Deloitte.

Tech Treks

Building a DevOps Pipeline

Welcome!

- ROI leads the industry in designing and delivering customized technology and management training solutions
- Meet your instructor
  - Name
  - Background
  - Contact info
- Let's get started!

ROITRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Course Objectives

In this course, you will:

- Create a complete DevOps pipeline
- Manage application code and versions using Git and GitHub
- Execute CI/CD pipelines using GitHub Actions
- Package application code and dependencies using Docker Images
- Deploy containers using Serverless cloud environments

Agenda

This course is taught over three 2-hour sessions.

Session 1:

Managing Software Development with Git

Session 2:

Packaging Code with Docker

Session 3:

Automating Deployment to Serverless Compute

Agenda

This is session two.

Session 1:

Managing Software Development with Git

Session 2:

Packaging Code with Docker

Session 3:

Automating Deployment to Serverless Compute

ROITraining
Maximize Your Training Investment

Session 2: Packaging Code with Docker

Session Objectives

In this session, you will:

- Create and run Docker images
- Push Docker images to Container Registries
- Automate using Docker with GitHub Actions

Session Concepts

Docker

GitHub Actions and Docker

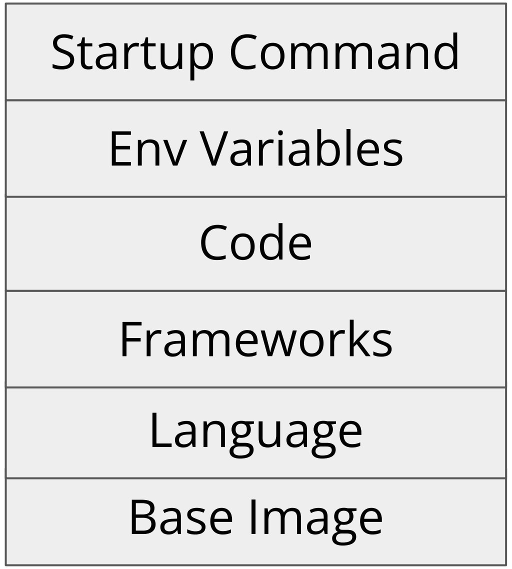
Lab

Docker

- Allows applications or microservices to be deployed to containers
    - Multiple containers can run on a single virtual machine
- Docker images are very lightweight, pre-configured virtual environments
    - Include the required software to run an application
    - Applications are inside the Docker image
- Docker images will run on any platform that has Docker installed
- Docker images allow applications to be easily moved
    - From developer to test to production environments
    - Between local and cloud-based data centers
    - Between different cloud providers

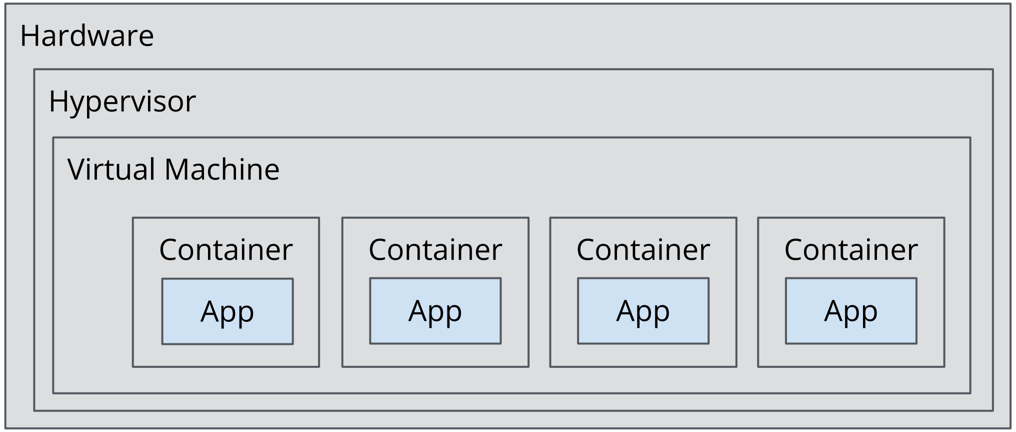
Images

- Images are deployment packages that are used to build containers
  - Containers are running instances of images
- Images are built in layers
  - Start with a base image
  - Add languages and frameworks used by your app
  - Copy in your code
  - Create environment variables
  - Specify how your application starts



