

# DOCKER COMPOSE

FUNDAMENTALS



# About Me

My name is Kirk Patrick, and I have 17+ years of experience in the technology industry, including:

- 10 years as a Software Engineer
- 4 years as a DevOps Engineer
- 3 years as a Solutions Architect

## Academic Background

- Technical Degree in Accounting — Rui Barbosa School
- Bachelor's Degree in Systems Analysis and Development — FIAP
- Professional Baccalaureate in Data Science — Datatech Florida
- Postgraduate Degree in Machine Learning — FIAP
- Currently pursuing a Postgraduate Degree in Astronomy with an emphasis in Planetary Geology - FGE São Paulo / Space Today

## Additional Information

- Speaker at Campus Party Brazil (Goiânia Edition), my hometown.
- Guest Mentor for NASA events in Brazil (NASA Space Apps Challenge)
- Certified Professional in AWS, Azure, and GCP (Google Cloud Platform).

## Current Position

Currently working at AtoBe, part of the Brisa Group, where I am responsible for restructuring the entire back-office system and leading a company-wide Artificial Intelligence initiative.

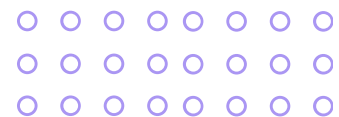
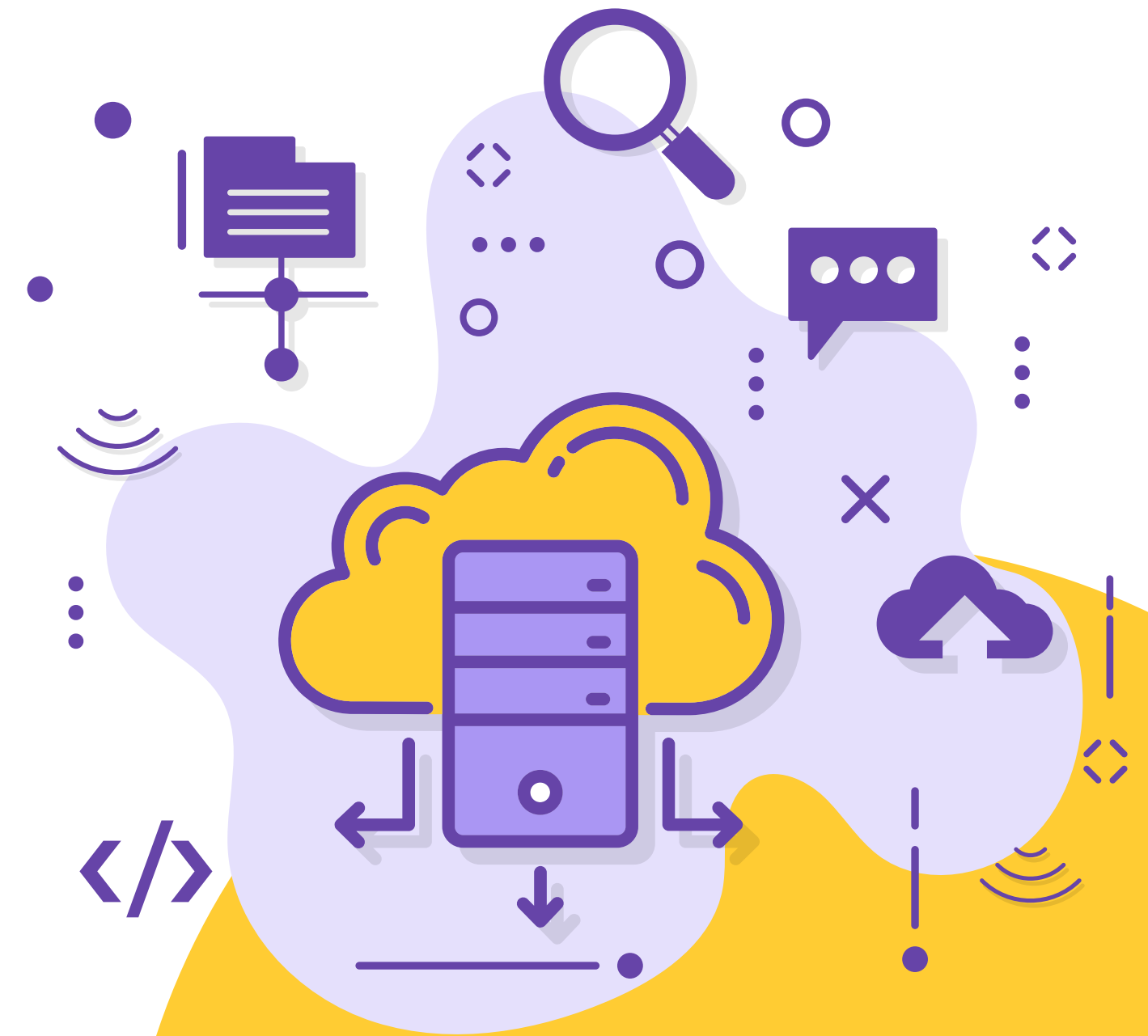


# What is Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services, networks, and volumes. Then, with a single command, you create and start all the services from your configuration.

## Why use it?

- *Simplicity*: Define your entire application stack in one file
- *Reproducibility*: Same environment every time, everywhere
- *Isolation*: Each component runs in its own container
- *Scalability*: Easy to scale services up or down
- *Maintainability*: Version control your infrastructure as code

