CLOUD APPLICATION AND DEVELOPMENT(CAD)

Project: **Media Streaming with IBM Cloud Video Streaming**

Phase 4: **Development Part 2**

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:

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

Bash

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

```
Flask Run
```

Bash

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)
```

3. 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

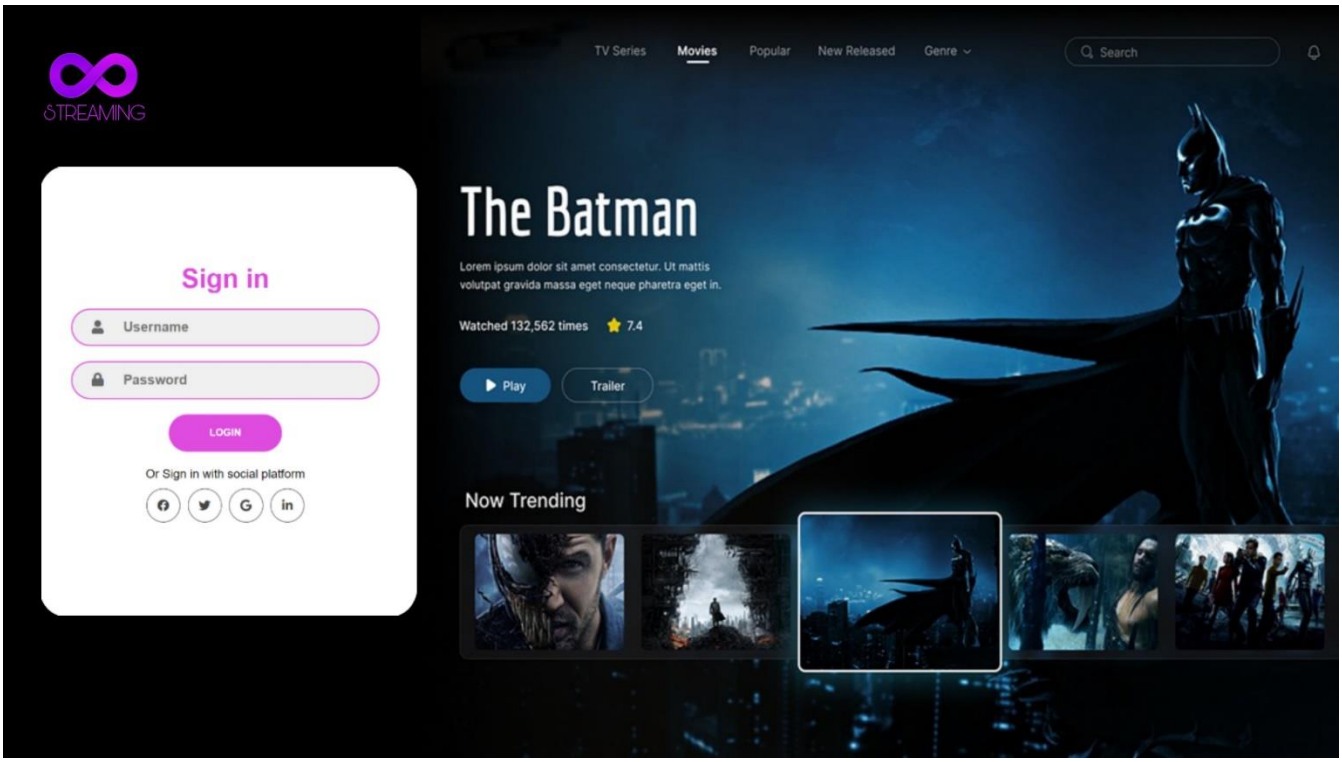
```
<!-- templates/playback.html -->
{% extends 'layout.html' %}
{% block content %}
                    <h2>{{video.title}}</h2>
            <video width="800" controls>
                <source src="{{ url_for('static',
        filename='videos/' + video.filename) }}"
        type="video/mp4">
    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