Containers

- Containers are running instances of images
- Containers do not include the operating system
  - The OS requires the container software be installed (Docker)



Advantages of Containers

- Start very fast (milliseconds)
- Small, so easy to move around
- Isolates one program from another
- Run on any machine

Some Basic Docker Commands

Command	Description
docker build [OPTIONS] PATH | URL | - **Example:** docker build -t drehnstrom/converter-dar:latest .	Build a custom Docker container based on a `Dockerfile`. Run the command from the same folder as the `Dockerfile`
docker run [OPTIONS] IMAGE [COMMAND] [ARG...] **Example:** docker run -d -p 8080:8080 drehnstrom/converter-dar	Run a Docker image.
docker ps [OPTIONS	List running docker images. Displays containers and their IDs.
docker stop [OPTIONS] CONTAINER [CONTAINER...] **Example:** docker stop <container-id-here>	Stop a running image.
docker login [OPTIONS] [SERVER]	Login to Docker Hub.
docker push [OPTIONS] NAME[:TAG] **Example:** docker push drehnstrom/converter-dar	Push a container to Docker Hub.
docker pull [OPTIONS] NAME[:TAG] **Example:** docker pull drehnstrom/converter-dar	Get a container from Docker Hub.

Creating Custom Docker Images

- To build a custom image, create a file call `Dockerfile`
- Steps
  1. Start with a base image from Docker Hub or another registry
  2. Identify yourself (*so you can upload your custom image later*)
  3. Install prerequisite software onto the base image
  4. Copy your application onto the image
  5. Configure your application
  6. Specify how to start your application
- Use `docker build` command to create the container
- Once the container is created use `docker run` command to start it

Example `Dockerfile` for Python App

```
FROM python:3.11
WORKDIR /app
COPY . .
RUN pip install gunicorn
RUN pip install -r requirements.txt
ENV PORT=8080
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 main:app
```

Building Docker Images

- Use the Docker build command to create the image
  - `-t` parameter tags (*names*) the image (can include a version number)
  - Specify the path to the `Dockerfile`
- Tag is used later to specify which image you want to run
- Syntax:

  -
    ```
    docker build -t your-docker-id/your-image:v0.1 .
    ```

Example Build Command Output

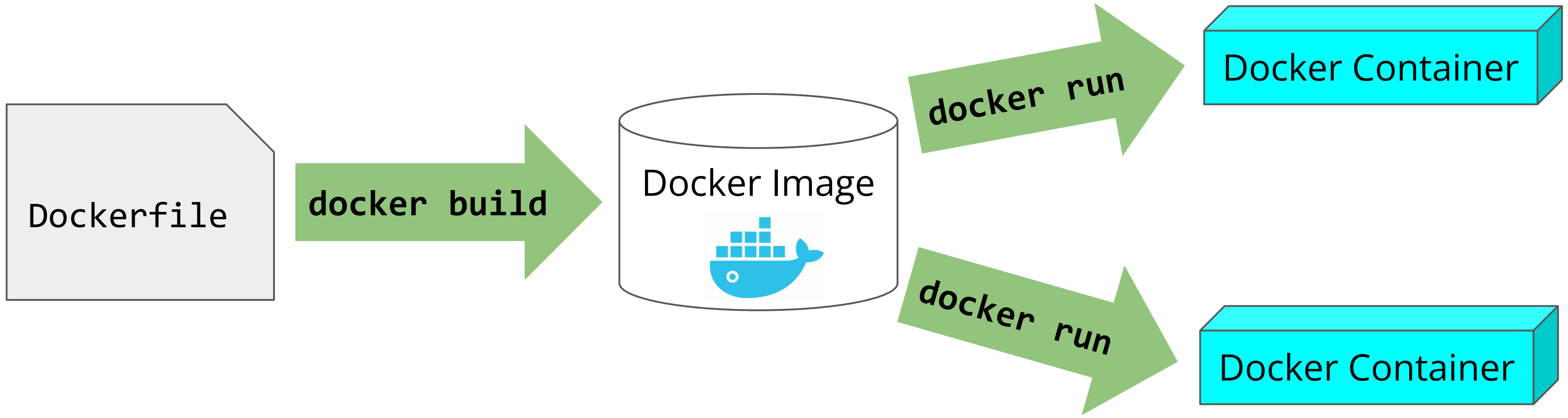
```
$ docker build -t drehnstrom/devops-demo:v0.1 .
Sending build context to Docker daemon  2.828MB
Step 1/7 : FROM python:3.11
 ---> 34a518642c76
Step 2/7 : WORKDIR /app
<< CODE OMITTED>>
Step 6/7 : ENV PORT=8080
 ---> Using cache
---> 7045daaafd44Step 7/7 : CMD exec gunicorn --bind :$PORT --workers 1 --threads 8
main:ap ---> Using cache
---> 7c32a538632e
Successfully built 7c32a538632e
Successfully tagged drehnstrom/devops-demo:v0.1
```

Starting Containers

- Use the Docker run command to start a container based on an image
  - -p parameter specifies the port to listen on and the port to forward to
- Example:

```
$ docker run -p 8080:8080 drehnstrom/devops-demo:v0.1
[2019-07-02 12:07:13 +0000] [1] [INFO] Starting gunicorn 19.9.0[2019-07-02 12:07:13
+0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)[2019-07-02 12:07:13 +0000]
[1] [INFO] Using worker: threads[2019-07-02 12:07:13 +0000] [8] [INFO] Booting
worker with pid: 8
```

Docker Images and Containers



Listing Containers and Images

- To see your containers, use the Docker ps command
  - -a parameter shows all containers, not just those that are running

```
$ docker ps -a
CONTAINER ID   IMAGE                           COMMAND
7db7aed583f8   drehnstrom/devops-demo:v0.1     "/bin/sh -c 'exec gu…"
```

- To see your images, use the Docker images command

```
$ docker images
REPOSITORY               TAG    IMAGE ID       CREATED          SIZE
drehnstrom/devops-demo   v0.1   7c32a538632e   23 minutes ago   946MB
python                   3.7    34a518642c76   3 weeks ago      929MB
```

Deleting Containers and Images

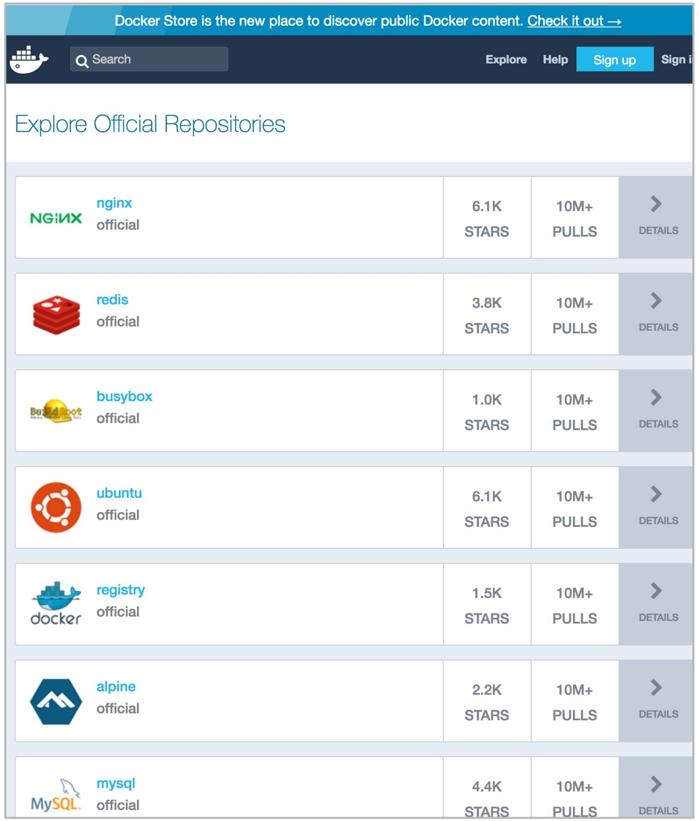
- Use Docker `rm` command to remove containers
  - `docker rm <CONTAINER ID>`
- To stop all running containers:
  - `docker stop $(docker ps -a -q)`
- To remove all containers:
  - `docker rm $(docker ps -a -q)`
- Use Docker `rmi` command to remove images
  - `docker rmi <IMAGE ID>`

Docker Registries

- Registries are centralized locations where Docker images can be stored
- Public registries are available to everyone
  - Base images for different environments are often stored publicly
  - Open-source applications might be stored in public registries
- Private registries are secured and managed by some organization