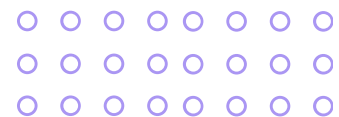
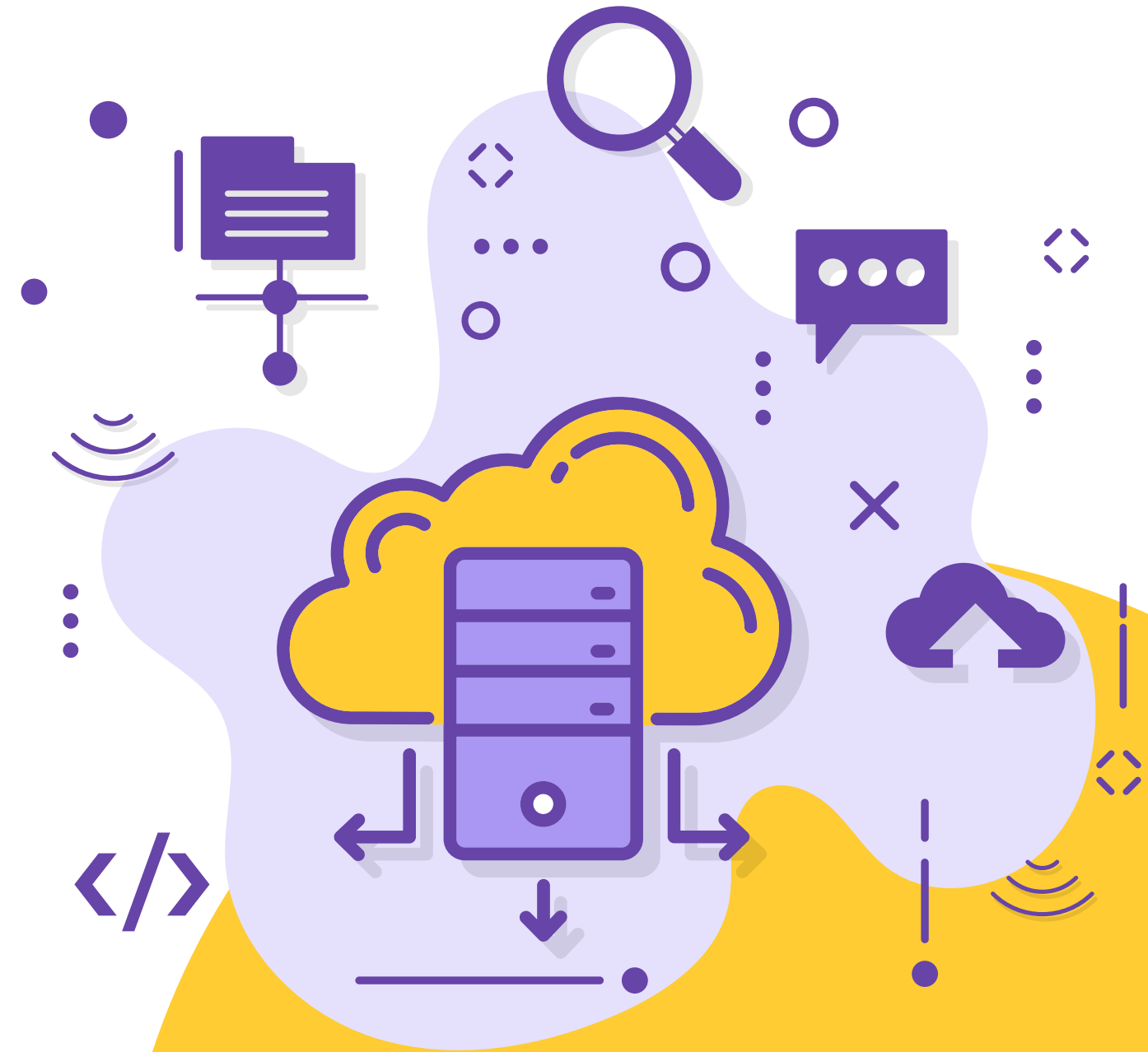


# Docker vs Docker Compose

## Key Differences:

- Docker: Manages individual containers
- Docker Compose: Orchestrates multiple containers as a single application

*Think of Docker as managing single instruments, while Docker Compose conducts the entire orchestra*



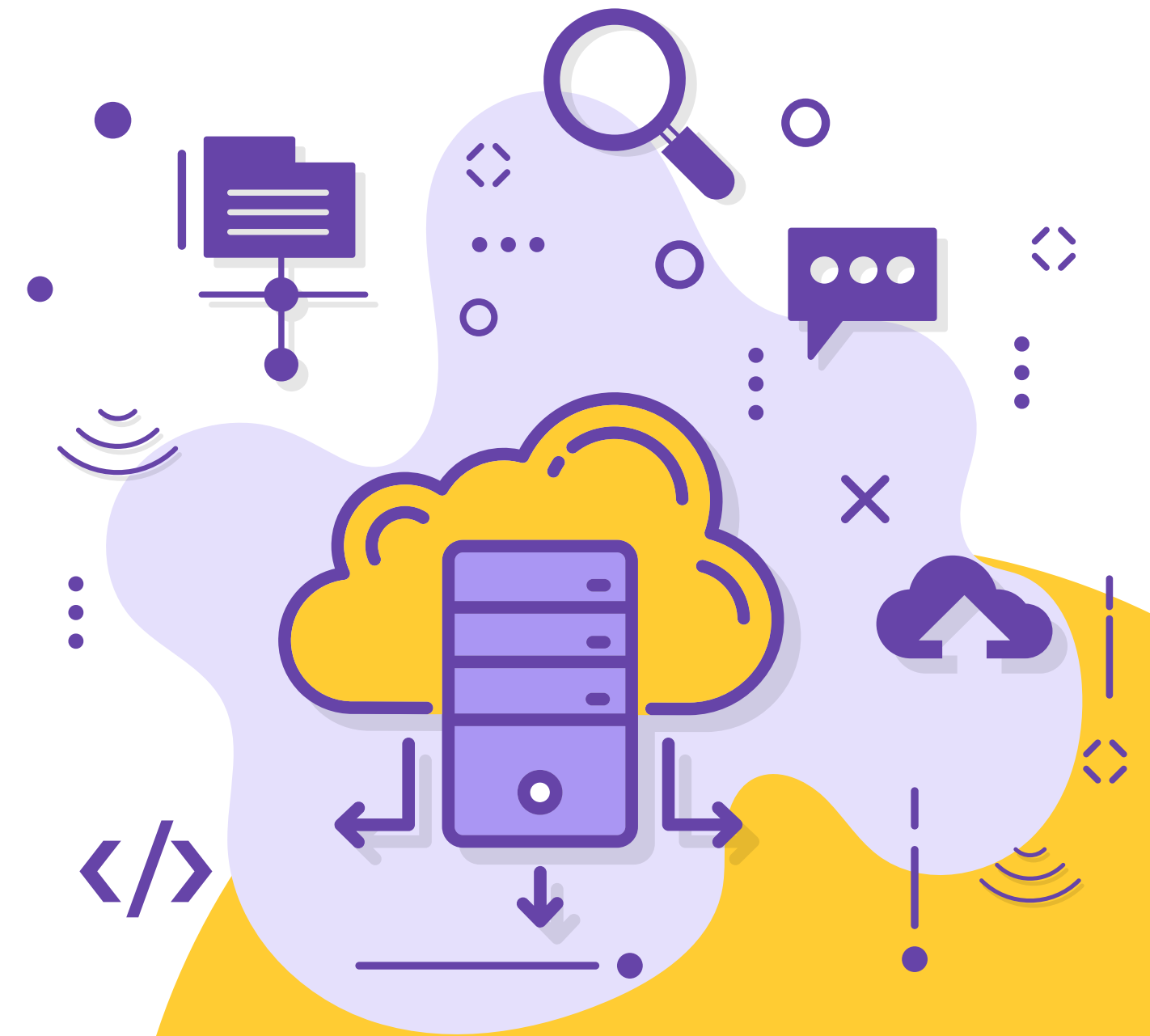
# Installation and Setup

## For Windows Users:

1. Install Docker Desktop for Windows from [Docker's official website](#)
2. Docker Compose is included with Docker Desktop for Windows
3. Verify installation by opening PowerShell or Command Prompt and running:
  - `docker-compose --version`

