Tool / Platform	Purpose & Key Features	Best For	Deployment /Integration	Resources
GitHub Actions	Automate workflows with CI/CD integrated into GitHub; supports hosted runners, matrix builds, secrets	Automating testing, building, and deploying code	GitHub platform, containers, multi-cloud	GitHub Docs Tutorial
CircleCI	Cloud-based CI/CD with Docker support, job orchestration, orbs for integrations	Continuous Integration and Delivery pipelines	Docker, AWS, GCP, Kubernetes	CircleCI Docs Tutorial
Kubeflow	Kubernetes-native ML orchestration platform; pipelines, distributed training, model serving	End-to-end ML workflows on Kubernetes	Kubernetes clusters, multi-cloud	Kubeflow Docs Tutorial
MLflow	ML lifecycle management: experiment tracking, model registry, deployment	Experiment & model management	Supports multiple ML frameworks and deployment targets	MLflow Docs Tutorial

AWS SageMaker	Fully managed ML service with training, tuning, deployment, monitoring	Managed cloud ML workflows	AWS Cloud	AWS Docs Tutorial
Evidently AI	Open-source production monitoring for ML: data/ model drift, performance	Monitoring ML models & data in production	Integrates with ML pipelines, Airflow, Kubeflow, etc.	Evidently Tutorial
Grafana	Visualization and monitoring platform for infrastructure and ML metrics	Dashboards and alerting for ML infrastructure	Supports Prometheus, Loki, many data sources	Grafana Tutorial
Apache Airflow	Workflow scheduling & orchestration platform using Python DAGs	Orchestration of complex ML/data workflows	Cloud & on-prem, integrates with many ML/data tools	Airflow
BentoML	Packaging ML models as scalable REST/gRPC APIs with Docker/ Kubernetes deployment	Model serving and scalable API deployment	Kubernetes, Docker	Tutorial
DVC	Data and model version control integrated with Git	Managing datasets, model versions, reproducibility	Git-based workflows, remote storage backends	DVC Tutorial

Docker	Containerisation tool for packaging ML apps and dependencies	Ensuring environment consistency & portability	Local, cloud, Kubernetes	

GitHub Actions CI/CD Pipelines

GitHub Actions is a powerful automation platform integrated into GitHub repositories. It facilitates Continuous Integration and Continuous Deployment (CI/CD) by allowing workflows to be triggered on any GitHub event (code push, PR, issue creation).

- Supports Linux, macOS, Windows, ARM, GPU, and containers runners.
- Matrix builds allow testing code across multiple environments simultaneously.
- Secrets management built-in for secure handling of API keys, tokens.
- Marketplace offers thousands of reusable workflows and actions, including container building, deployment steps, and integrations.
- Supports multi-container testing with docker-compose.
- Free for public repositories, with detailed logging and live real-time workflow views.
  GitHub Actions Docs
  YouTube Tutorial: **End To End NLP Project with GitHub Actions**

CircleCI

CircleCI is a cloud-based CI/CD platform that automates software builds, tests, and deployment with integration to popular VCS like GitHub and Bitbucket.

- Supports pipelines triggered by VCS, API, or UI.
- Jobs can be configured to run inside Docker containers or custom machine executors.
- Certified orbs (reusable packages of CircleCI config) enable easy integrations with AWS, GCP, Kubernetes, and many DevOps tools.
- Allows building and deploying Docker images in workflows.
- Provides centralized dashboards for monitoring, logs, and management of multiple projects and organizations.
  CircleCI Docs
  YouTube Tutorial: **Learn CircleCI in 30 mins**

Kubeflow

Kubeflow is an open-source project to enable ML workflows on Kubernetes. It focuses on making deployments portable and scalable on any cloud or on-prem.

- Provides a Kubernetes-native environment to build, deploy, and manage ML workflows.
- Key components include Kubeflow Pipelines for automating and reproducing ML workflows, hyperparameter tuning, scalable training,

and KFServing for serverless model serving.
- Integrates with Argo workflows and supports TensorFlow, PyTorch, and other ML frameworks.
- Supports distributed training, metadata tracking, notebook servers, and centralized dashboards.
  Kubeflow Docs
  YouTube Tutorial: **Kubeflow End-to-End ML Pipeline**

MLflow
MLflow is a platform for managing ML lifecycle including experiment tracking, reproducibility, model packaging, and deployment.