  - Control access to your proprietary software
- Registries are easy to create
- Access registries over the internet or your private network

Docker Hub

- Official registry of Docker images
  - Can create both public and private Docker repositories
- Images for many operating systems and languages
  - Starting points for building your images
- Can upload custom images
  - When deployed onto systems, your custom images are downloaded from Docker Hub


Docker Store is the new place to discover public Docker content. Check it out →

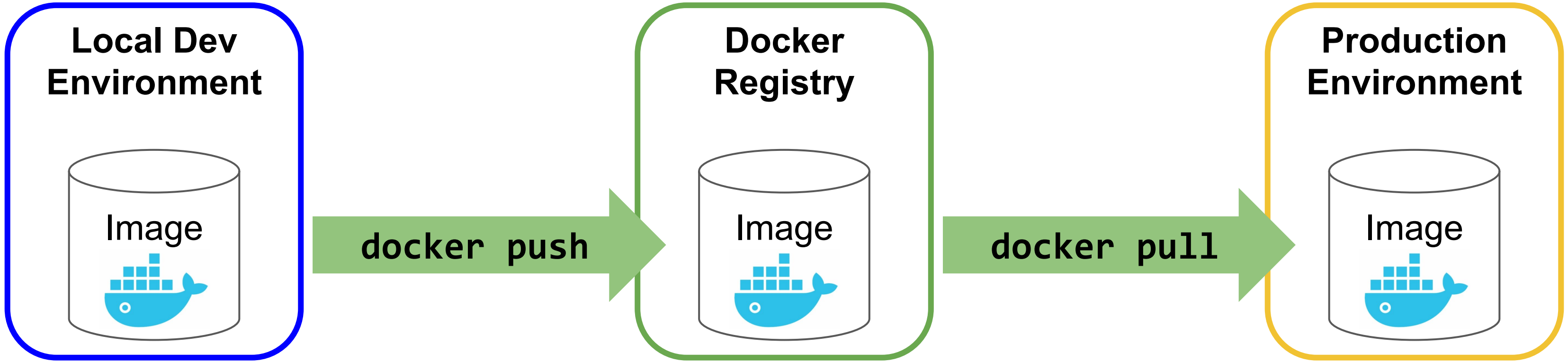
Explore Official Repositories

nginx	official	6.1K STARS	10M+ PULLS	DETAILS
redis	official	3.8K STARS	10M+ PULLS	DETAILS
busybox	official	1.0K STARS	10M+ PULLS	DETAILS
ubuntu	official	6.1K STARS	10M+ PULLS	DETAILS
registry	official	1.5K STARS	10M+ PULLS	DETAILS
alpine	official	2.2K STARS	10M+ PULLS	DETAILS
mysql	official	4.4K STARS	10M+ PULLS	DETAILS

Push and Pull to Docker Hub

- Use the Docker push command to save an image to a repository
  - To save a container to Docker Hub:

    `docker push` ***`your-docker-id`***`/devops-demo:v0.1`
- Use the pull command to get an image from a repository
  - `docker pull` ***`your-docker-id`***`/devops-demo:v0.1`



Session Concepts

Docker

GitHub Actions and Docker

Lab

GitHub Workflows

- **Workflows** consist of one or more Jobs programmed in YAML
- **Jobs** have one or more steps which represent individual tasks
- **Steps** can be automated with GitHub Actions or can be Shell commands
- **Triggers** determine when a workflow runs
- **Runners** are servers hosted in GitHub or in your environment that execute the workflow

Example Workflow