## For Mac Users:

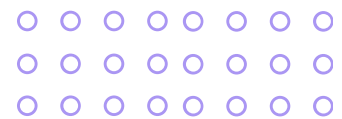
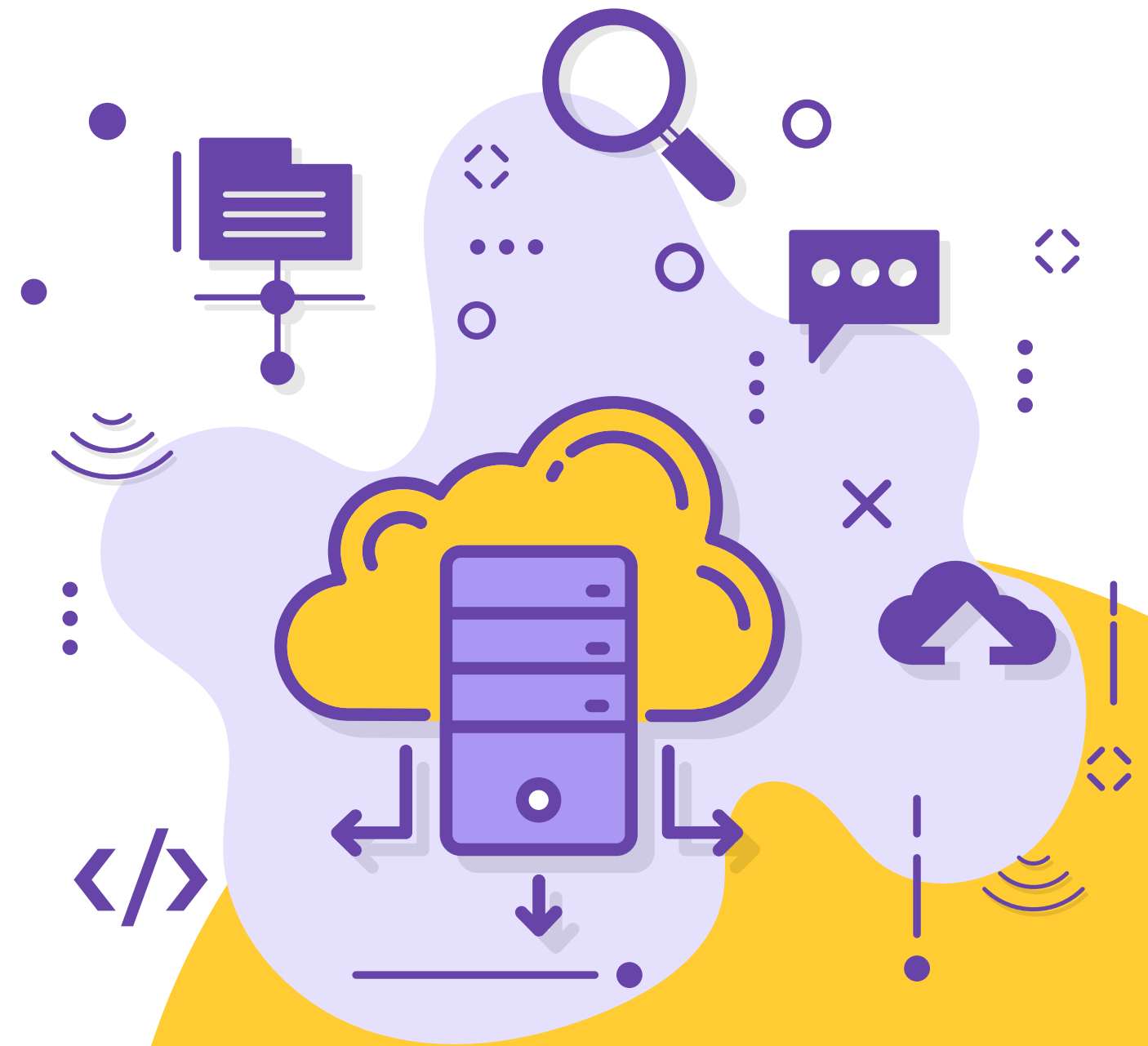
1. Install Docker Desktop for Mac from [Docker's official website](#)
2. Docker Compose is included with Docker Desktop for Mac
3. Verify installation by opening Terminal and running:
  - `docker-compose --version`



# Installation and Setup

## Note for Windows Users:

- Ensure virtualization is enabled in your BIOS
- Docker Desktop requires at least 4GB of RAM
- WSL 2 backend is recommended for Windows users



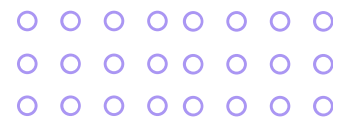
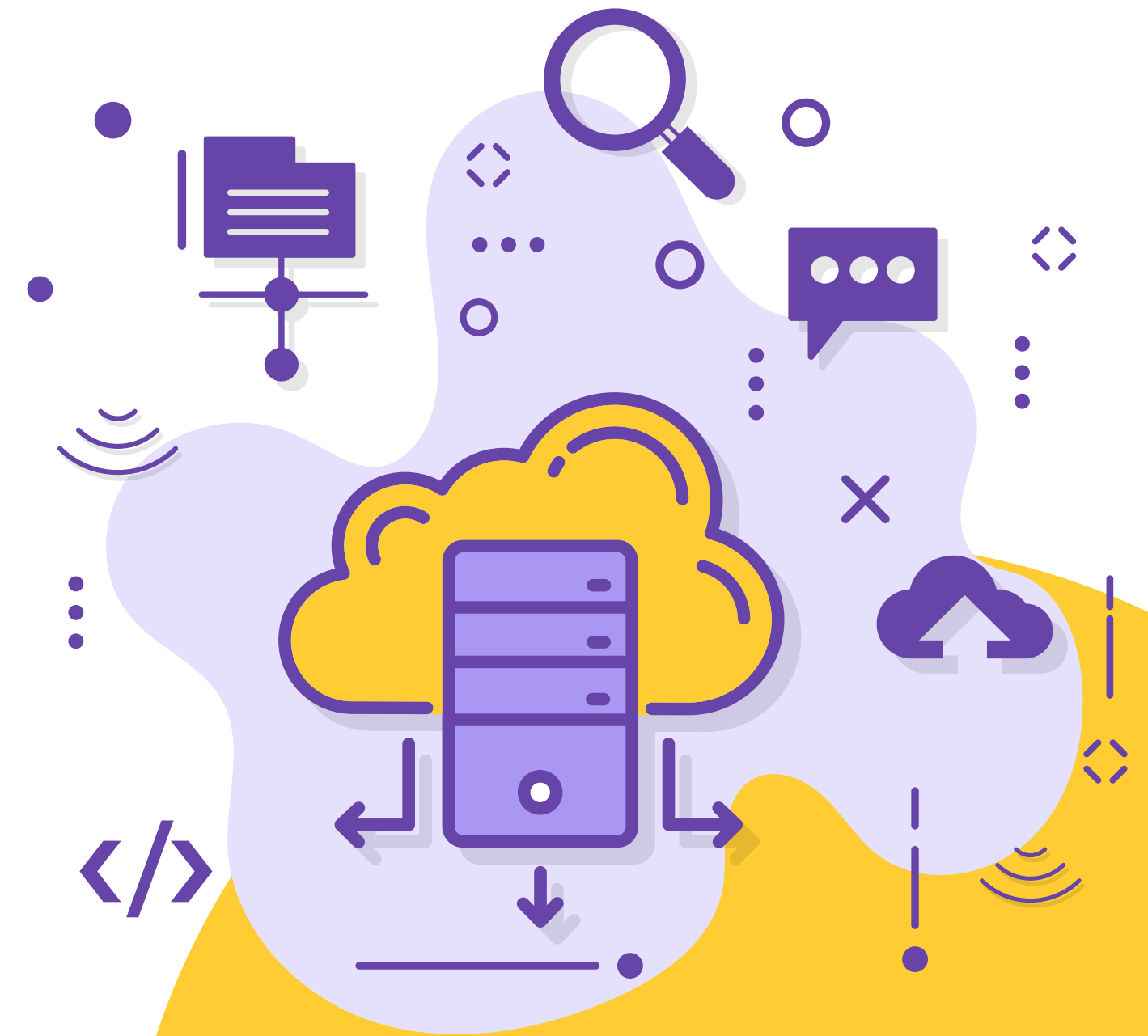
# Core Concepts