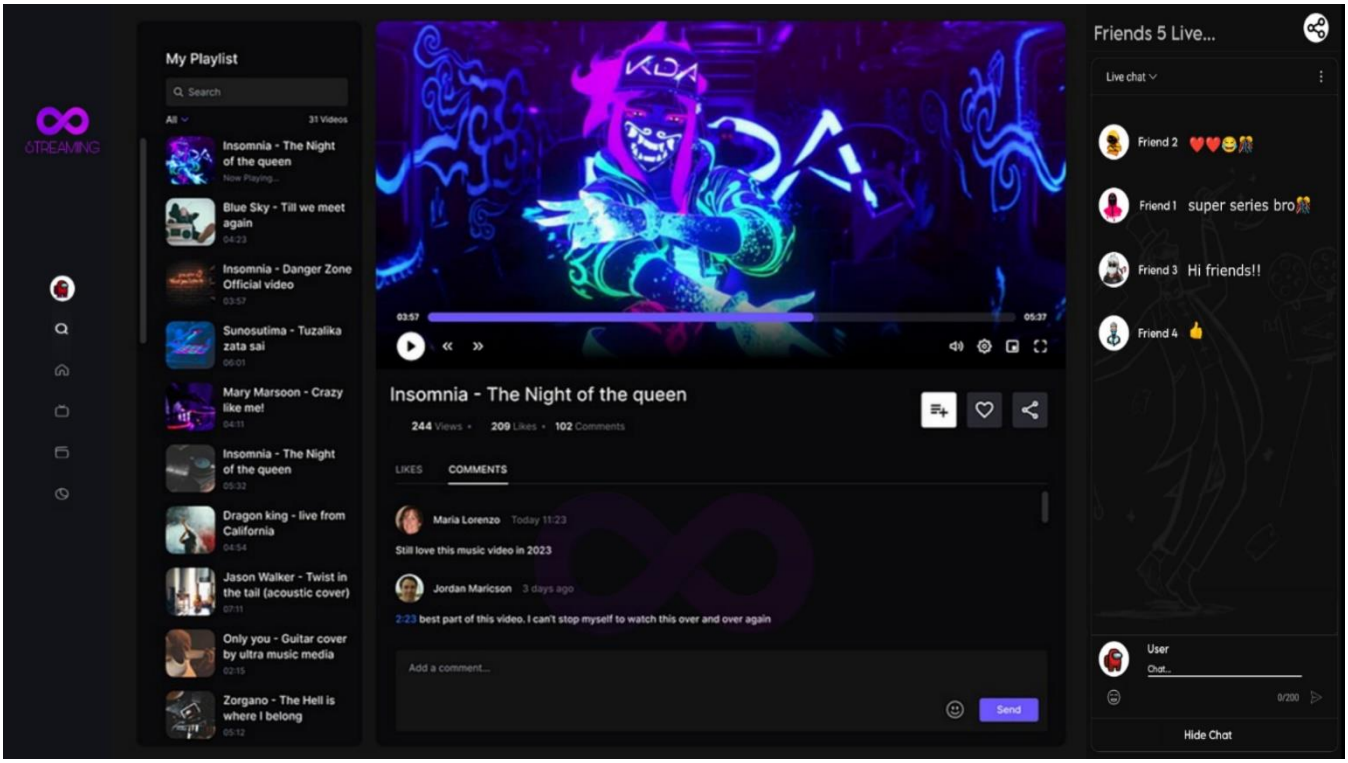
Run the Application:

Bash

```
Flask Run
```

Video Streaming Application:

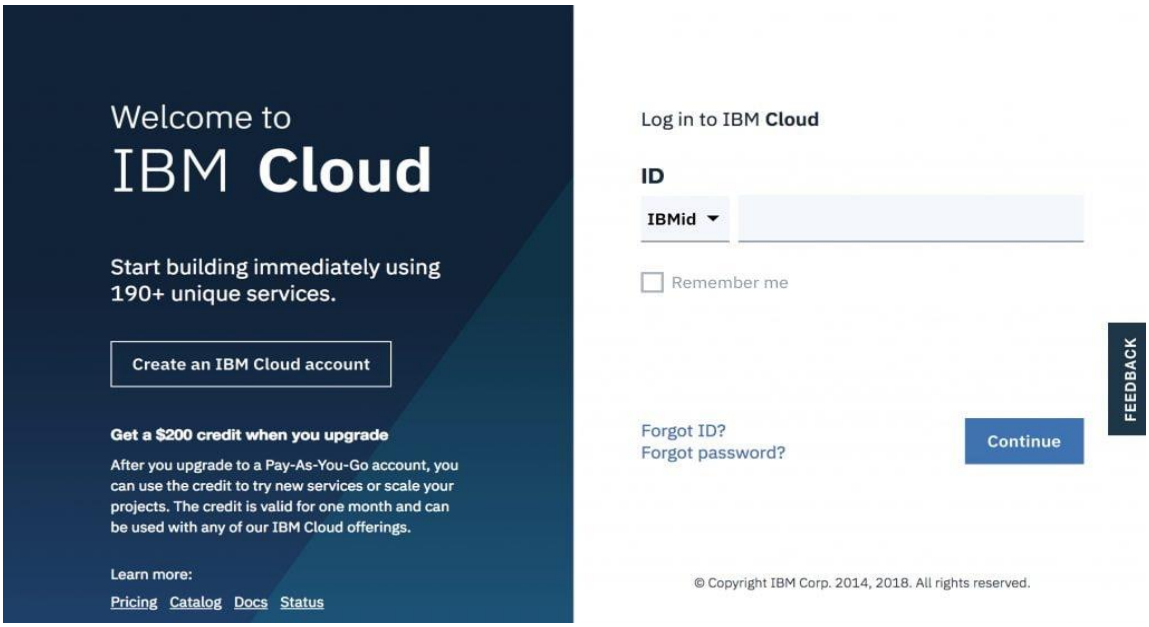




Integrating IBM Cloud Video Streaming services with Cloud Foundry involves configuring your Cloud Foundry application to interact with the IBM Cloud Video API for video management and playback. Here are the procedural steps:

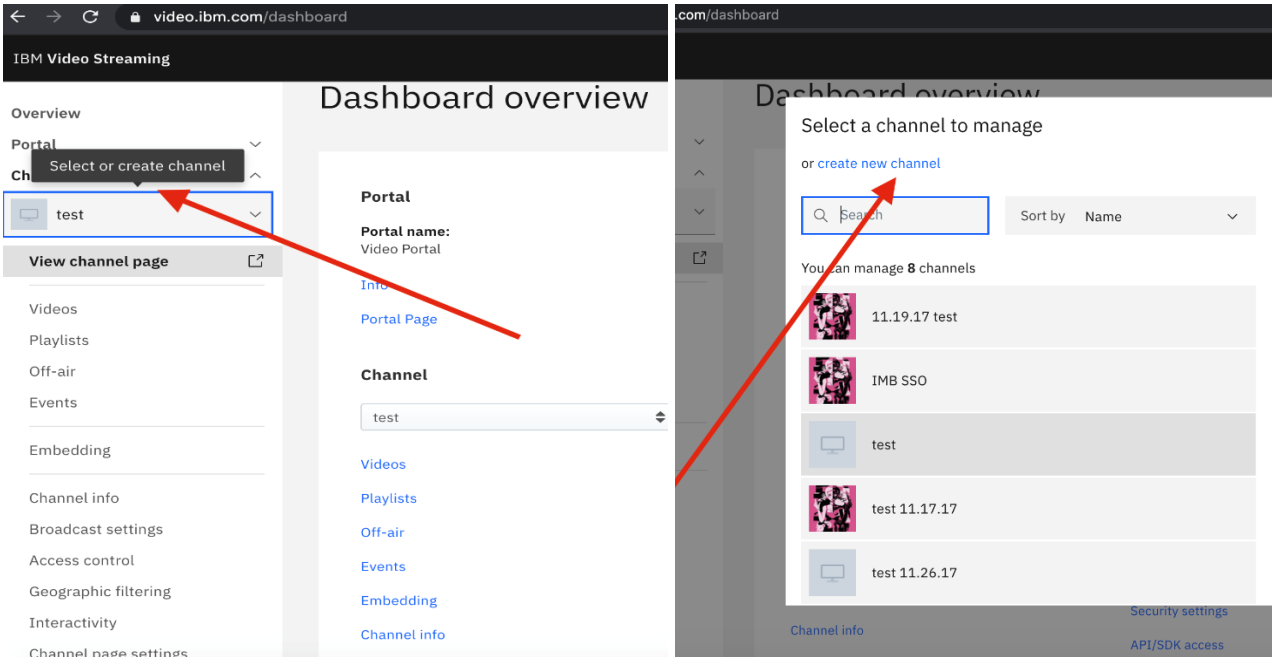
1. IBM Cloud Account Setup:

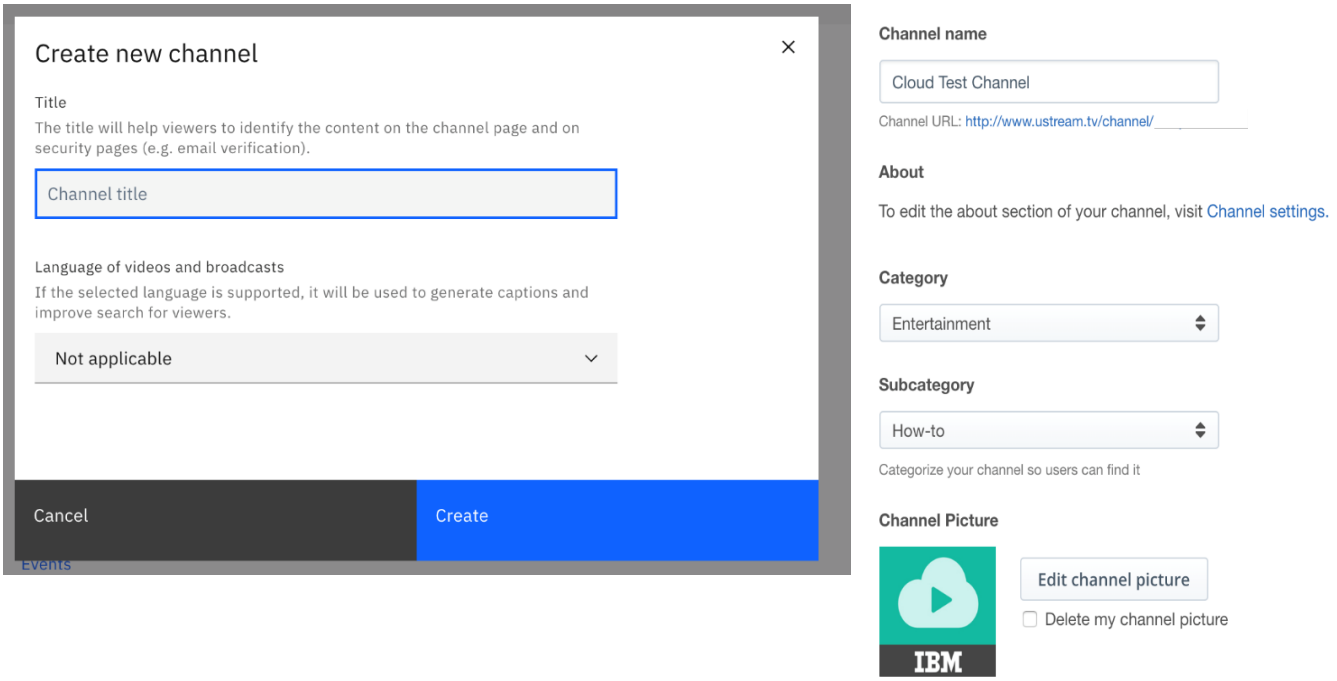
- If not already done, sign up for an IBM Cloud account.