- Key features: Experiment Tracking to log parameters and metrics; Projects packaging runnable code; Models to manage and serve models; Model Registry for lifecycle states like staging or production.
- Integrates with multiple ML libraries and deployment platforms.
- Facilitates reproducibility and collaboration via sharing and versioning experiments and models.
  MLflow Docs
  YouTube Tutorial: **MLflow 3.0 Ultimate Guide**

Deployment Techniques in AWS, Azure, GCP, Docker, and Kubernetes
AWS

- AWS SageMaker offers a fully managed service for building, training, and deploying ML models.
- Supports notebook instances, distributed training, automated model tuning, and scalable endpoints.

Azure

- Azure Machine Learning provides a similar managed service with compute clusters, drag-and-drop pipelines, and ML ops integrations.

GCP

- Google Vertex AI offers end-to-end AI platform capabilities for data preparation, training, tuning, deployment, and monitoring.

Docker

- Containerizes applications including ML models and environments for reproducibility.

Kubernetes

- Orchestrates containerized applications for scaled, resilient deployments with rolling updates, autoscaling, and service discovery.
- Used by Kubeflow to run ML pipelines across clusters.

Evidently AI
Evidently helps monitor ML models and data quality in production by tracking data drift, concept drift, and performance degradation.

- Provides visual dashboards with real-time monitoring and alerts.
- Integrates with common ML pipelines and orchestration tools.

Evidently AI
YouTube Tutorial: Evidently AI Tutorial

Grafana (Monitoring)
Grafana is an open-source platform for monitoring and observability, widely used for visualizing metrics and logs from systems including ML deployments.

- Supports multiple data sources like Prometheus, Loki, InfluxDB.
- Enables custom dashboards and alerting for infrastructure, applications, and ML pipeline health.
  Grafana
  YouTube Tutorial: Grafana Beginner Tutorial

Apache Airflow
Airflow is a platform to programmatically author, schedule, and monitor workflows.

- Defines directed acyclic graph (DAG) pipelines in Python code.
- Integrates with many data and ML tools, cloud providers, and supports complex dependency management and retries.
  Apache Airflow

BentoML
BentoML is a framework for packaging ML models as REST/gRPC APIs ready for serving and deployment.

- Supports Docker/Kubernetes deployment.
- Simplifies building scalable, fast prediction services.
  YouTube Tutorial: BentoML Tutorial

AWS SageMaker
Fully managed ML service on AWS with training, tuning, deployment, monitoring, and governance tools.

- Supports built-in algorithms, custom containers, and integration with AWS ecosystem.
  YouTube Tutorial: AWS SageMaker End-to-End Implementation

DVC (Data Version Control)
DVC extends Git capabilities to version large data files, ML models, and pipelines, ensuring reproducibility and collaboration.

- Supports pipeline stages with caching and remote storage backends.
  DVC
  YouTube Tutorial: DVC Complete Tutorial

Dockers
Docker containers package applications and their dependencies portably.

- Essential for shipping ML models with consistent environments.
- Integrates well with CI/CD and cloud deployment strategies.

YouTube Tutorial: **Docker for Beginners 2025**

Here are the detailed notes on each of the MLOps and CI/CD tools you listed.

1. GitHub Actions CI/CD Pipelines

GitHub Actions is a **Continuous Integration (CI)** and **Continuous Delivery (CD)** platform built directly into GitHub. It allows you to automate your build, test, and deployment pipelines right from your repository.

- **Core Concepts:**
  - **Workflow:** An automated process defined in a YAML file (e.g., .github/workflows/ci.yml). A repository can have multiple workflows.
  - **Event:** A specific activity that triggers a workflow, such as a push to a branch, a pull_request, or a scheduled time (on: [push, pull_request]).
  - **Job:** A set of steps that execute on the same **runner** (virtual machine). Workflows can have one or more jobs that run sequentially or in parallel.
  - **Step:** An individual task within a job. A step can be a shell command (e.g., run: npm install) or an **Action**.
  - **Action:** A reusable, pre-built command or script. You can use actions from the GitHub Marketplace (e.g., actions/checkout@v4 to check out your code) or write your own.
- **How it Works:**
  1. You create a .yml file in the .github/workflows directory of your repository.
  2. You define the on: event that will trigger the workflow.
  3. You define one or more jobs.
  4. Within each job, you specify a runs-on: environment (like ubuntu-latest).
  5. You list the steps for the job, such as checking out code, setting up a language (e.g., actions/setup-python@v5), installing dependencies, running tests, and deploying.
- **Key Features:**
  - **Integrated:** Lives with your code in GitHub.
  - **Marketplace:** A huge library of reusable actions for AWS, Azure, GCP, Docker, etc.
  - **Matrix Builds:** Easily test your code against multiple versions of a language or on different operating systems in parallel.
  - **Secrets:** Securely store and use secrets like API keys and

passwords in your workflows.

