user experience with engagement features, recommendations, and system assistance.

Flowchart



Introduction:

Transforming the design into a Cloud Media Streaming Platform involves development, testing, and deployment stages.

Innovative solutions:

Incorporate user-generated playlists and real-time chat for enhanced user engagement in the Cloud Media Streaming Platform.

1. User-Generated Playlists

Development Steps:

1. Playlist Creation Tools:

-Use frontend components (UI, Drag & Drop, Preview) for playlist creation and customization. Implement backend services (Database, APIs for CRUD, Authentication) to manage and store playlists.

2. Collaborative Playlist Editing:

- Implement real-time updates (e.g., Socket.io) to reflect changes made by multiple users.

3. Real-Time Chat

Development Steps:

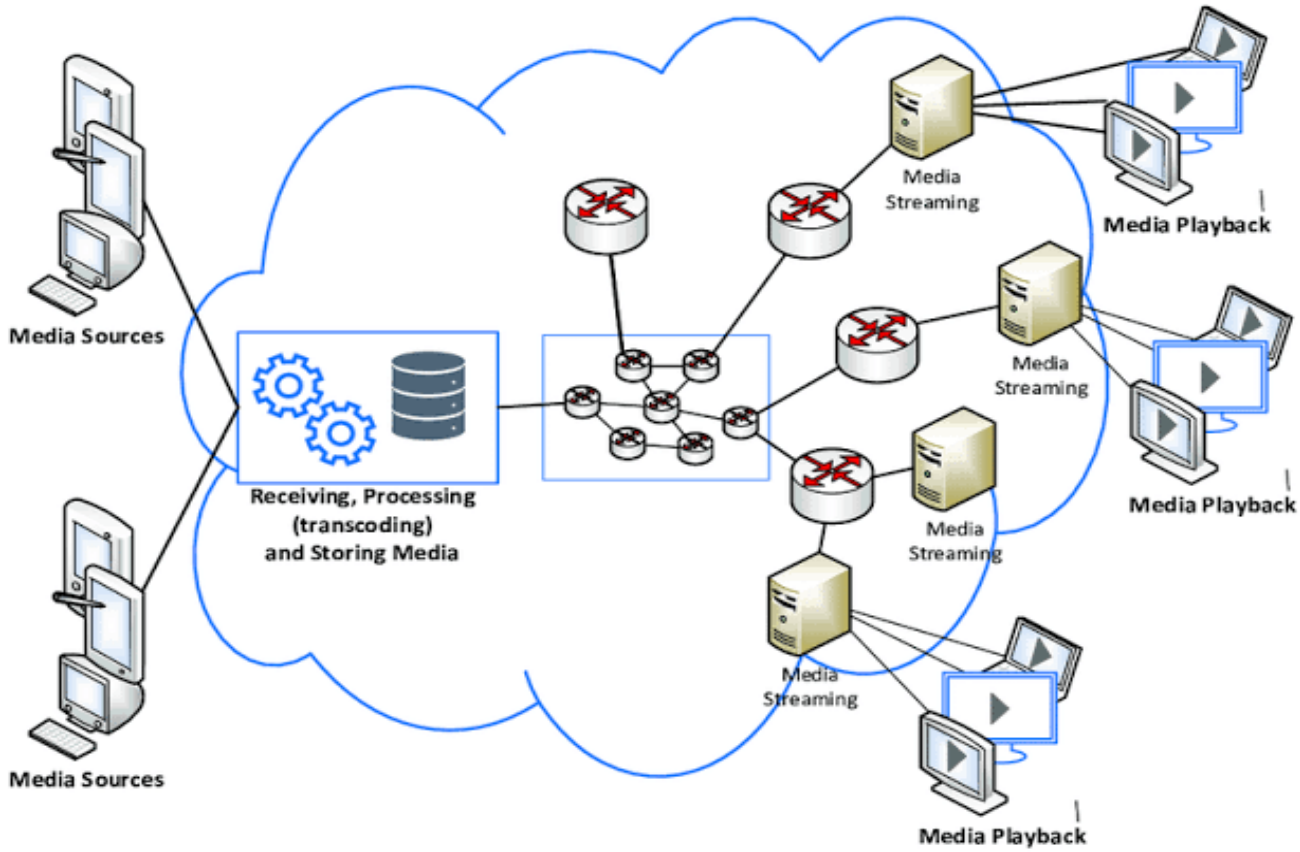
1. Chat Interface:

Developing a real-time chat interface using an SDK for one-on-one and group chat. Implement chat rooms for different user groups and communities.

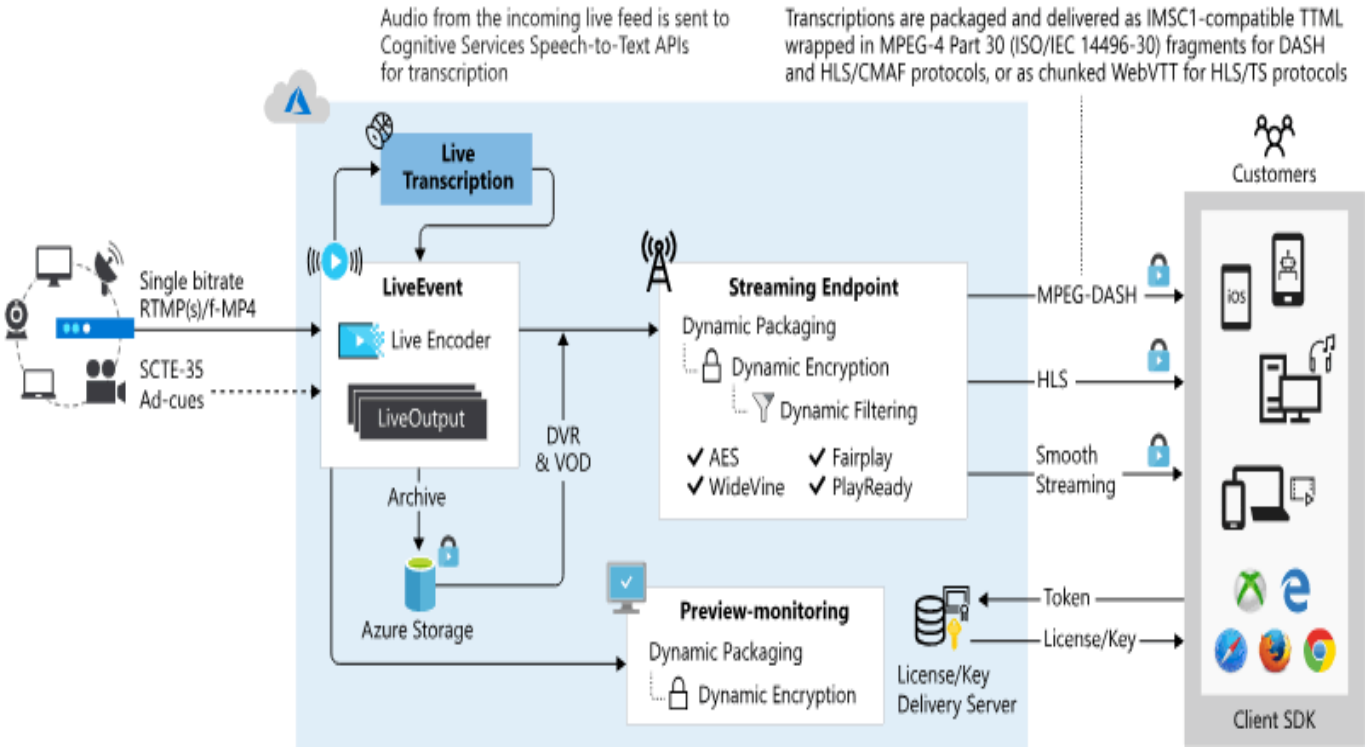
2. Chat Moderation Tools:

