Docker Compose Cheatsheet

A comprehensive guide to Docker Compose, covering essential commands, configurations, and best practices for defining and managing multi-container Docker applications.

Basic Concepts & Setup

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. Then, with a single command, you create and start all the services from your configuration.

Key benefits include:

- Simplified multi-container management.
- Infrastructure as code.
- Reproducible environments.

Installation

Docker Compose is now integrated into Docker Desktop. Ensure Docker Desktop is installed and running. For standalone installation (if needed):

```
# Example for Linux
sudo apt-get update
sudo apt-get install docker-compose-plugin
```

Docker Compose File (docker-compose.yml)

The `docker-compose.yml` file defines the services, networks, and volumes for your application. Here's a basic structure:

```
version: '3.8'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
```

Essential Commands

Lifecycle Management

`docker compose up`	Builds, (re)creates, starts, and attaches to containers for all services defined in the `docker-compose.yml` file. Flags: `-d` (detached mode).
`docker compose down`	Stops and removes containers, networks, volumes, and images created by `up`.
`docker compose start`	Starts existing containers.
`docker compose stop`	Stops running containers without removing them.
`docker compose restart`	Restarts all services.

Service Interaction

`docker compose ps`	Lists the status of the containers.
`docker compose logs`	View output from the containers. Service can be specified `docker compose logs <service>`.
`docker compose exec`	Execute a command in a running container. Example: `docker compose exec web bash`
`docker compose run`	Run a one-off command against a service. Example: `docker compose run web python manage.py migrate`

Configuration Inspection

`docker compose config`	Validate and view the Compose file configuration. Useful for verifying your setup.
`docker compose version`	Displays the Docker Compose version.

Configuration Options

Build Configuration

Use the `build` directive to configure how a service is built from a Dockerfile.

```
version: '3.8'
services:
  web:
    build:
      context: ./web
      dockerfile: Dockerfile.dev
      args:
        NODE_ENV: development
```

- `context`: Path to the build context (directory containing the Dockerfile).
- `dockerfile`: Name of the Dockerfile (defaults to `Dockerfile`).
- `args`: Build-time arguments.

Image Configuration

Specify a pre-built image using the `image` directive:

```
version: '3.8'
services:
  web:
    image: nginx:latest
```

You can also specify a private registry:

```
image: your-registry.com/your-image:tag
```

Port Mapping

Expose ports from the container to the host machine:

```
version: '3.8'
services:
  web:
    ports:
      - "80:80"  # host:container
      - "443:443"
```

Use `expose` to expose ports between linked services (not accessible from the host):

```
expose:
  - "3000"
```

Volumes

Share directories or volumes between the host and containers.

```
version: '3.8'
services:
  web:
    volumes:
      - ./app:/var/www/html  #
host_path:container_path
      - data-volume:/data      # named volume

volumes:
  data-volume:
```

Environment Variables

Set environment variables for services.

```
version: '3.8'
services:
  web:
    environment:
      -
DATABASE_URL=postgres://user:pass@db:5432
      - API_KEY=${API_KEY}
    env_file:
      - .env
```

- `environment`: Define variables directly in the Compose file.
- `env_file`: Load variables from one or more `.env` files.
- `${VARIABLE}`: Use environment variables from the host system.

Advanced Configuration

Dependencies & Health Checks

Define service dependencies and health checks to ensure proper startup order and service availability.

```
version: '3.8'
services:
  web:
    depends_on:
      db:
        condition: service_healthy
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
```

- `depends_on`: Define service dependencies and startup order. Conditions: `service_healthy`, `service_started`.
- `healthcheck`: Define how Docker determines if a service is healthy.

Networks

Create custom networks for inter-container communication.

```
version: '3.8'
services:
  web:
    networks: [frontend]
  db:
    networks: [frontend]

networks:
  frontend:
    driver: bridge
```

- `networks`: Specify which networks a service belongs to.
- `driver`: Network driver (e.g., `bridge`, `overlay`).

Extending Services

Use `extends` to share configurations between services.

```
version: '3.8'
services:
  web:
    extends:
      file: common-config.yml
      service: webapp
```

- `file`: Path to the configuration file containing the base service.
- `service`: Name of the service to extend.

Resource Limits

Limit the resources a container can use.

```
version: '3.8'
services:
  web:
    deploy:
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
```

- `cpus`: CPU limit (e.g., `0.5` for 50% of a CPU core).
- `memory`: Memory limit (e.g., `512M`, `1G`).