2. CircleCI

CircleCI is a popular, third-party CI/CD platform known for its speed, performance, and rich feature set. It connects to your repository (GitHub, Bitbucket) and runs your pipelines.

- **Core Concepts:**
  - **Configuration:** All configuration lives in a single file: .circleci/config.yml.
  - **Pipeline:** The entire set of processes triggered when you push code.
  - **Workflow:** Defines the orchestration and order of your jobs. You can use workflows to run jobs in parallel, sequentially, or on a schedule.
  - **Job:** A collection of steps that run in a specified execution environment (e.g., a Docker container or a virtual machine).
  - **Step:** An individual command, similar to GitHub Actions.
  - **Orbs:** Reusable packages of CircleCI configuration. They are like GitHub Actions but for CircleCI, allowing you to import pre-built jobs and commands (e.g., an aws-s3 orb for deploying to S3).
- **Key Features:**
  - **Performance:** Known for very fast execution, with powerful caching and parallelism.
  - **Execution Environments:** Strong support for Docker, allowing you to specify the exact container image for your job, ensuring consistency.
  - **Contexts:** A secure way to manage and share environment variables (secrets) across different projects.
  - **Local CLI:** Allows you to run and debug jobs locally on your machine before pushing code.

3. Kubeflow

Kubeflow is an **open-source MLOps platform for Kubernetes**. Its goal is to make deploying, scaling, and managing machine learning workflows on Kubernetes simple, portable, and scalable. Think of it as "Kubernetes for Data Scientists."

- **Core Components:**
  - **Kubeflow Pipelines:** The most popular component. It allows you to build and run **reproducible ML workflows (pipelines)** where each step runs as a container in Kubernetes. This is excellent for complex training, validation, and deployment sequences.

    - **Jupyter Notebooks:** Provides a simple way to spin up and manage hosted Jupyter notebooks directly within the Kubernetes cluster, with easy access to cluster resources.
    - **Model Serving:** Integrates tools like **KServe** (formerly KFServing) and **Seldon Core** to deploy your trained models as scalable, production-ready inference services.
    - **Experiment Tracking:** Integrates with tools (or has its own) to track hyperparameters, metrics, and artifacts from your training runs.
- **Why Use It?**
    - **Scalability:** Leverages Kubernetes to scale your ML tasks (like distributed training or hyperparameter tuning) up and down.
    - **Portability:** Since it runs on Kubernetes, your entire ML workflow can run on any cloud (AWS, GCP, Azure) or on-premise without changes.
    - **End-to-End:** Aims to provide a single platform for the entire ML lifecycle, from experimentation (notebooks) to training (pipelines) to production (serving).

4. MLflow

MLflow is an **open-source platform to manage the end-to-end machine learning lifecycle**. It is framework-agnostic (works with TensorFlow, PyTorch, scikit-learn, etc.) and focuses on solving key MLOps challenges.
It is composed of four main components:

- **1. MLflow Tracking:**
    - **What it is:** An API and UI for logging and querying experiments.
    - **Use Case:** You add mlflow.log_param(), mlflow.log_metric(), and mlflow.log_artifact() to your training code.
    - **Result:** A central server (the **Tracking Server**) captures all your hyperparameters, performance metrics (like accuracy or RMSE), and output files (like model weights or plots). This allows you to easily compare runs and see what worked.
- **2. MLflow Projects:**
    - **What it is:** A standard format for packaging your ML code so it can be reproduced by others.
    - **How it works:** You define a MLproject file that specifies the code's dependencies (e.g., a conda.yaml file) and its entry points (the commands to run, like python train.py).
    - **Result:** Anyone can run your project with a single command (mlflow run .), and MLflow automatically sets up the correct environment.
- **3. MLflow Models:**
    - **What it is:** A standard format for packaging trained models.