Implement moderation tools (Content Moderation API, Reporting, Administrator dashboard) for a positive chat environment. Develop reporting mechanisms for inappropriate content.

Integrate innovative solutions into the Cloud Media Streaming Platform with a strategic implementation aligned with design thinking principles.



Commonly, video streaming services use adaptive streaming algorithms like **HTTP Live Streaming (HLS)** or **Dynamic Adaptive Streaming over HTTP (DASH)**. These algorithms dynamically adjust the quality of the video based on the viewer's internet connection and device capabilities to ensure smooth playback.



It also provides features for content protection, analytics, and integration with other IBM Cloud service.

Start Building the virtual cinema platform using IBM Cloud Video Streaming. By defining Platform definition and User Interface Design. Also discuss about the User Registration and Authentication.

Platform Definition and User Interface Design:

Here, we consider the simpler platforms to build the respective application as,

- Using Python and Flask for the Backend,
- Using HTML and CSS for the Frontend,
- Using JavaScript for handling client-side interactions.

Platform Features:

The below features are basics to a Video Streaming Application,

1. **User Registration and Authentication** - Authentication is handled using IBM Cloud Video Streaming credentials and optionally integrated with a user authentication system.

2. **Video Streaming and Management** - Supports live video streaming and on- demand playback and Video metadata, such as title, description, and thumbnail, can be edited.

3. **Real-Time Chat** - Integrated real-time chat for users to engage in discussions while watching videos to Enhances the social aspect of the virtual cinema experience.

4. **On-Demand Playback** - With On-Demand, viewers can pause, play, fast-forward, rewind, and re-watch the show On-Demand as much as they would like to watch.

5. **User Feedback** - Provides feedback messages for successful actions or errors.

Intuitive User Interface Design:

For an Intuitive User Interface to the Virtual Cinematic Platform, we list out some ideas to give better User Interface (UI).

First, we build our application into pages which includes:

- **Home Page**

  - contains Navigation links (My Videos, Playlists, Profile, Logout)

  -Thumbnails of Videos selection allows users to explore videos by categories or genres.

- **Video Playback Page**

Displays the video player with controls for play, pause, volume, and full-screen mode. Includes video title, description, and uploader information. Real-time chat panel for users to engage in conversations.

- **User Profile Page**

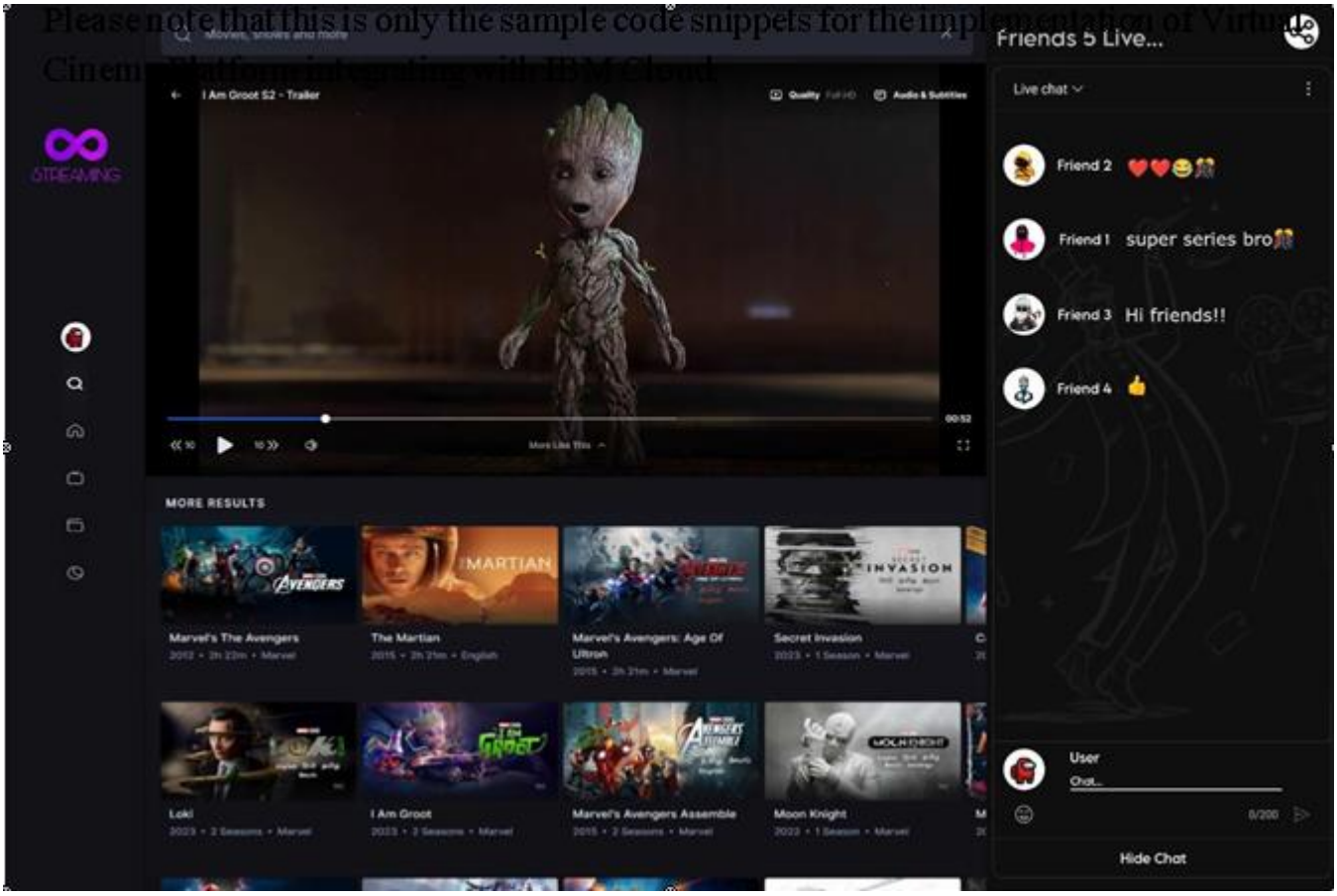
Includes Profile information's like user statistics, such as the number of uploaded videos and playlists. User's playlists with the option to create, edit, or delete.

