Docker Compose

Reminders

1. HW1 due tonight, make sure you can push to your repository!
2. HW2 will be released later today
3. GitHub classroom problems

Agenda

1. Docker Compose Introduction
2. YAML Overview
3. Docker Compose Specification
4. Lab

Recap

- So far Jarvis is a mono-repo where each service is a separate `uv` project
  - This setup is common for large companies where each service is owned by a team
- We created configurations for our services using environment variables or a `.env` file
  - This paradigm of loading a file then environment variables is called dotenv
- We created Dockerfiles for each service to containerize them
- Goal for today: finish creating a robust local development environment that mirrors the components we need in production

What could be better about the local setup?

Recap

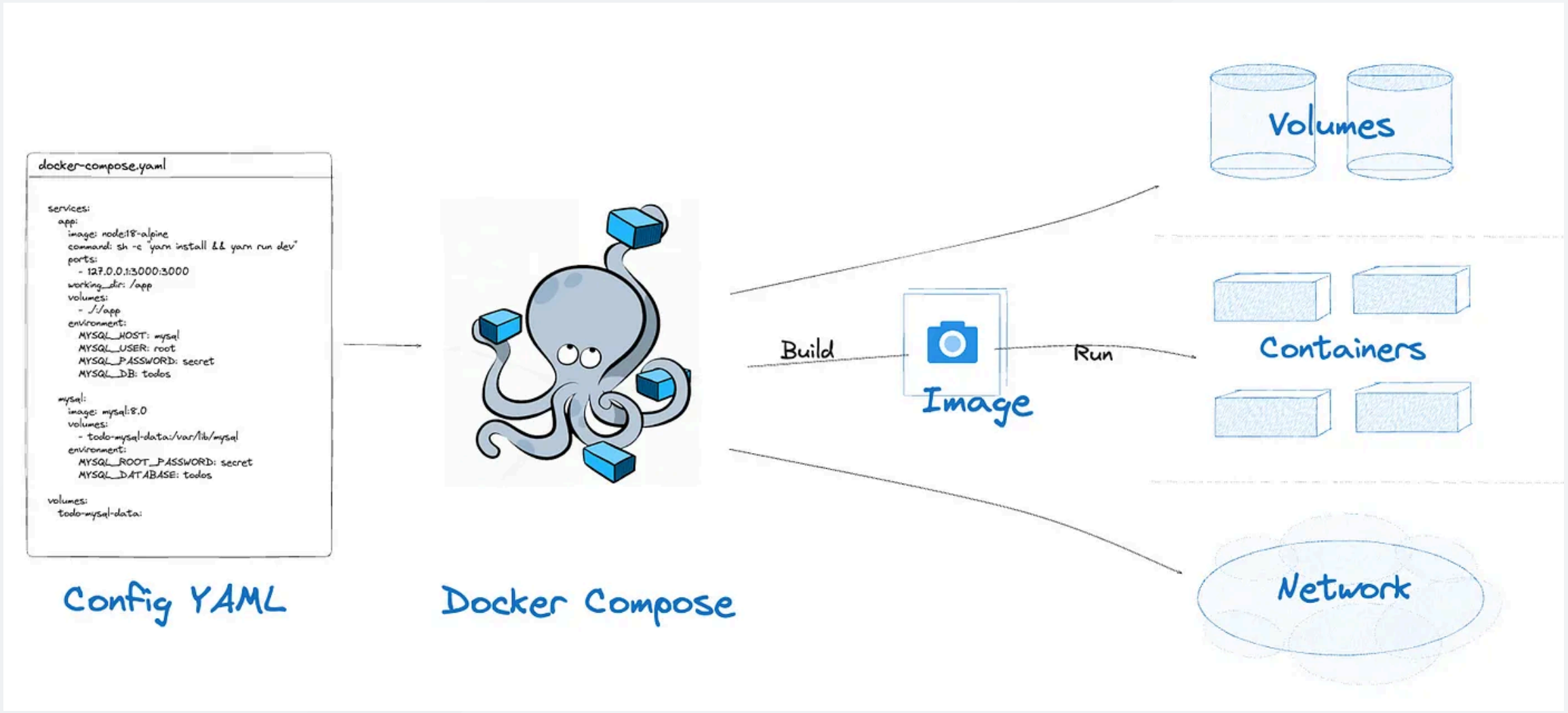
We have a problem, running Jarvis locally is a pain!

- We need to open a terminal for each service and run its container
  - If we add a service (chat), this is only going to get worse
- We need to make sure we use the right commands for running each service, i.e. picking the right port
- If we're actively developing on a service, we need to make code changes, build the Docker image, then run it again

Enter: Docker Compose

Docker Compose is a tool that allows us to define and run *multi-container applications*.