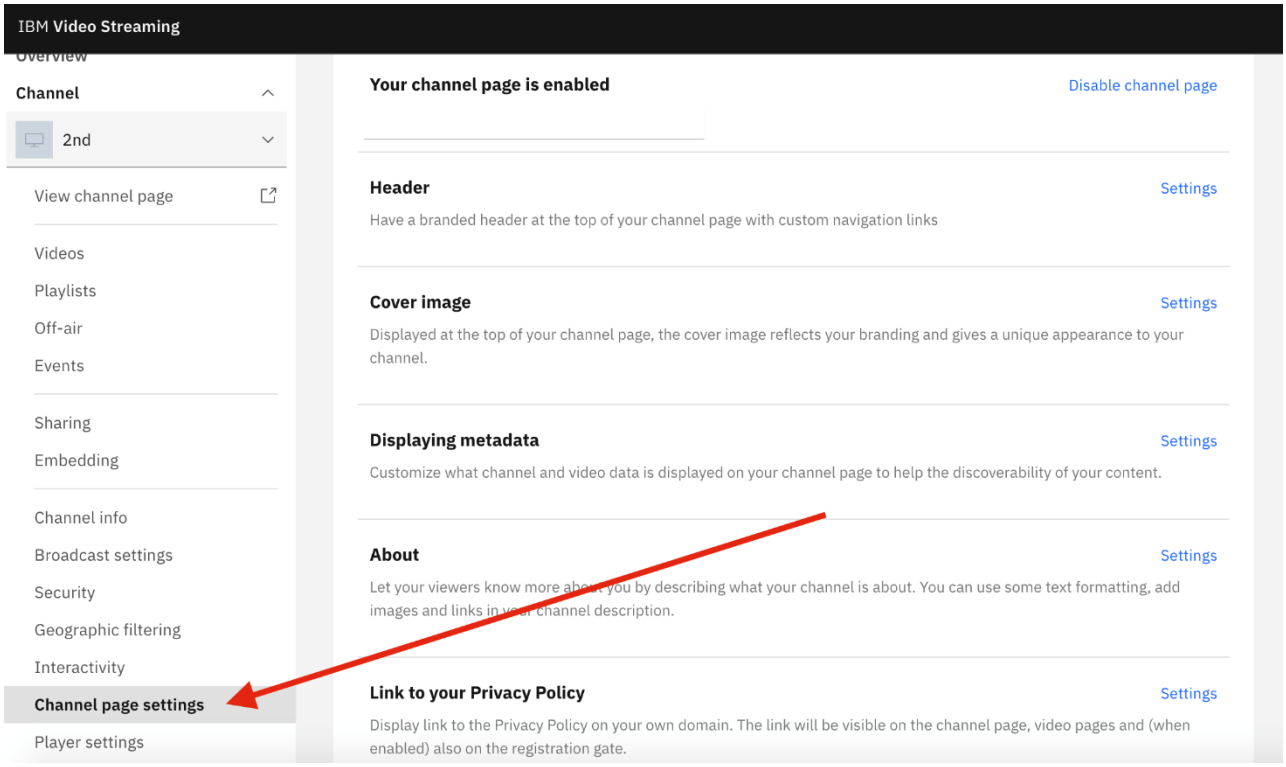
2. Create a Streaming Channel:

- 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.



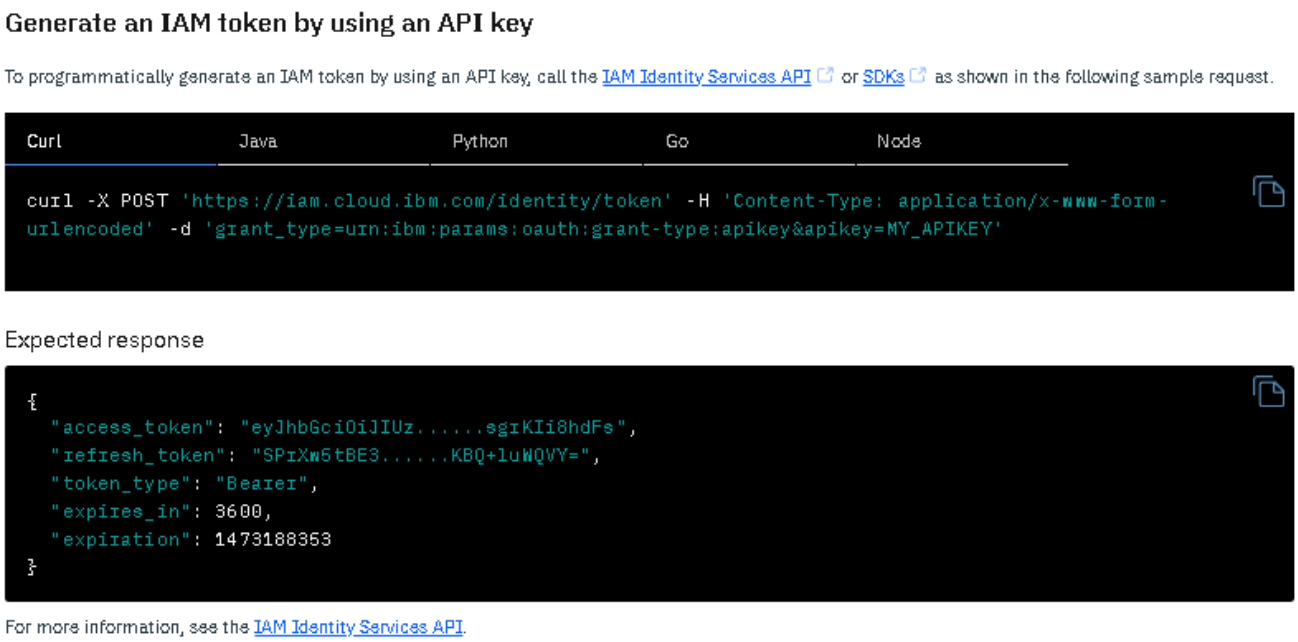


Once you have created your new channel or wish to update an existing channel, click on the Channel Page settings tab in your channel settings dashboard in order to start customizing your channel.