## Services

Services are the containers that make up your application. Each service runs a single image and can be configured with various options including environment variables, volumes, ports, and more.

## Networks

Networks enable communication between your containers and with the outside world. Docker Compose creates a default network for your application, but you can define custom networks with specific drivers and options.



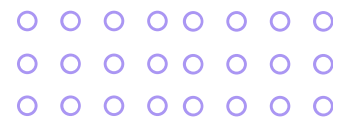
# Core Concepts

## Volumes

Volumes provide persistent storage for your containers. They can be used to share data between containers or to persist data even after a container is removed.

## Dependencies

Dependencies allow you to specify the order in which services should start. This is useful when one service depends on another being available before it can function properly.

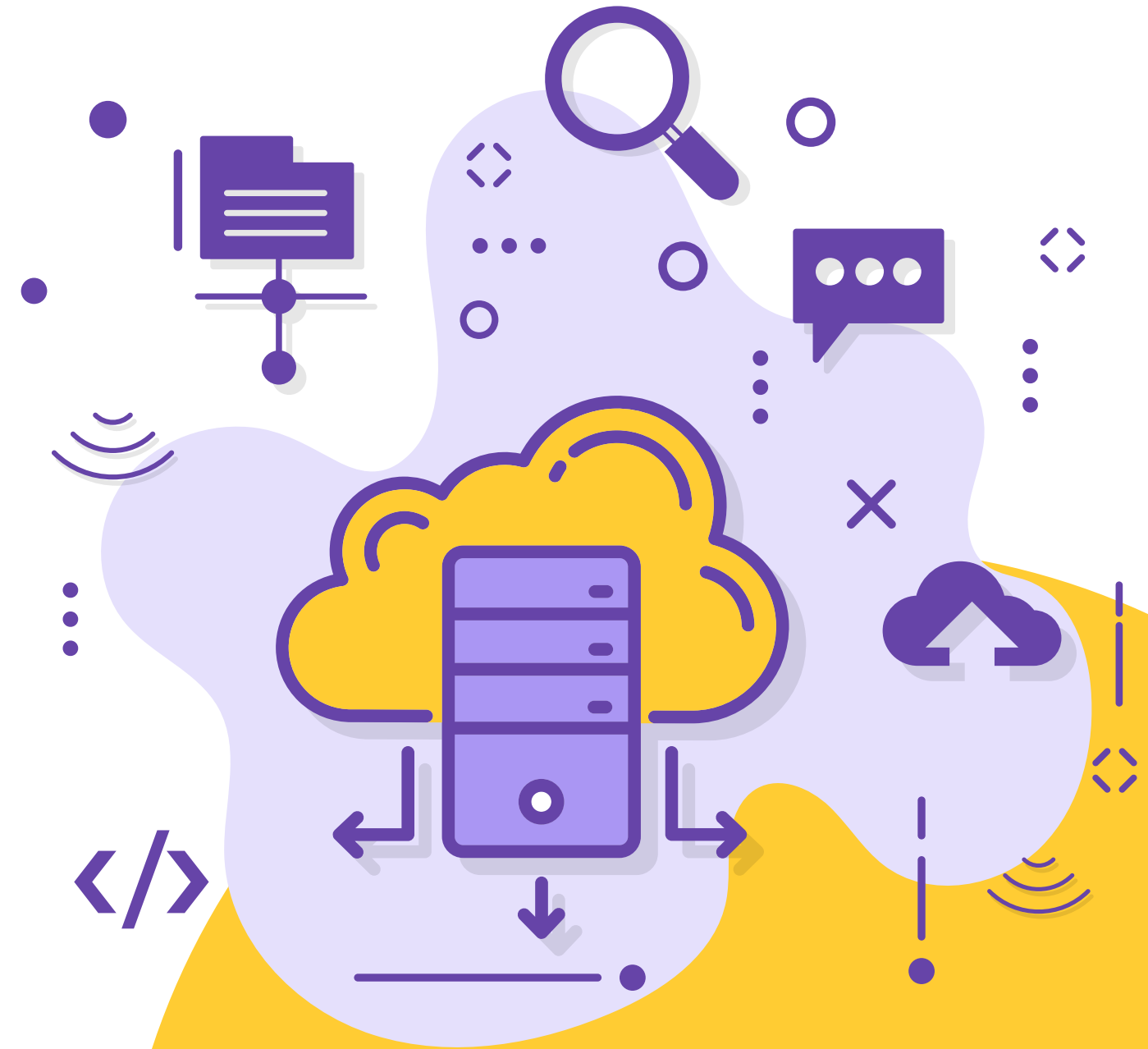




# Docker Compose File Structure

Docker Compose configurations are written in YAML format, typically in a file named ***docker-compose.yml***:

```
docker-compose.yml x
001-first-compose > docker-compose.yml > ...
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-
1  version: '3' # Compose file version
2
3  >Run All Services
4  services: # Container definitions
5    >Run Service
6    service1: # First service
7      image: nginx
8      ports:
9        - "8080:80"
10
11    >Run Service
12    service2: # Second service
13      build: ./app
14      volumes:
15        - ./app:/code
16      depends_on:
17        - service1
18
19  networks: # Network definitions
20    frontend:
21    backend:
22
23  volumes: # Volume definitions
24    data:
```