- **Upload Video Page**

Allows users to select a video file for upload which includes fields for title, description, and thumbnail selection.

- **Playlist Page**

Allows users to add, view or manage videos to existing playlists or create new ones. Our Design Principles is to be simple and user-friendly navigation with a clear menu structure. Design a responsive layout for a seamless experience on different devices. Design intuitive controls for video playback and interaction like engaging visuals such as thumbnails and cover pages. Mainly to provide feedback messages and visual cues for user actions.



We creating this application by HTML and CSS for a few key pages in the virtual cinema platform: the home page, the video playback page, the user profile page, and the upload video page.

Please note that this is only the sample code snippets for the implementation of Virtual Cinema Platform integrating with IBM Cloud.

User Registration and Authentication:

In this development part we set up user registration and authentication mechanisms to ensure secure access to the platform. As we referred from the web resource here are the common procedures.

UserRegistrationandAuthenticationSetupProcedure

1. Package Installation:

- Begin by installing the required packages for your Flask application. Execute the following command in your terminal or command prompt:

Bash

```
pip install Flask Flask-Login Flask-WTF Flask-Bcrypt
```

2. Flask App Configuration:

- Create a new file named app.py to configure your Flask application. Import necessary modules, set a secret key, and define your database configuration. Here's a simplified snippet:

Python

```
from flask import Flask, render_template, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user,
login_required, logout_user, current_user
from flask_bcrypt import Bcrypt
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Length, EqualTo

app = Flask(   name   )
app.config['SECRET_KEY'] = 'your_secret_key' app.config
['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db' db = SQLAlchemy(app)
bcrypt = Bcrypt(app) login_manager = LoginManager(app) login_manager.login_view = 'login'
```

3. **User Model Definition:**

- Define the User model in python file to represent users in the database. Include fields like id, username, email, and password.

4. **Form Creation:**

- Create registration and login forms using Flask-WTF. These forms include fields for username, email, password, and confirm_password.

5. **User Registration Route:**

- Implement a route in app.py for user registration (/register). Handle form submissions, hash passwords with Flask-Bcrypt, and store user information in the database.

6. **User Login Route:**

- Set up a user login route (/login). Validate user credentials, log the user in using Flask-Login, and redirect to the main page upon successful login.

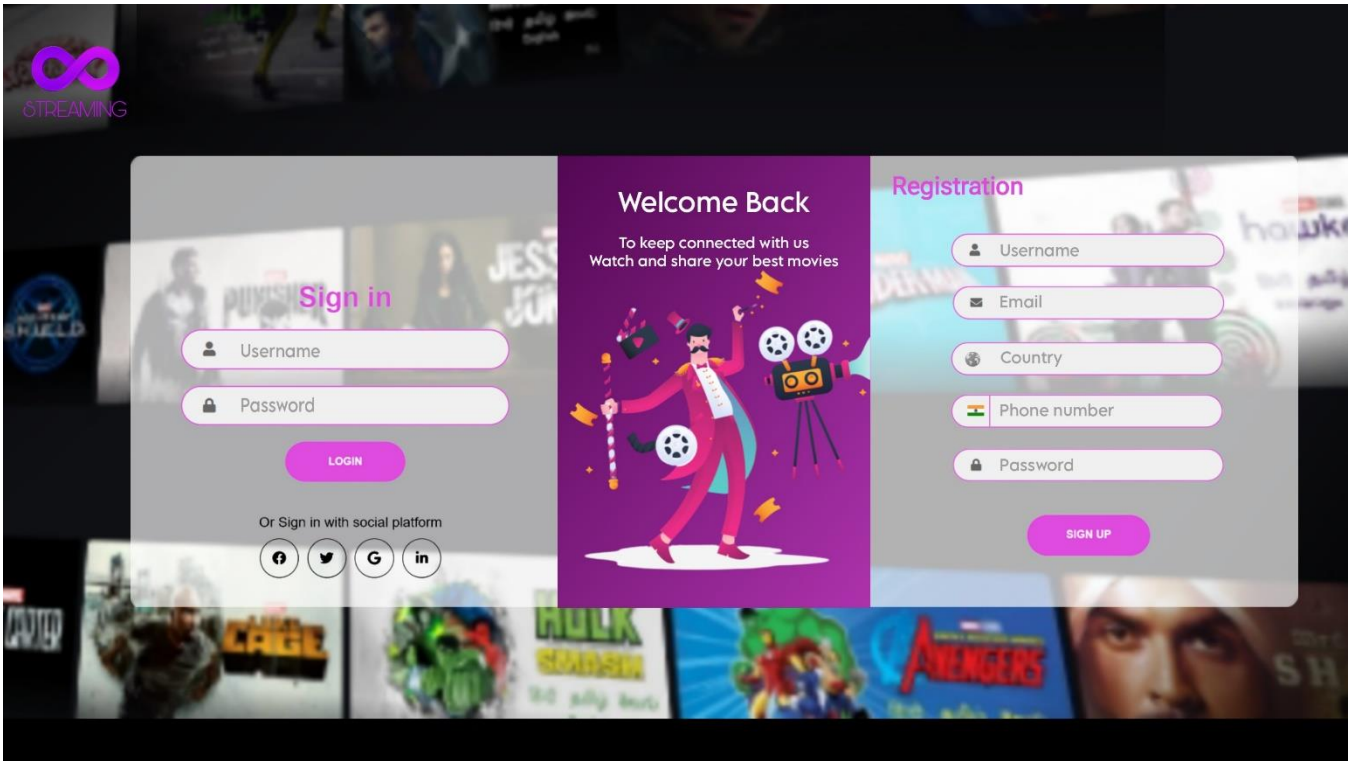
7. **User Logout Route:**

- Create a user logout route (/logout). Log the user out using Flask-Login and redirect to the main page.

8. **Application Execution:**

- Run your Flask application using the command flask run in the terminal or command prompt. Access the application at http://127.0.0.1:5000/.

By following these simplified steps, users can seamlessly register, log in, and log out of your Flask application. Customize routes, templates, and additional features based on your specific application requirements



Continue building the platform by integrating video streaming services and enabling on-demand playback. To enhance the virtual cinema platform, we'll integrate video streaming services and enable on-demand playback. We'll use Flask and HTML to achieve by follow the given steps below,

1. **Environment Setup:**
   - Ensure you have Python and pip installed on your system.
   - Create a virtual environment: `python -m venv venv`
   - Activate the virtual environment:
     - On Windows: `venv\Scripts\activate`
     - On Unix or MacOS: `source venv/bin/activate`
2. **Install Dependencies:**
   - Install Flask and required packages:

Bash

```
pip install Flask Flask-WTF Flask-Login Flask-SocketIO
```

3.Create Flask App

4. Set up a basic Flask app with a secret key and SocketIO support.

5.Define Video Model:(using Flask-SQLAlchemy)

6.Create Video Upload Form:(using Flask-WTF.)