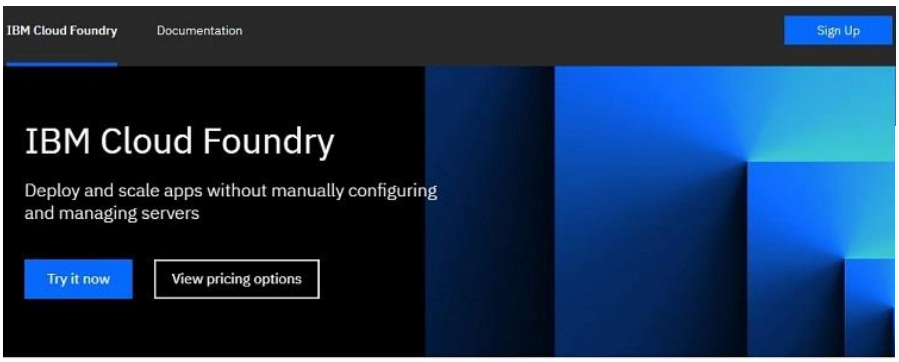
3. Get API Key and Access Token:

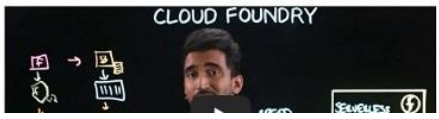
- Obtain an API key and access token from the IBM Cloud Dashboard. These credentials will be used to authenticate your requests to the IBM Cloud Video Streaming API.



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.yml 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:

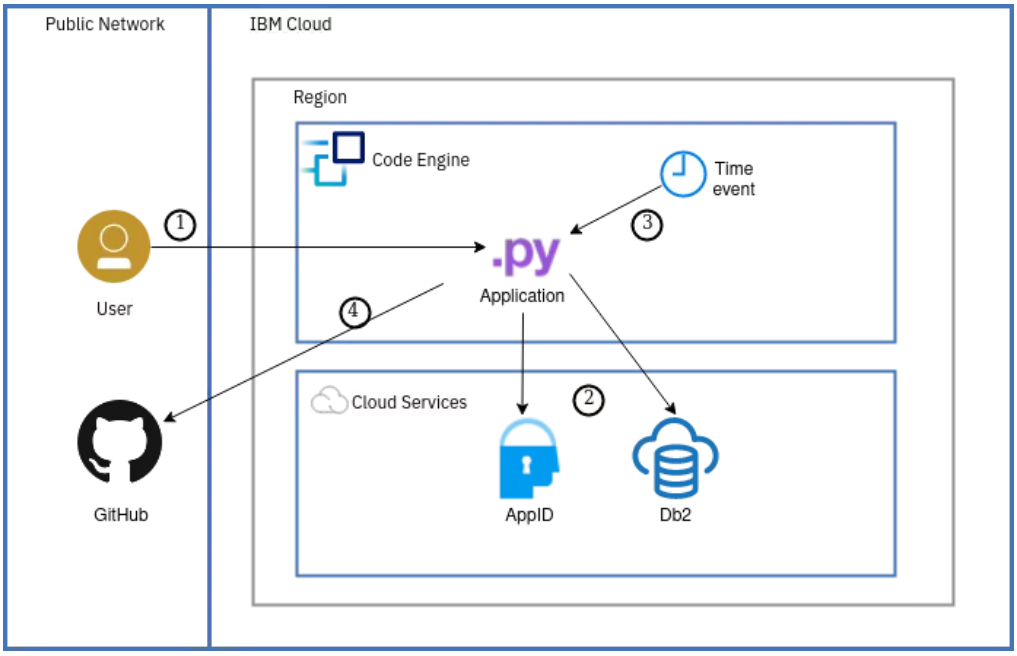
- Restage your application to apply the changes:

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.