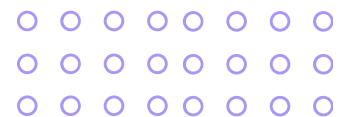
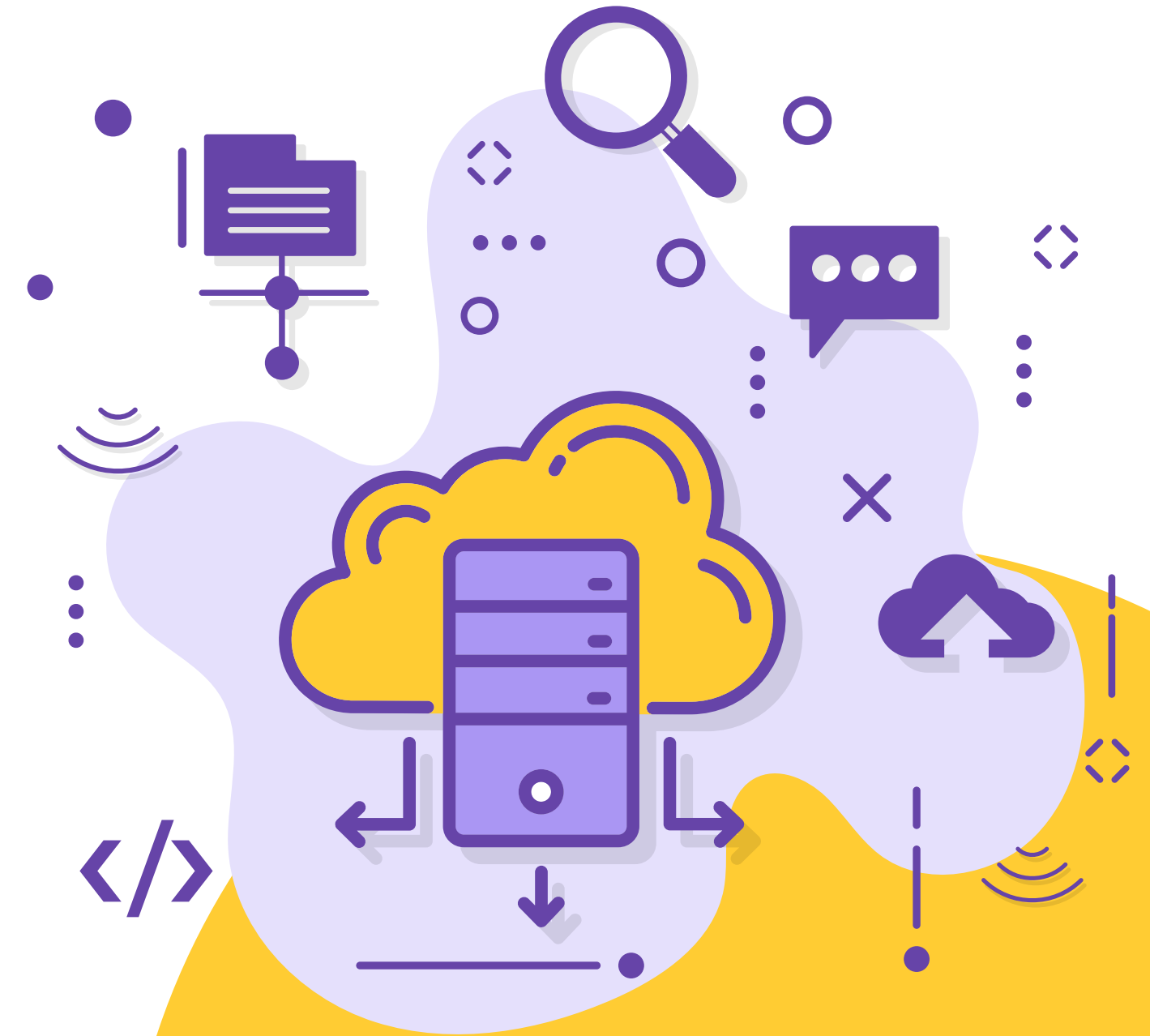
# Docker Compose

## Key Componentes of *docker-compose.yml* file:

- *version*: Specifies the Compose file format version
- *services*: Defines the containers you want to run
- *networks*: Configures networking for your application
- *volumes*: Defines persistent data storage
- *configs* and *secrets*: Manages configuration and sensitive data (in newer versions)

## Additional Concepts:

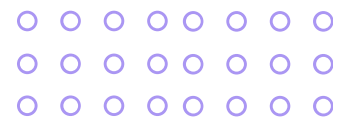
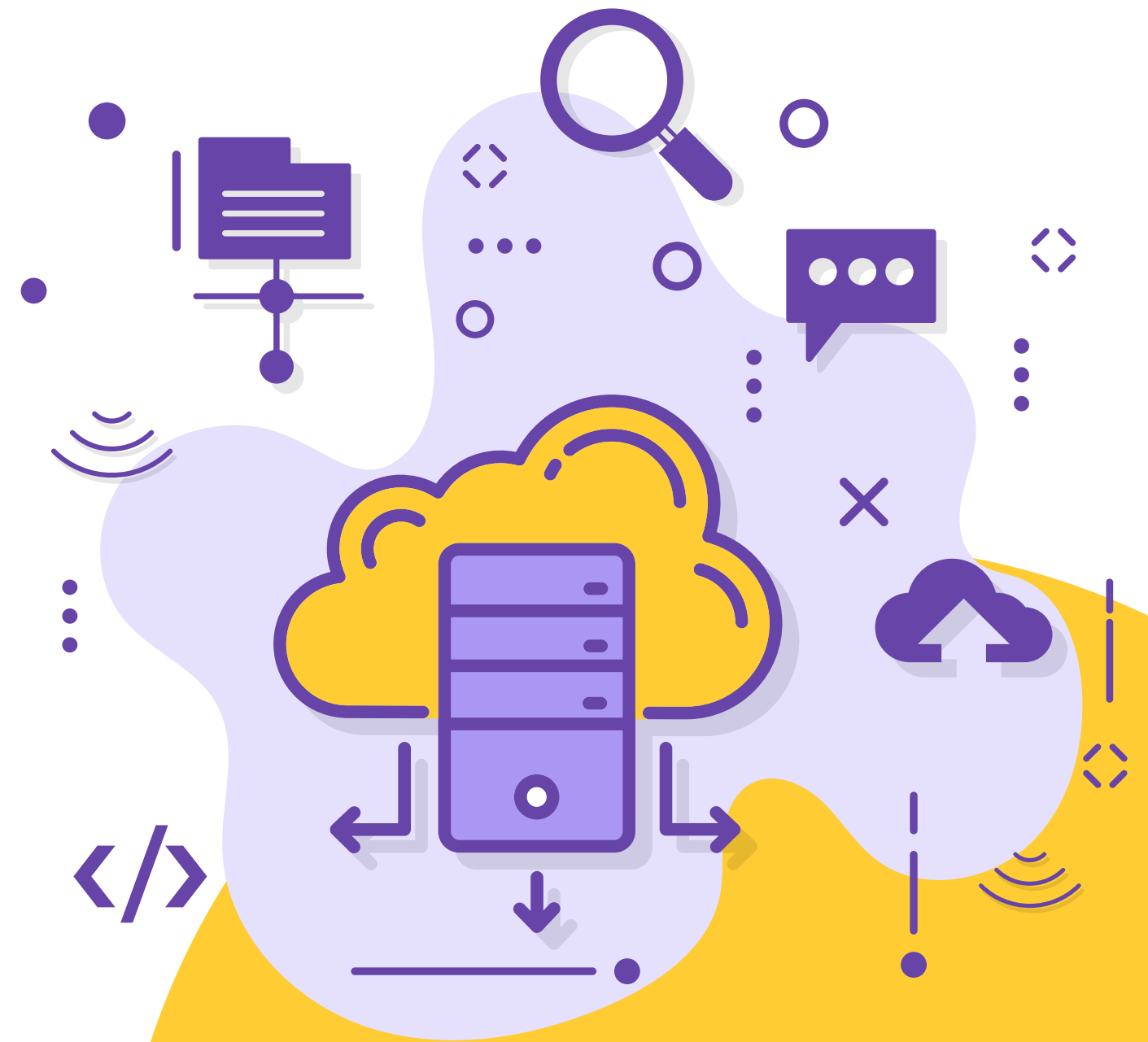
- Services share the same default network, enabling them to communicate by service name.
- Configuration is declarative — Compose figures out what to do based on file state.



