**[Project Overview]**  
"Throughout this project, we’ll create a monitoring application in Python using Flask. We’ll start by building the app and then containerize it using Docker. We'll create a Dockerfile, build the image, and run the container locally."

"Once the application is containerized and running locally, we’ll move on to setting up an ECR — that’s Elastic Container Registry. We'll use Python's Moto 3 module to manage Docker images efficiently and securely in ECR."

"Then, it's time to deploy the application on Kubernetes. We’ll create an Elastic Kubernetes Cluster, or EKS, with nodes, and deploy the app to make it accessible from the internet. All of this will be done using Python."

"Before we start building, there are a couple of things you’ll need. First, you'll need an **AWS account** because we’ll be using AWS services for this project, like Kubernetes and ECR. Once you have your account, make sure you set up **programmatic access** so we can interact with AWS using the CLI or Python.

To set up programmatic access, go to the IAM section in AWS, and create your **access key** and **secret access key**." Now that you're ready, let's get started with the project!

Now that we've added some styling to the application, let’s take a look at the output. We’ve integrated a template into the index.html file, which presents the data in a more readable format. Here, I’ve set up the message variable to display the CPU and memory usage."

*Pause as the screen shows saving the file and running the app.*

"As you can see, once the application is refreshed, the CPU and memory values are displayed in a clear format: the CPU usage is shown as a percentage, and the memory usage follows below it. The values are dynamically updated in real-time with each refresh."

*Pause as you demonstrate the refresh and real-time data update.*

"This demonstrates the core functionality of the monitoring application — retrieving and displaying system performance data in an accessible format."

*Transition to Dockerization section.*

"Next, we need to containerize this application. Since the app is currently running locally on localhost, our goal now is to prepare it for deployment in a Docker container or on a Kubernetes cluster. Let’s walk through the steps to Dockerize this application."

Here’s a refined version of your demo video script, structured in clear paragraphs for better flow and readability:

---

### \*\*Demo Video Script – Cloud Native Python Application with Docker & Kubernetes\*\*

\*\*Duration:\*\* ~5 minutes

#### \*\*Introduction\*\*

Hi! I’m [Your Name], and today I’ll walk you through a project I built—a cloud-native Python application containerized with Docker and deployed on Kubernetes using AWS EKS. This project demonstrates my skills in DevOps, cloud computing, and automation, showcasing my ability to develop, deploy, and manage scalable applications. Let’s dive in!

---

#### \*\*Part 1: The Python Application\*\*

The core of this project is a Python Flask application that monitors CPU and memory usage. The app exposes an API endpoint (`/metrics`) to return system metrics in JSON format. I used libraries like `psutil` for system monitoring and `Flask` to build the web framework. The application is lightweight but serves as a perfect example of how to create a functional, metrics-driven service.

To ensure reproducibility, I documented all dependencies in a `requirements.txt` file. This step is critical for maintaining consistency across different environments, whether running locally or in production.

---

#### \*\*Part 2: Dockerizing the Application\*\*

To make the app portable and easy to deploy, I containerized it using Docker. The `Dockerfile` starts with a Python 3.9-slim base image for efficiency. It then copies the `requirements.txt` file into the container and installs all dependencies using `pip`.

I set the working directory to `/app` and configured environment variables to ensure Flask runs on `0.0.0.0`, making it accessible outside the container. Finally, I exposed port 5000 and defined the command to start the Flask app. After building the image with `docker build`, I ran it locally using `docker run` and verified it worked by accessing `localhost:5000` in my browser.

---

#### \*\*Part 3: Pushing to AWS ECR\*\*

For cloud deployment, I needed a centralized repository to store the Docker image. Using AWS ECR (Elastic Container Registry), I created a private repository to host my container image. I automated this process with a Python script leveraging the `boto3` SDK, which programmatically created the ECR repo.

After authenticating Docker with ECR, I tagged my local image and pushed it to the repository using `docker push`. This step ensured my image was stored securely in the cloud and ready for deployment in a Kubernetes cluster.

---

#### \*\*Part 4: Kubernetes Deployment on AWS EKS\*\*

With the image in ECR, I deployed the application on Kubernetes using Amazon EKS. First, I created an EKS cluster with managed node groups, configuring networking and security settings to ensure smooth operation.

Next, I defined a Kubernetes deployment manifest, specifying the container image from ECR, the number of replicas, and resource limits. I also created a service to expose the application externally. Using `kubectl`, I applied these configurations and verified the pods were running. Finally, I used `port-forward` to test the app locally, confirming it was accessible and functioning as expected.

---

#### \*\*Part 5: Automation and DevOps Practices\*\*

To streamline the process, I wrote Python scripts to automate infrastructure tasks. Using `boto3`, I created the ECR repository programmatically. For Kubernetes management, I used the `kubernetes-client` library to define and apply deployments and services directly from Python.

This approach highlights my ability to integrate development and operations workflows, reducing manual effort and ensuring consistency. All code is available on my GitHub, including the Dockerfile, Kubernetes manifests, and automation scripts.

---

#### \*\*Conclusion\*\*

This project encapsulates my expertise in building, containerizing, and deploying cloud-native applications. By combining Python, Docker, Kubernetes, and AWS, I’ve demonstrated end-to-end proficiency in modern DevOps practices.

I’d love to discuss how these skills can benefit your team. Feel free to connect with me on LinkedIn or check out the code on GitHub. Thanks for watching, and I look forward to the opportunity to contribute to your projects!

\*\*[End screen: Contact info + GitHub/LinkedIn links]\*\*

---

### \*\*Key Improvements:\*\*

1. \*\*Logical Flow\*\*: Each paragraph transitions smoothly to the next, maintaining a clear narrative.

2. \*\*Conciseness\*\*: Technical details are explained succinctly without overwhelming the viewer.

3. \*\*Engagement\*\*: The conclusion ties the project back to real-world applications and invites discussion.

This version keeps the script professional yet approachable, perfect for a job application demo. Good luck! 🚀