```
name: Run Tests
on:
  push:
    branches:
      - session-1
  pull_request:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
```

Triggers

Job

Runner

<<Continued on Next Slide>>

Example Workflow (continued)

<<Continued from Previous Slide>>

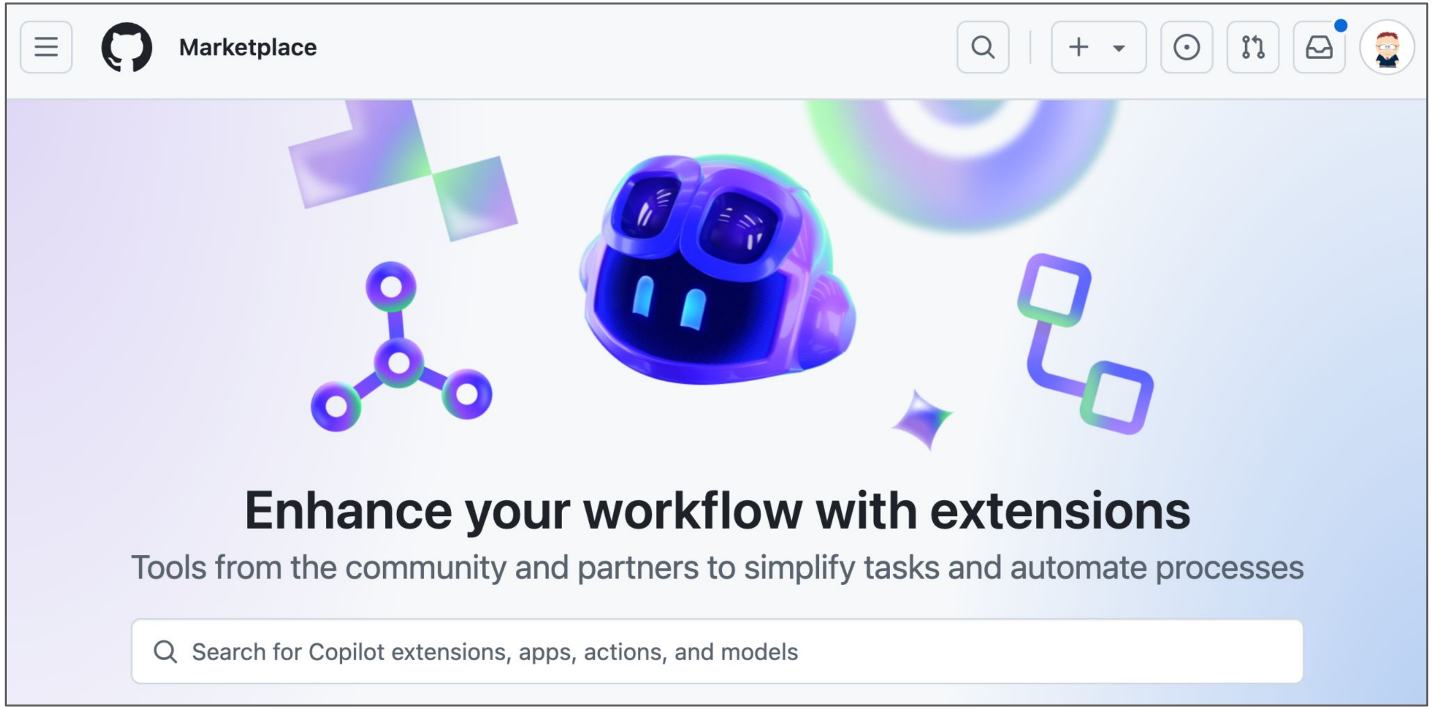
```
steps:
  - name: Checkout code
    uses: actions/checkout@v3