    - **How it works:** When you save a model (e.g., mlflow.sklearn.log_model()), MLflow saves it in a "flavor" format.
    - **Result:** This single model format can be easily deployed to many different platforms (e.g., as a local REST API, on AWS SageMaker, or on Azure ML) with simple commands.
- **4. MLflow Model Registry:**
    - **What it is:** A centralized hub for managing the lifecycle of your production models.
    - **Use Case:** After training and logging a model, you "register" it. From there, you can manage its **versions** (e.g., Model "Fraud-Detector" Version 1, Version 2) and its **stages** (e.g., Staging, Production, Archived).
    - **Result:** Your applications can automatically pull the "Production" version of a model, and you have a clear, auditable process for promoting new models.

5. Deployment Techniques (AWS, Azure, GCP, Docker, Kubernetes)

This is a broad topic about *how* and *where* you deploy applications (including ML models).

- **Docker:**
    - **Concept:** The foundational technology. Docker lets you package your application, its dependencies (libraries, code, etc.), and its configuration into a single, isolated unit called a **container**.
    - **Why:** A container runs the **exact same way** on a developer's laptop, a test server, and a production machine. This solves the "it works on my machine" problem.
- **Kubernetes (K8s):**
    - **Concept:** The **container orchestrator**. While Docker lets you create and run *one* container, Kubernetes lets you manage *thousands* of containers across a *cluster* of machines.
    - **What it does:**
        - **Scalability:** Automatically scales your application up (adds more containers) or down based on traffic.
        - **Self-healing:** If a container crashes, K8s automatically restarts it. If a whole machine (node) dies, K8s moves its containers to a healthy node.
        - **Service Discovery & Load Balancing:** Gives your containers a single, stable network address (a "Service") and distributes traffic between them.
        - **Rollouts & Rollbacks:** Lets you perform rolling updates to your application with zero downtime. If the new version is bad, you can instantly roll back.

- **Cloud Platforms (AWS, Azure, GCP):** These are the providers that give you the infrastructure to run everything. For deployment, they offer two main paths:
    - **1. Infrastructure as a Service (IaaS):** You rent virtual machines (e.g., **AWS EC2**, **Azure VM**, **GCP Compute Engine**) and you install/manage everything (Docker, K8s) yourself. This is flexible but complex.
    - **2. Platform as a Service (PaaS) / Managed Services:** The cloud provider manages the underlying infrastructure for you. This is the modern, preferred approach.
        - **Managed Kubernetes:**
            - **AWS EKS** (Elastic Kubernetes Service)
            - **Azure AKS** (Azure Kubernetes Service)
            - **GCP GKE** (Google Kubernetes Engine)
        - You get a fully managed K8s control plane, and you just add your worker machines. This is the most common way to run K8s.

        - **Managed Container Services (Serverless):**
            - **AWS Fargate** (run containers without managing servers)
            - **Azure Container Apps**
            - **GCP Cloud Run**
        - You give them a Docker container, and they run and scale it for you. You don't even see the underlying machines. This is often the simplest path.

        - **Managed ML Platforms:**
            - **AWS SageMaker**, **Azure ML**, **GCP Vertex AI** (more on SageMaker below).

6. Evidently AI

Evidently AI is an **open-source Python library for monitoring and evaluating ML models**. It helps you answer the question: "Is my model still working well in production?" It specializes in detecting **data drift** and **model drift**.

- **Core Concepts:**
    - **Drift:** The phenomenon where the production data your model sees starts to look different from the training data, causing model performance to degrade.
        - **Data Drift:** The statistical properties of the *input features* change (e.g., average purchase price suddenly increases).
        - **Concept Drift:** The *relationship* between features and the target changes (e.g., a feature that used to predict "buy" now predicts "don't buy").

    - **Reference Dataset:** A "golden" dataset that represents what your model was trained on (e.g., your test or validation set).
    - **Current Dataset:** The new data coming from production (e.g., the last 24 hours of predictions).
- **Key Features:**
    - **Reports:** Generates rich, interactive HTML reports that compare your current dataset to your reference dataset. It includes dozens of statistical tests and visualizations for drift, missing values, and performance metrics (if you have ground truth labels).
    - **Test Suites:** Allows you to define pass/fail checks (e.g., "Alert me if the mean of feature_X drifts by more than 10%"). This is perfect for running in an automated CI/CD pipeline or a scheduled job.
    - **Monitoring Dashboard:** A visual dashboard (built on Grafana) that continuously tracks these metrics over time, so you can see trends.