# Hello Docker Compose

The *docker-compose.yml*:

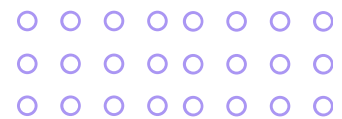
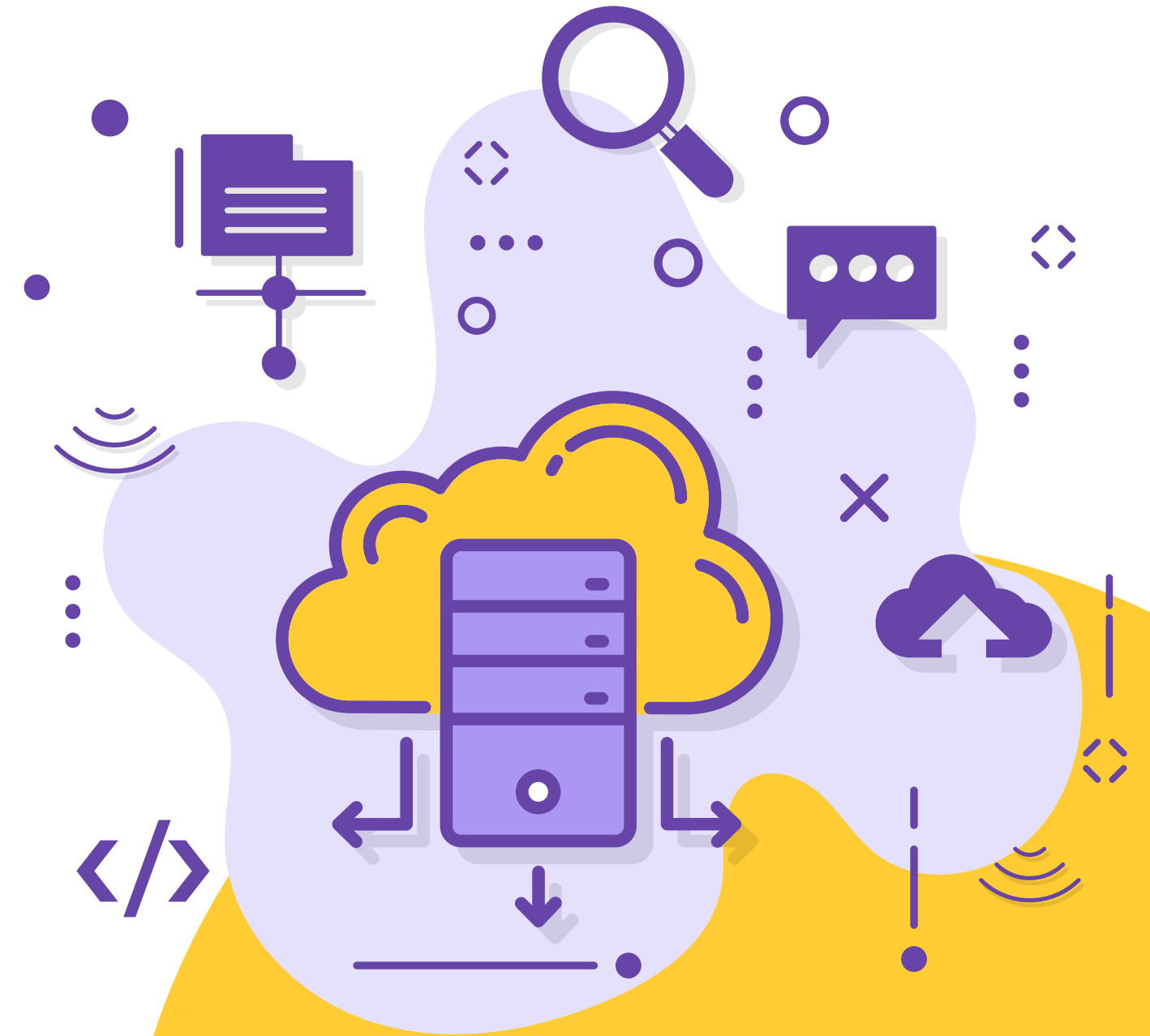
```
docker-compose.yml x
001-hello-compose > docker-compose.yml > YAML > {} services > {} web
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container pla
1  version: '3'
2
3  >Run All Services
4  services:
5    >Run Service
6    web:
7      image: nginx:alpine
8      ports:
9        - "8080:80"
10     volumes:
11       - ./website:/usr/share/nginx/html
12     restart: always
```



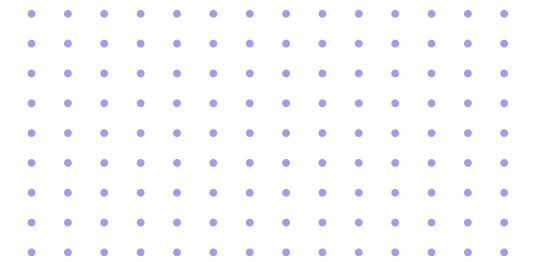
# Hello Docker Compose

The *index.html* file:

```
index.html x
001-hello-compose > website > index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Hello Docker Compose</title>
8  > <style>...
47 </style>
48 </head>
49
50 <body>
51   <div class="container">
52     <h1>Hello Docker Compose!</h1>
53     <p>This simple page is being served by Nginx running in a Docker container.</p>
54
55     <div class="docker-info">
56       <p><strong>Container:</strong> Nginx Alpine</p>
57       <p><strong>Port:</strong> 8080:80</p>
58       <p><strong>Status:</strong> Running</p>
59     </div>
60
61     <p>Congratulations! Your Docker Compose setup is working correctly.</p>
62   </div>
63 </body>
64
65 </html>
```



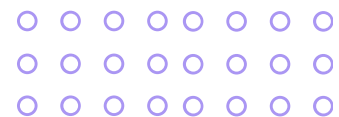
# What's Happening?