7.**Implement Video Upload Route**

8.Implement Video Streaming Route

9.Create Video Streaming HTML Template

10.Run the Application

Bash

```
Flask Run
```

By following these steps, your virtual cinema platform now supports video streaming with on-demand playback. Users can upload videos, and the platform displays a list of available videos with playback options.

Let us define Implement the functionality for users to upload their movies and videos to the platform in detail.

Movie and Video Upload Implementation

To enable users to upload movies and videos to your virtual cinema platform, follow these straightforward steps:

1. Update Video Model:

In python file add a method to the Videomodel for retrieving the video file path:

Python

```
class Video(db.Model):
    # ... existing fields ...

    def get_video_path(self):
        return os.path.join(app.config['VIDEO_UPLOAD_FOLDER'],
        self.filename)
```

2. Create Video Upload Form:

define a simple form for video uploads:

Python

```
from flask_wtf import FlaskForm
from wtforms import StringField, TextAreaField,
FileField, SubmitField from wtforms.validators import
DataRequired

class VideoUploadForm(FlaskForm):
    title = StringField('Title',
    validators=[DataRequired()]) description =
    TextAreaField('Description')
    video = FileField('Video File',
    validators=[DataRequired()]) submit =
    SubmitField('Upload')
```

3.Update Video Upload Route:

create a route to handle video uploads: **Python**

```
from werkzeug.utils import secure_filename
import os

 # ... existing code ...
 @app.route('/upload_video', methods=['GET', 'POST'])
 @login_required
def upload_video():
     form = VideoUploadForm()
     if form.validate_on_submit():
       video_file = form.video.data
       filename = secure_filename(video_file.filename)
      video_file.save(os.path.join(app.config['VIDEO_UPLOAD_FOLDER'],
      filename))
       video = Video(title=form.title.data,
      description=form.description.data, filename=filename,
      user_id=current_user.id)
      db.session.add(video)
      db.session.commit()
      flash('Video uploaded successfully!', 'success')
return redirect(url_for('dashboard'))
       return render_template('upload_video.html', form=form)
```

4.Create Video Upload HTML Template:

Create a template for video uploads:

Html

```
<!-- templates/upload_video.html -->
{% extends 'layout.html' %}
{% block content %}
    <h2>Upload Video</h2>
    <form method="post" enctype="multipart/form-
    data">
        {{ form.hidden_tag() }}
        <label for="title">Title:</label>
        {{ form.title() }}
        <label for="description">Description:</label>
        {{ form.description() }}
        <label for="video">Video File:</label>
        {{ form.video() }}
        <button type="submit">Upload</button>
    </form>
{% endblock %}
```

Run the Application:

Bash

```
Flask Run
```

Enabling on-demand playback

Continuing from the previous steps, now let's focus on enabling on-demand playback for the uploaded videos. We'll create a new route that allows users to view and play the uploaded videos.

Create Video Playback Route:

- add a route for on-demand playback. This route should render a template with a video player:

Python

```
# app.py
@app.route('/playback/')
def playback(filename):
      video = Video.query.filter_by(filename=filename).first()
      return render_template('playback.html', video=video)
```

Create Video Playback HTML Template :

- Design a simple template for video playback with a video player element:

Html

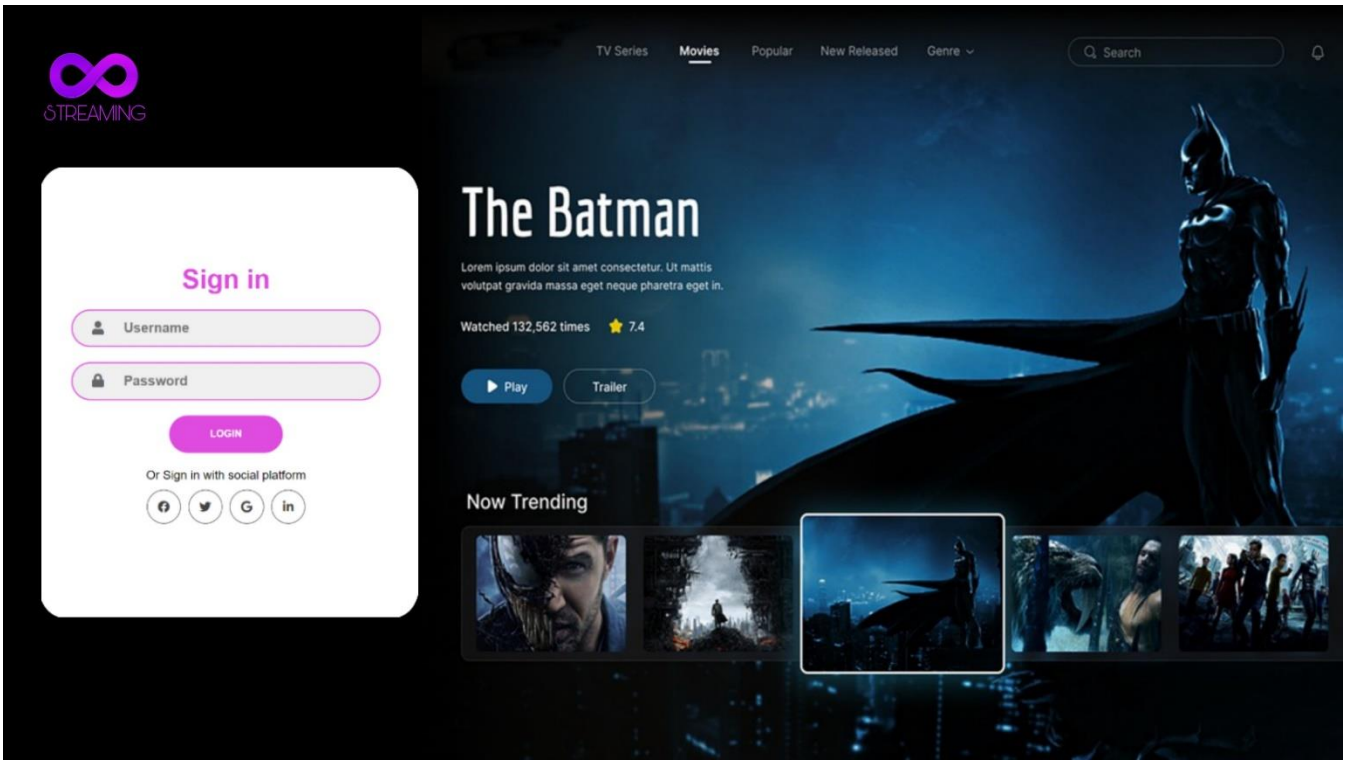
```
<!-- templates/playback.html -->
{% extends 'layout.html' %}
{% block content %}
                    <h2>{{video.title}}</h2>
            <video width="800" controls>
                <source src="{{ url_for('static',
       filename='videos/' + video.filename) }}"
       type="video/mp4">
                Your browser does not support the video tag.
            </video>
{% endblock %}
```