7. Grafana (Monitoring)

Grafana is an **open-source platform for data visualization, monitoring, and alerting**. It is the *de facto* industry standard for creating monitoring dashboards.

- **How it Works:**
    - Grafana **does not store any data itself**.
    - It is a visualization layer that **connects to data sources** where your metrics, logs, and traces are already stored.
    - **Common Data Sources:**
        - **Prometheus:** (The most common) A time-series database for metrics (e.g., CPU usage, memory, request counts).
        - **Loki:** (Made by Grafana) A database for logs (e.g., application logs).
        - **Tempo:** (Made by Grafana) A database for traces (for tracking a request as it moves through different microservices).
        - Others: SQL databases, Elasticsearch, AWS CloudWatch, etc.
- **Key Features:**
    - **Dashboards:** Lets you build powerful, beautiful, and highly customizable dashboards by querying your data sources.
    - **Visualization:** Offers a huge variety of panels (graphs, tables, heatmaps, gauges, etc.).
    - **Alerting:** You can define alert rules based on your data (e.g., "Alert me if error rate is > 5% for 10 minutes") and send notifications to Slack, PagerDuty, etc.
    - **Explore:** An interface for interactively querying and "digging" through your logs and metrics to debug issues.

8. Apache Airflow

Airflow is an **open-source platform to programmatically author, schedule, and monitor workflows**. It is primarily used for **Data Engineering** and **ETL (Extract, Transform, Load)** pipelines.

- **Core Concepts:**
  - **Workflows as Code:** You define your workflows (pipelines) in **Python**.
  - **DAG (Directed Acyclic Graph):** A DAG is a Python file that defines a collection of tasks and their dependencies. It describes *what* to run and in *what order*.
  - **Task:** A single unit of work in a DAG (e.g., run a SQL query, call an API, run a Python script).
  - **Operators:** Pre-built templates for tasks. Airflow has operators for almost everything (e.g., BashOperator, PythonOperator, PostgresOperator, SlackOperator).
- **How it Differs from a CI/CD Tool (like GitHub Actions):**
  - **CI/CD Tools (GitHub Actions):** Are **event-driven**. They react to code changes (like a push) to *build and deploy* software.
  - **Airflow:** Is **data-driven and schedule-driven**. It runs complex, multi-step tasks on a schedule (e.g., "Every night at 1 AM, pull data from Salesforce, transform it in Spark, and load it into our data warehouse"). It's built for data orchestration, not software deployment.
- **Key Features:**
  - **Rich UI:** A web interface to visualize your pipelines (past and present), monitor progress, and debug failures.
  - **Scalability:** Can run thousands of tasks per day across a cluster of workers.
  - **Extensibility:** You can easily write your own custom operators and hooks in Python.

9. BentoML

BentoML is an **open-source framework for building, shipping, and running high-performance ML model inference services**. It bridges the gap between a trained model (e.g., a .pkl file) and a production-ready API.

- **The Problem it Solves:** A trained model isn't an application. You still need to write an API server (like Flask/FastAPI), handle data validation, manage dependencies, and package it all in a Docker container. BentoML automates this.
- **How it Works (The Workflow):**
  1. **Save Model to Local Bento Store:** After training, you save your

model to the BentoML store (e.g., bentoml.sklearn.save_model("my_classifier", model)).
2. **Define a Service:** You create a service.py file. In it, you import your model and define an API endpoint. You use Python type hints to define the input/output schema (e.g., input=NumpyNdarray(), output=JSON()).
3. **Build a "Bento":** You run bentoml build. This command packages your service.py, your saved model, and all necessary dependencies into a standardized, versioned folder called a **Bento**.
4. **Deploy the Bento:**
    - bentoml serve: Instantly runs a high-performance API server locally for testing.
    - bentoml containerize: Automatically builds an optimized Docker image for your Bento.
    - You can then deploy this Docker image anywhere (Kubernetes, AWS Fargate, GCP Cloud Run, etc.).

- **Key Features:**
    - **Automatic API Generation:** Creates optimized REST APIs with OpenAPI/Swagger docs out of the box.
    - **High Performance:** Built on top of FastAPI and other high-performance tools.
    - **Adaptive Batching:** A powerful feature that automatically batches incoming requests (e.g., from many users) to be processed by your model (especially on a GPU) at the same time, dramatically increasing throughput.