**When you run `docker-compose up -d`:**

- Docker downloads the *nginx:alpine* image (if it's not already available locally)
- It creates a container based on that *image*
- It maps *port 8080* on your computer to *port 80* on the container
- It mounts your local "*website*" folder in the directory where Nginx serves the files
- It starts the container in detached mode (*-d flag*)

As a result, Nginx inside the container serves the *index.html* file and we can access it from the browser at *localhost:8080*.



# Result of Hello Docker Compose

## Hello Docker Compose!

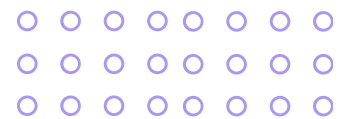
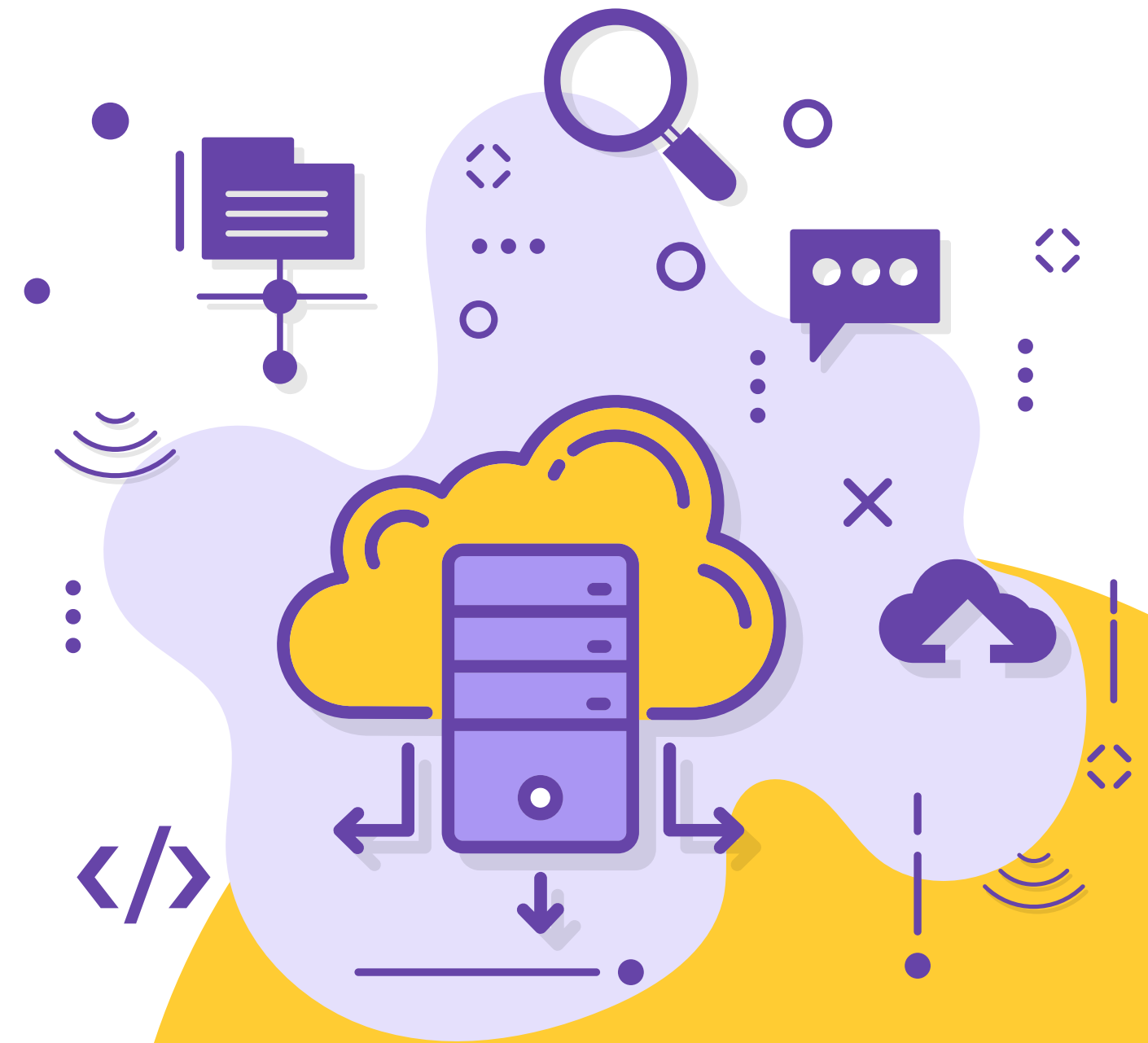
This simple page is being served by Nginx running in a Docker container.

**Container:** Nginx Alpine

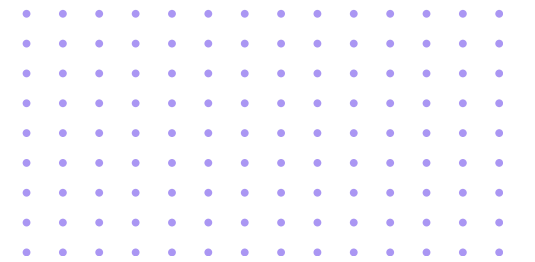
**Port:** 8080:80

**Status:** Running

Congratulations! Your Docker Compose setup is working correctly.



# Command Workflow & Lifecycle

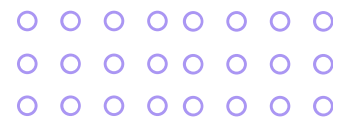


## Command Description:

- *docker-compose up* → Builds (if needed) and starts the services
- *docker-compose up -d* → Runs services in the background (detached mode)
- *docker-compose down* → Stops and removes containers, networks, volumes
- *docker-compose ps* → Lists services and status
- *docker-compose logs -f* → Tails logs from all services
- *docker-compose exec <service> bash* → Executes a command in a running container

## Behind the scenes Docker Compose creates:

- A custom bridge network
- Named volumes for persistence
- Container names prefixed with the project name



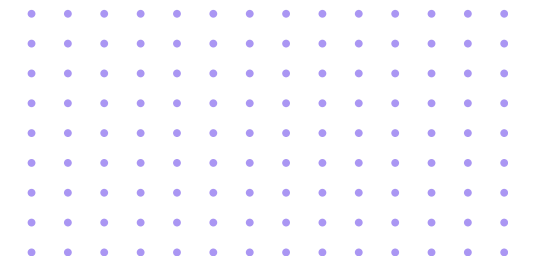
# Updating Hello Docker Compose

New *docker-compose.yml*:

```
docker-compose.yml x
002-phpmyadmin > docker-compose.yml > ...
 2  services:
16    db:
24    volumes:
27    networks:
28      - backend
29    restart: always
30    ports:
31      - "3306:3306"
32
  > Run Service
33  phpmyadmin:
34    image: phpmyadmin/phpmyadmin
35    ports:
36      - "8081:80"
37    environment:
38      PMA_HOST: db
39      PMA_PORT: 3306
40      MYSQL_ROOT_PASSWORD: rootpassword
41      # Remove auto-login to allow logging in as root
42      # PMA_USER: user
43      # PMA_PASSWORD: password
44    depends_on:
45      - db
46    networks:
47      - frontend
48      - backend
49    restart: always
50
51  networks:
52    frontend:
53    backend:
54
55  volumes:
56    db_data:
57
```