  - name: Set up Python
    uses: actions/setup-python@v5
    with:
      python-version: '3.11'

  - name: Install dependencies
    run: |
      python -m pip install --upgrade pip
      pip install -r requirements.txt

  - name: Run tests
    run: |
      pytest --maxfail=1 --disable-warnings
```

steps

Actions

Shell commands

GitHub Marketplace has 1000s of Actions



Enhance your workflow with extensions

Tools from the community and partners to simplify tasks and automate processes

Search for Copilot extensions, apps, actions, and models

- https://github.com/marketplace

Example Job Using Docker Actions

```
jobs:
  build_and_push:
    runs-on: ubuntu-latest
    needs: test
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}
```

Run after test job

Action to log into Docker Hub

Docker Hub credentials

<<Continued on Next Slide>>

Example Job using Docker Actions

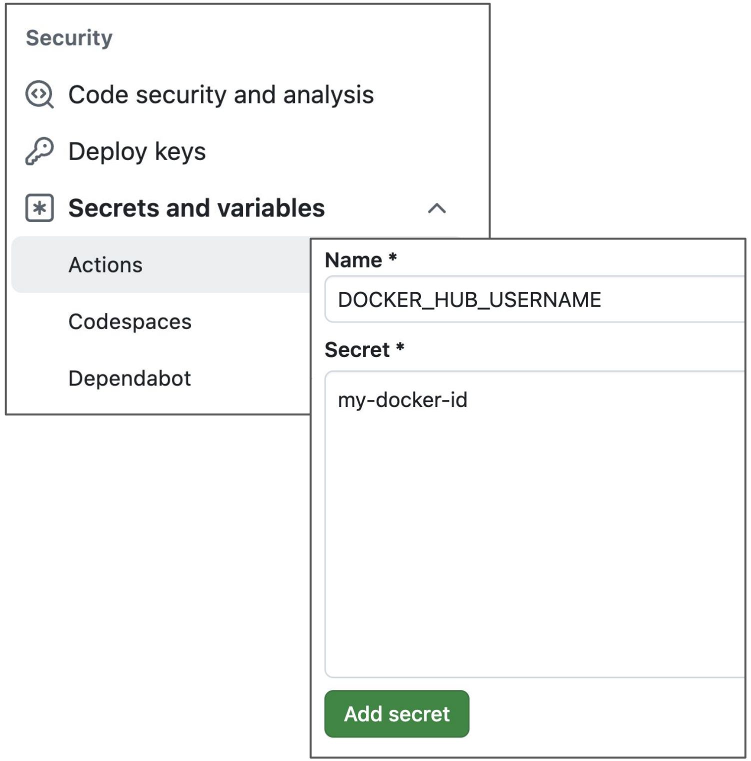
```
<<Continued from Previous Slide>>

- name: Build and push Docker image
  uses: docker/build-push-action@v5
  with:
    context: .
    push: true
    tags: ${{ secrets.DOCKER_HUB_USERNAME }}/tech-trek:${{ github.sha }}
```

Docker action to build the image and upload it to Docker Hub

Variables in ${{ }}

GitHub Secrets and Variables

- Created in repository settings
- Secrets are encrypted and you cannot see the value after it is created
- Use variables are for values that are not sensitive



Security

Code security and analysis

Deploy keys

Secrets and variables

Actions

Codespaces

Dependabot

Name *

DOCKER_HUB_USERNAME

Secret *

my-docker-id

Add secret

Session Concepts

Docker

GitHub Actions and Docker

Lab

Hands-On Exercise

- Do the following exercise:
  - Session 2 Lab: Building Docker Images with GitHub Actions

Session Summary

In this session, you have:

- Created and ran Docker images
- Pushed Docker images to Container Registries
- Automated using Docker with GitHub Actions

Discussion: Recap