10. AWS SageMaker

Amazon SageMaker is a **fully managed, end-to-end platform for Machine Learning on AWS**. It is a massive, comprehensive service that aims to handle every part of the ML lifecycle, from data labeling to production monitoring.

- **Key Components:**
    - **SageMaker Studio:** The central IDE (like RStudio or JupyterLab) that brings all the other components together.
    - **SageMaker Data Wrangler:** A visual tool for cleaning, transforming, and preparing data for training.
    - **SageMaker Feature Store:** A central repository to store, share, and manage curated features for model training and inference.
    - **SageMaker Training:** A service to run managed, distributed training jobs. You provide your training script (in a Docker container), and SageMaker spins up the required instances (e.g., powerful GPUs), runs the job, and then spins them down, so you only pay for what you use.
    - **SageMaker Autopilot:** An "AutoML" feature that automatically

trains and tunes many models on your data and gives you the best one.
    - **SageMaker Endpoints (Deployment):** The service for deploying your trained model. It creates a fully managed, auto-scaling HTTPS endpoint. You send it data, and it returns predictions.
    - **SageMaker Model Monitor:** Automatically monitors your production endpoints for data drift and model quality drift (similar to Evidently AI, but integrated into AWS).
- **Pros:** Fully managed, deeply integrated with the AWS ecosystem (S3, IAM, etc.), and extremely scalable.
- **Cons:** Can be complex, expensive, and locks you into the AWS ecosystem.

11. DVC (Data Version Control)

DVC is an **open-source tool for versioning data and ML models**. It is **built to work *with* Git**.

- **The Problem it Solves:**
    - Git is great for versioning **code**, which is small text files.
    - Git is **terrible** for versioning **data** and **models**, which are large binary files (GBs or TBs). Pushing a 10 GB dataset to GitHub will fail.
- **How it Works (The "Pointer" System):**
    1. You run dvc add my_data/data.csv.
    2. DVC moves data.csv to a special cache (e.g., .dvc/cache) and *out* of Git's tracking.
    3. DVC creates a tiny text file called my_data/data.csv.dvc. This file is just a **pointer** (containing a hash of the data) that tells DVC where to find the real file.
    4. You **commit this small .dvc file to Git**.
    5. You then run dvc push. This command looks at the pointer file and uploads the *actual* large data file from your cache to a remote storage (like **S3**, **GCP Storage**, **Azure Blob**, or an SSH server).
- **The Result:**
    - Your Git repository stays small and fast (it only contains code and small .dvc pointers).
    - Your large data/model files are versioned in cheap, remote blob storage.
    - When a teammate runs git pull (they get the new code and pointers) and then dvc pull (DVC downloads the correct data files), they have a fully reproducible environment.
- **DVC Pipelines:** DVC also has a simple pipelining feature (defined in dvc.yaml) to codify the steps of your ML process (e.g., preprocess ->

train -> evaluate).

12. Docker

(This was covered in section 5, but here it is as a standalone topic.)
Docker is a **platform that enables you to build, ship, and run applications in isolated environments called containers**.

- **Container vs. Virtual Machine (VM):**
  - **VM:** A VM (like VirtualBox or VMware) **virtualizes the hardware**. Each VM includes a *full copy* of an operating system (e.g., its own Windows or Linux kernel). This is heavy, slow to boot, and resource-intensive.
  - **Container:** A container **virtualizes the operating system**. All containers on a machine *share* the same host OS kernel. They only package their own app, libraries, and binaries. This makes them:
    - **Lightweight:** Megabytes in size, not gigabytes.
    - **Fast:** Start in milliseconds, not minutes.
    - **Efficient:** You can run many more containers on a single machine than VMs.
- **Core Docker Objects:**
  - **Dockerfile:** A simple text file that contains the **instructions** for building a Docker image (e.g., FROM python:3.10, COPY . ., RUN pip install -r requirements.txt, CMD ["python", "app.py"]).
  - **Image:** The **read-only blueprint** created from a Dockerfile. It's a "snapshot" of your application and its dependencies. You push and pull images from a registry.
  - **Container:** A **runnable instance** of an image. It's the "running" version of the blueprint. You can start, stop, and delete containers.
  - **Registry:** A place to store and share your images. **Docker Hub** is the main public registry. AWS (ECR), Azure (ACR), and GCP (GCR) all have their own private registries.