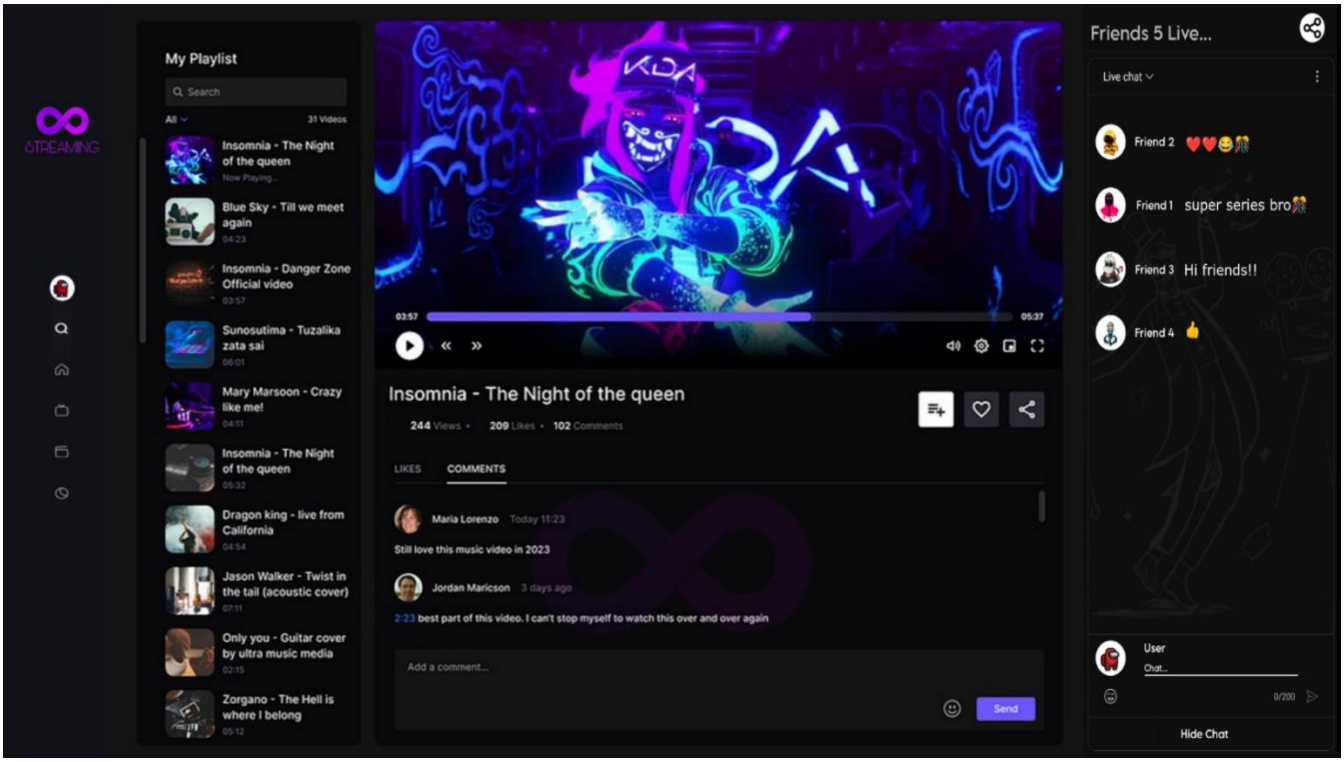
Run the Application:

Bash

```
Flask Run
```

Video Streaming Application:

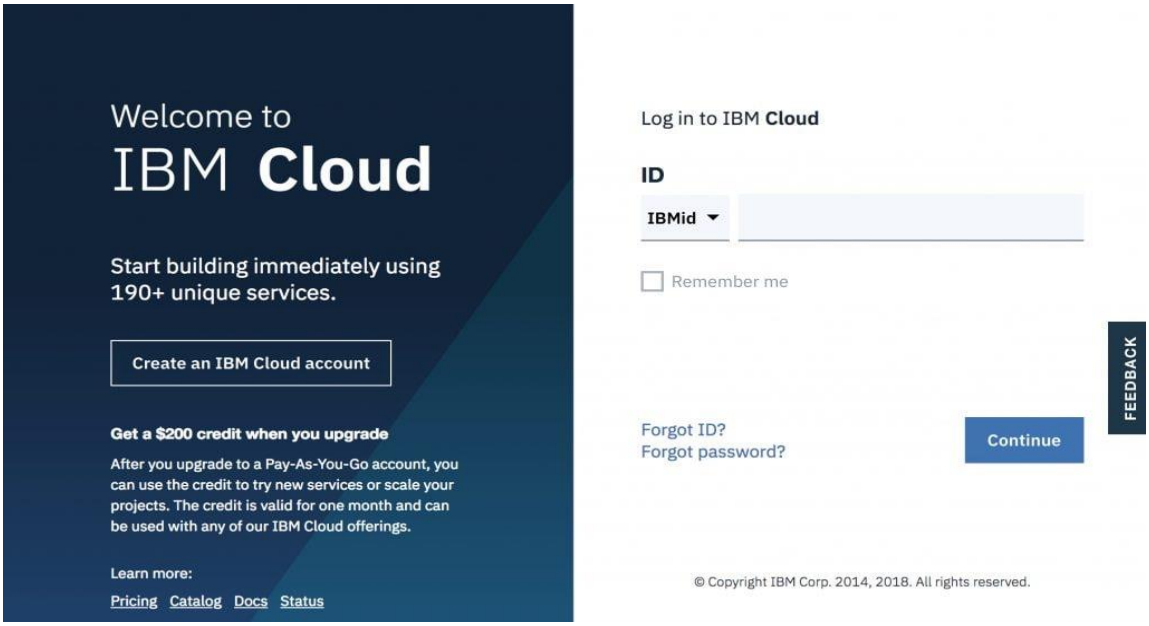




Integrating IBM Cloud Video Streaming services with Cloud Foundry involves configuring your Cloud Foundry application to interact with the IBM Cloud Video API for video management and playback. Here are the procedural steps:

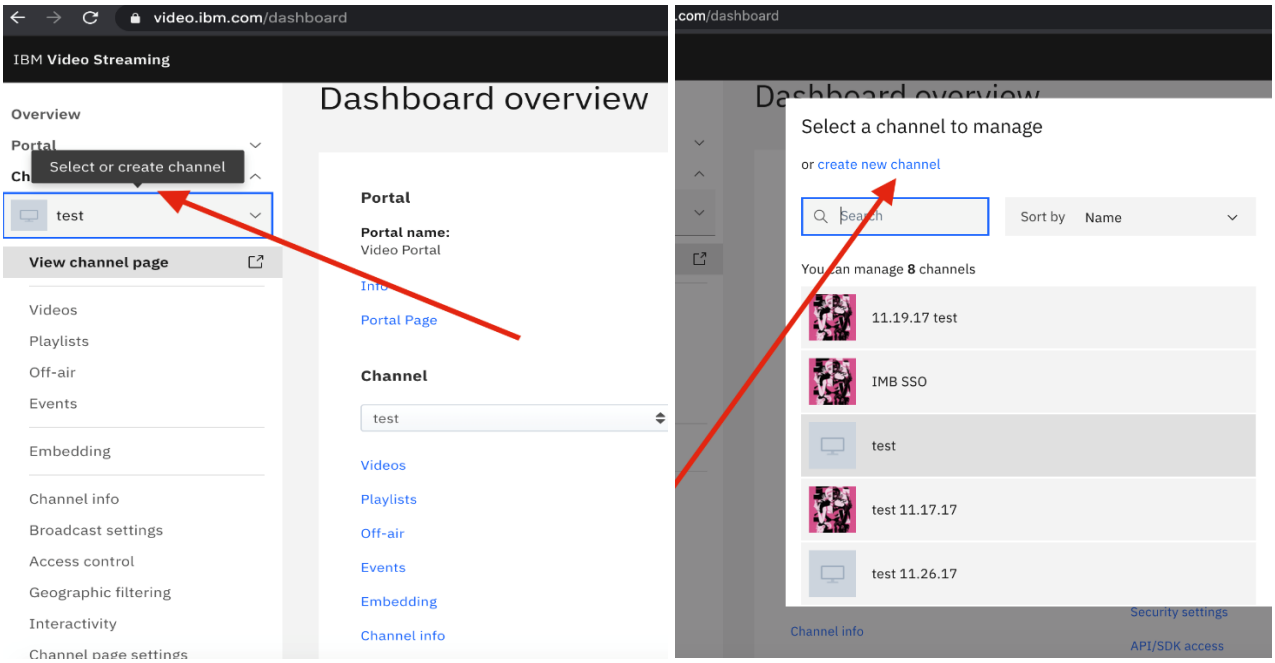
1. IBM Cloud Account Setup:

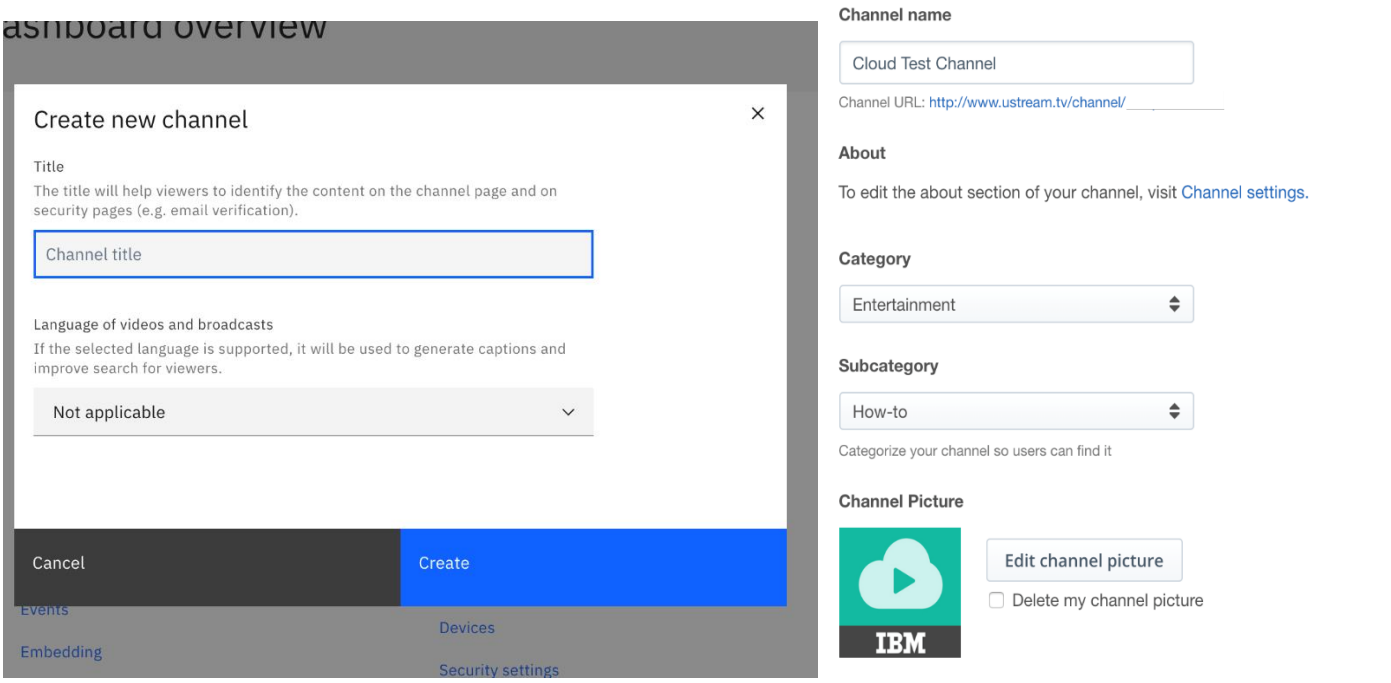
- If not already done, sign up for an IBM Cloud account.