- An application is defined as a collection of containers using a YAML file `compose.yml`
- Run all containers using a single command `docker-compose up`
- Provides configurations for almost anything you need within a container
- Mainly used locally, typically not recommended for production



https://builder.aws.com/content/2qi9qQstGnCWguDzgLg1NgP8IBF/file-structure-of-docker-composeyml-file

YAML: Overview

YAML stands for "YAML Ain't Markup Language"

- Meant to be a human readable serialization format
- Used very commonly for various DevOps tools
  - Docker, Kubernetes, Terraform, CI/CD, etc
- File extension is either `.yaml` or `.yml`

```yaml
# Key-value pairs (scalars)
name: "John Doe"
age: 30
is_active: true
salary: 75000.50
department: null

# Strings (quotes optional for simple strings)
simple_string: Hello World
quoted_string: "Special chars: @#$%"
multiline_string: |
  This is a multiline string
  that preserves line breaks
  and formatting.
folded_string: >
  This is a folded string that
  will be converted to a single
  line with spaces replacing newlines.
```

```yaml
# NOTE: indentation defines scope in YAML
# Lists/Arrays (two styles)
fruits:
  - apple
  - banana
  - orange
colors: [red, green, blue]
random_things: [123, "123", true]

# Nested objects/maps
person:
  name: Alice Smith
  contact:
    email: alice@example.com
    phone: "555-1234"
  skills:
    - Python
    - JavaScript
    - YAML
```

```yaml
# Lists can contain objects
employees:
  - name: Bob Johnson
    position: Developer
    years_experience: 5
  - name: Carol Wilson
    position: Designer
    years_experience: 3
```

Docker Compose Specification

- Docker Compose introduces a few new concepts: **services**, **networks**, and **volumes**
- Each of these are defined as top level objects in the `compose.yaml` file
- Services are effectively a container running a program, e.g. a FastAPI application, Next.js frontend, Postgres database, etc
- Networks allow services to communicate with each other
- Volumes store data outside containers so it survives container restarts and updates

Services: Defining Your Containers

Services are the **containers** that make up your application. Each service runs one part of your system.

```yaml
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"

  database:
    image: postgres:15
    environment:
      POSTGRES_DB: myapp
```

- One service = one container
- Define image, ports, environment, and more

More Service Syntax

```yaml
services:
  api:
    build: ./backend           # Build from a Dockerfile
    ports:
      - "3000:3000"
    env_file:                  # Inject env variables from local file
      - ./api/.env
    depends_on:                # Start only after database service is ready
      - database
    restart: unless-stopped    # Restart policy

  database:
    image: postgres:15
    volumes:
      - db_data:/var/lib/postgresql/data    # Use volume for persistence
```

Networks: Connecting Services

- **Default network:** Unless specified, all services are included in the `default` network. This allows them to connect to each other using service names as hostnames, e.g. `http://api` connects to the `api` service.
- If you have a need to split network connections from each service, you can explicitly add services to networks
  - This can be useful from a security context

Networks Syntax

```yaml
services:
  api:
    ...
    networks:
      - web-tier      # include api service in web-tier and db-tier networks
      - db-tier

networks:
  web-tier:
    driver: bridge     # The network driver to use, typically bridge
  db-tier:
    driver: bridge
    internal: true     # Prevent external access, false by default
```

Volumes: Persisting Data

- We can create volumes to store persistent data
  - Think of these as folders
- We *mount* these volumes in our containers to use them
- We can also mount *local* files to our Docker containers
  - Useful for local development, e.g. mount the `app/` directory so that FastAPI can restart automatically within the container when it sees files have changed!

Volumes Syntax

```yaml
services:
  database:
    image: postgres:15
    volumes:
      # Mount db_data volume at /var/lib/postgresql/data
      - db_data:/var/lib/postgresql/data

  api:
    volumes:
      # Mount local folder ./api/app as /app on container
      - ./api/app:/app

volumes:
  db_data:          # Managed by Docker
  static_files:     # Shared between services
```

Docker Compose CLI

Common commands:

- `docker compose up` - start services defined in `compose.yaml`
  - adding the `-d` flag runs in detached mode, i.e. the background
- `docker compose down` - stop and remove services
- `docker compose ps` - get status of containers
- `docker compose logs` - get logs of all containers
  - Only really needed if you started in detached mode

Docker Compose Specification

There's a lot more you can do with Docker Compose, we've only just gone over the basics we need. If you're curious, check out the specification on the docker documentation.

https://docs.docker.com/reference/compose-file/

Lab: Docker Compose