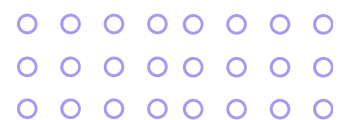


# Updating Hello Docker Compose

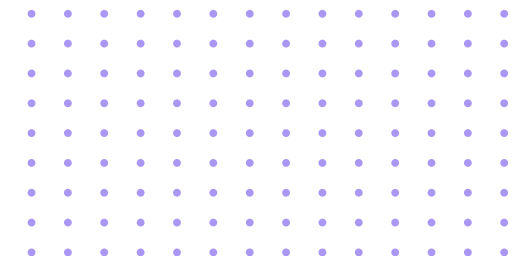


mysql-init/grant\_privileges.sql:

```
grant_privileges.sql ×  
002-phpmyadmin > mysql-init > grant_privileges.sql  
1 GRANT ALL PRIVILEGES ON *.* TO 'user'@'%';  
2 FLUSH PRIVILEGES;  
3
```



# Updating Hello Docker Compose



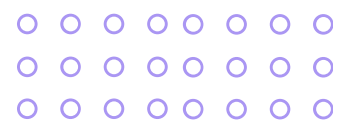
## Explanation of the New Docker Compose File

Version Declaration:

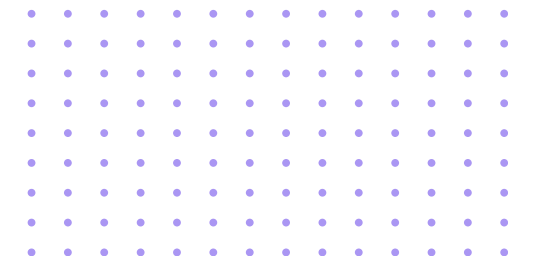
- *version*: '3' → This specifies the Docker Compose file format version. Version 3 is commonly used and supports most modern Docker features.

Services Section:

- This section defines three containers (services) that will work together:
  - Web Server (Nginx)
    - *image*: Uses the lightweight Alpine Linux version of Nginx
    - *ports*: Maps port 8080 on your host to port 80 in the container
    - *volumes*: Maps your local ./website folder to Nginx's web root
    - *depends\_on*: Ensures the database starts before this service
    - *networks*: Connects to both frontend and backend networks
    - *restart*: Automatically restarts the container if it stops

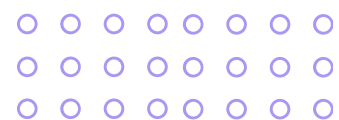


# Updating Hello Docker Compose

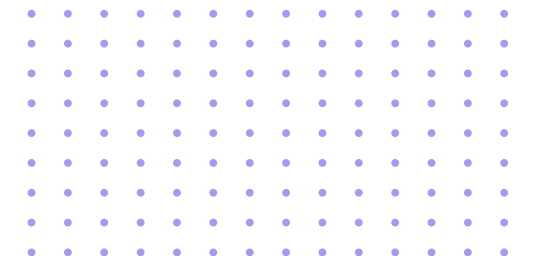


Services Section (continuation):

- Database Server (MySQL)
  - *image*: Uses MySQL version 8.0
  - *command*: Changes the authentication plugin to be compatible with more clients
  - *environment*: Sets up:
    - Root password
    - Initial database named "myapp"
    - A non-root user account with limited permissions
  - *volumes*:
    - db\_data: A named volume to persist database files
    - ./mysql-init:/docker-entrypoint-initdb.d: For SQL initialization scripts
  - *networks*: Only on the backend network for security
  - *ports*: Exposes MySQL on port 3306 (optional, for external access)
  - *restart*: Automatically restarts if it fails

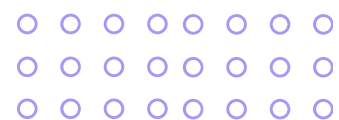


# Updating Hello Docker Compose

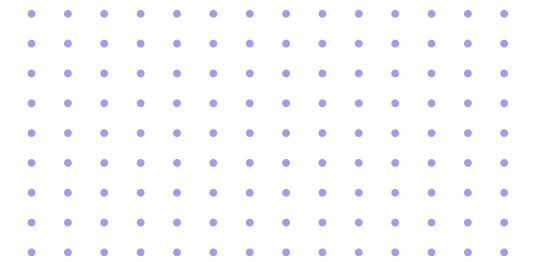


Services Section (continuation):

- Database Management Tool (phpMyAdmin)
  - *image*: Uses the official phpMyAdmin image
  - *ports*: Maps port 8081 on your host to port 80 in the container
  - *environment*:
    - PMA\_HOST: Tells phpMyAdmin which database server to connect to
    - PMA\_PORT: The port MySQL is running on
    - MYSQL\_ROOT\_PASSWORD: Passes the root password to phpMyAdmin
    - The commented-out PMA\_USER and PMA\_PASSWORD lines are the key change
  - *depends\_on*: Ensures the database starts before phpMyAdmin
  - *networks*: Connected to both networks
  - *restart*: Automatically restarts if it fails



# Updating Hello Docker Compose



## Network Section:

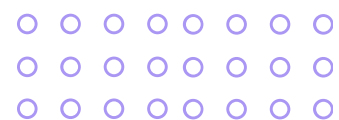
- Database Management Tool (phpMyAdmin)
  - *Creates two separate network spaces:*
    - *frontend: For external access to web servers*
    - *backend: For internal communication between services*

## Volumes Section:

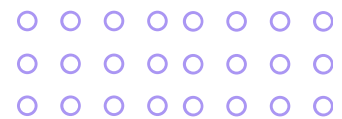
- Creates a named volume to store MySQL data persistently, even when containers are removed.

## How to Access:

- Website: localhost:8080
- phpMyAdmin: localhost:8081
  - On the phpMyAdmin login screen, use:
    - Server: db (pre-filled)
    - Username: root
    - Password: rootpassword



Questions?



# Learning More About Docker Compose

## Official Documentation

- Docker Compose Documentation
  - <https://docs.docker.com/compose/>
- Compose File Reference
  - <https://docs.docker.com/compose/compose-file/>
- Docker Compose CLI Reference
  - <https://docs.docker.com/compose/reference/>

## Interactive Learning

- Play with Docker – Free online Docker playground
  - <https://labs.play-with-docker.com/>