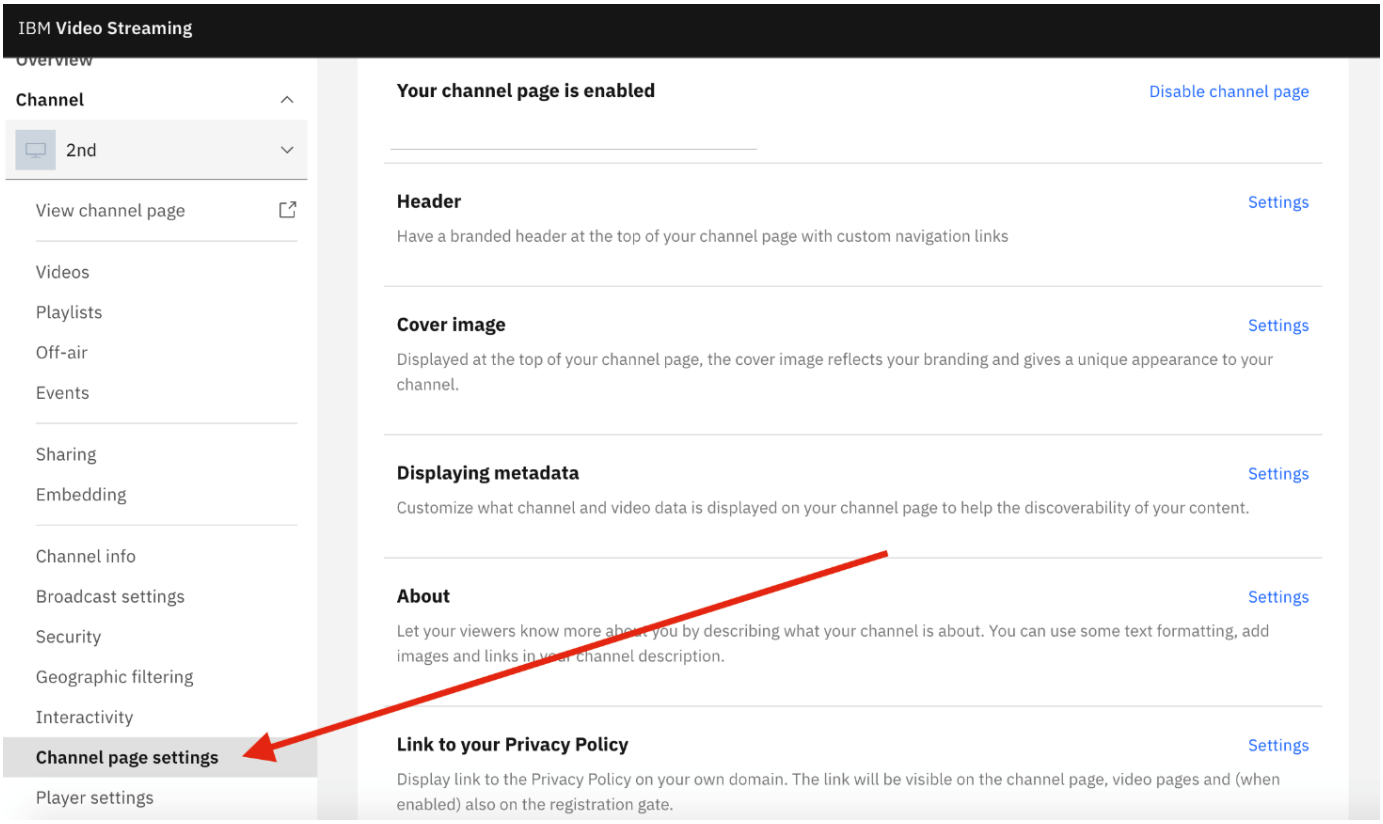
2. Create a Streaming Channel:

- Log in to your IBM Cloud Dashboard.
- Navigate to the IBM Cloud Video Streaming service.
- Create a new streaming channel. This channel will represent your virtual cinema within the IBM Cloud Video service.



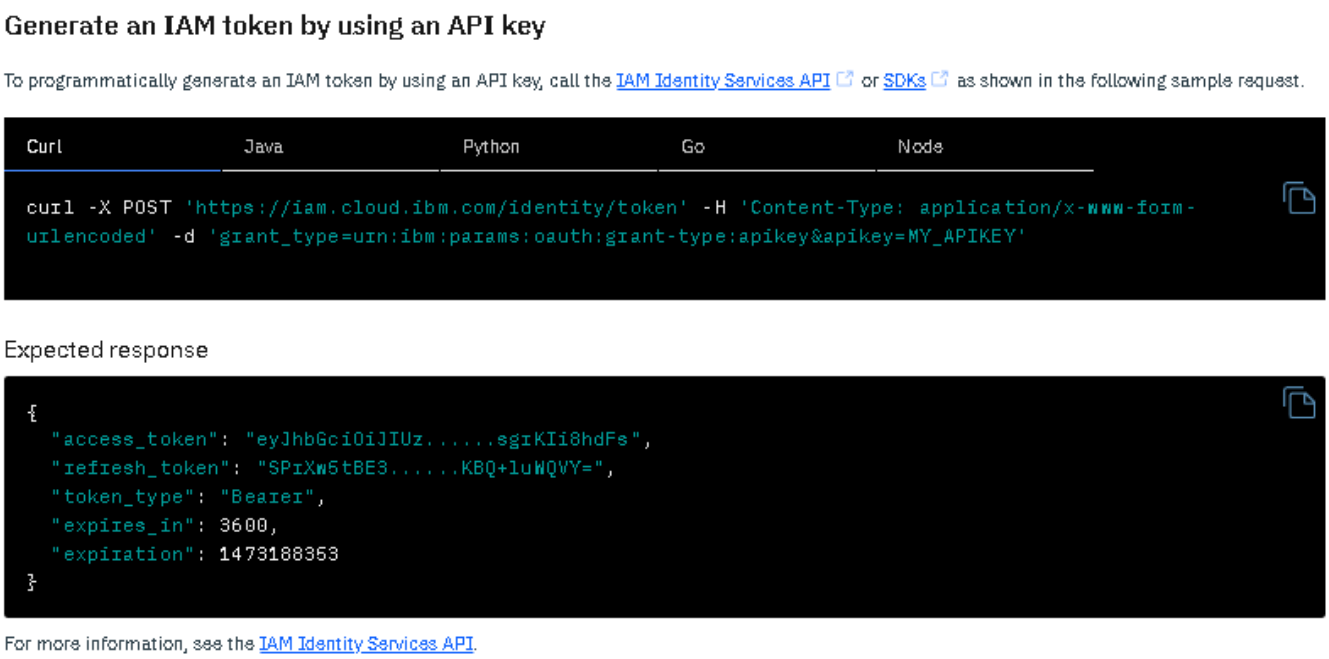


Once you have created your new channel or wish to update an existing channel, click on the Channel Page settings tab in your channel settings dashboard in order to start customizing your channel.



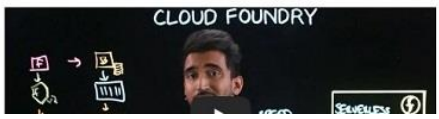
3. Get API Key and Access Token:

- Obtain an API key and access token from the IBM Cloud Dashboard. These credentials will be used to authenticate your requests to the IBM Cloud Video Streaming API.



4. Create a Cloud Foundry Application:

- Install the Cloud Foundry CLI if you haven't already.
- Log in to your Cloud Foundry account using the CLI.



5. Configure Application for IBM Cloud Video Streaming:

- Modify your Cloud Foundry application code to interact with the IBM Cloud Video Streaming API. Use the API key and access token for authentication.

6. Bind IBM Cloud Video Service to Your Cloud Foundry App:

- Use the Cloud Foundry CLI to bind your Cloud Foundry application to the IBM Cloud Video service instance:

Bash

```
cf bind-service YOUR_APP_NAME IBM_CLOUD_VIDEO_SERVICE_INSTANCE_NAME
```

guide for integrating IBM Cloud Video Streaming services with a Flask application using Docker and Cloud Foundry:

Using Docker and GitHub:

1. **Create a Flask Application:**

   Create a simple Flask application that interacts with the IBM Cloud Video Streaming API. Here's an example using Flask and the requestslibrary:

Python

```
From flask import Flask, jsonify import requests

app = Flask(__name__)

 @app.route('/getVideoDetails')
 def get_video_details():
      api_key = 'your-api-key'
      access_token = 'your-access-token'
       channel_id = 'your-channel-id'

       url =
f'https://api.video.ibm.com/v4/channels/{channel_id}/videos'
      headers = {'Authorization': f'Bearer
{api_key}:{access_token}'}

      response = requests.get(url, headers=headers)
       data = response.json()

       return jsonify(data)
 if __name__ == '__main__':
      app.run(host='0.0.0.0', port=5000)
```

- **Create a Dockerfile:**

  Create a Dockerfile in the root of your project:

Dockerfile

```
FROM python:3.9
 WORKDIR /usr/src/app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "./app.py"]
```

  Create a requirements.txt file listing your Flask and requests dependencies.

- **Push to GitHub:**

  Push your Flask application, Dockerfile, and requirements.txtto a GitHub repository.

- **Build Docker Image:**

  Build your Docker image locally or on a CI/CD platform and push it to a container registry like Docker Hub.

Bash

```
docker build -t your-docker-username/your-image-name:tag .
docker push your-docker-username/your-image-name:tag
```

- **Cloud Foundry Deployment:**

  Create a manifest.yaml file in the root of your project:

Yaml

```
applications:
      - name: your-app-name
       memory: 512M
       instances: 1
       random-route: true
      docker:
           image: your-docker-username/your-image-
      name:tag
```

Deploy your application to Cloud Foundry:

Bash

```
Cf push
```

7. Restage Your Cloud Foundry App:

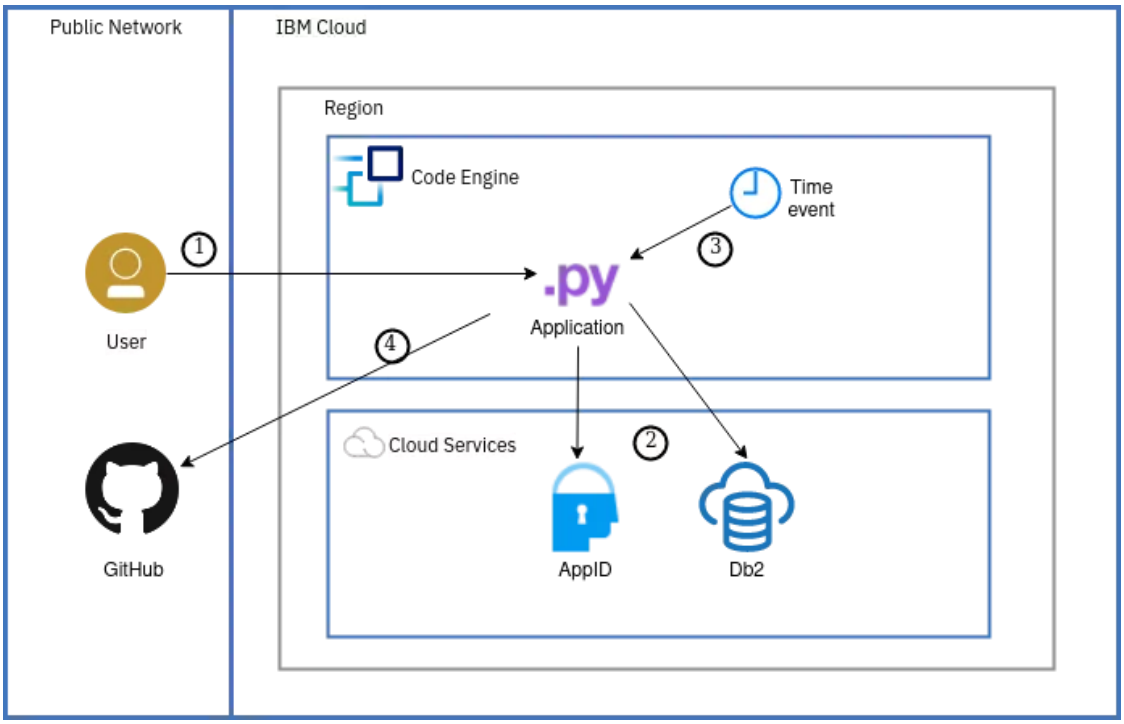
- Restage your application to apply the changes:

Bash

```
Cf restage YOUR_APP_NAME
```

8. Access Your Cloud Foundry Application:

- Open your web browser and navigate to the URL assigned to your Cloud Foundry application.



As you all know that IBM Cloud® is announcing the full deprecation of IBM Cloud Foundry on June 1, 2023. At that time, any IBM Cloud Foundry application runtime instances running IBM Cloud Foundry applications will be permanently disabled and deprovisioned. See the deprecation details for specific implications.

Since all IBM Cloud Foundry users need to decide where to move their Cloud Foundry workload applications prior to the IBM Cloud Foundry "End-of-Support" date, review the following migration checklist and IBM Cloud® compute service options in this announcement to help you decide which IBM Cloud® application hosting service is right for your organization. IBM Cloud® users have a choice of several modern application hosting runtime platforms on the IBM Cloud®.

- IBM Cloud Code Engine is a fully managed, serverless platform that runs your containerized workloads, including web apps, micro-services, event-driven functions, or batch jobs.
- IBM Cloud® Kubernetes Service is an open source platform for managing containerized workloads and services across multiple hosts, and offers management tools for deploying, automating, monitoring, and scaling containerized apps with minimal to no manual intervention.
- Red Hat OpenShift on IBM Cloud is a Kubernetes container platform that provides a trusted environment to run enterprise workloads.

Migrating applications

Review the suggested steps for migrating your IBM Cloud Foundry applications to a new IBM Cloud® compute service.

Step 1: Investigate & review your current IBM Cloud Foundry application deployments

Step 2: Investigate which IBM Cloud® compute services on which to host your new application workload

Step 3: Deploy your application to your chosen IBM Cloud® compute services

Step 4: Balance your incoming application traffic between the IBM Cloud® compute types

Step 5: Shutting down IBM Cloud Foundry applications

Conclusion:

Here is the procedure for integrating IBM Cloud Video Streaming services with Cloud Foundry for Virtual Cinema Platform application. Test the integration to ensure that video playback is smooth and that the integration with IBM Cloud Video Streaming is functioning correctly. Implement monitoring tools to track the performance of your Cloud Foundry application, addressing any issues promptly. Regularly update your application based on user feedback and evolving requirements.

Ensure that sensitive information, such as API keys and access tokens, is handled securely. Implement appropriate access controls and encryption measures. Make sure to refer to the official documentation of Cloud Foundry and IBM Cloud Video Streaming for detailed information and updates.