Containers & Kubernetes

Session #04

Prune

Docker compose

Lab

Prune

Maintenance

Docker prune

- The main purpose of the docker prune family of commands is to clean up unused Docker resources to reclaim disk space
- Over time, as you build and run containers, your system accumulates many stopped containers, unused images and unreferenced networks or volumes
- It finds resources that are not currently associated with any active container.
- It deletes these unused resources in bulk.
- This is crucial for maintaining a healthy and efficient Docker host, especially on development machines or CI/CD runners where resources are created and torn down frequently.

docker system prune

Docker prune

- All stopped containers
- All unused networks (not associated with at least one container)
- All dangling images (images with no tags and not used by any container)
- All build cache

docker container prune

Docker prune

- Removes only all stopped containers

docker image prune

Docker prune

- Removes only dangling images.
- Use **`docker image prune -a`** (or --all) to remove all unused images, not just dangling ones
- This includes images that are not associated with any existing container.

docker volume prune

Docker prune

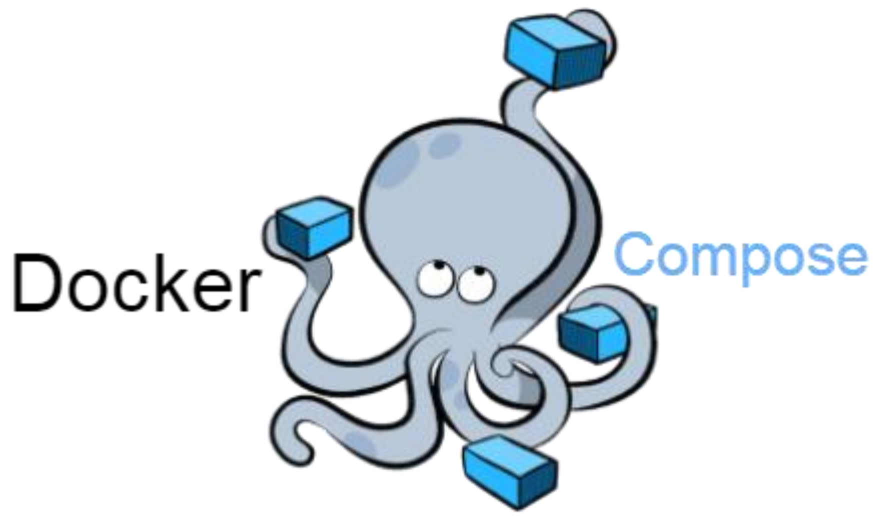
- Removes only unused local volumes (volumes not used by at least one container).
- By default, **`docker system prune`** does not remove unused volumes to prevent data loss
- You must run **`docker volume prune`** separately or use the **`--volumes`** flag (e.g., **`docker system prune --volumes`**) to include them in the system-wide prune.

Docker Compose

What it is?

Docker compose

- Docker Compose is a tool for defining and running complex applications with Docker.
- Define a multi-container application in a single file
- Spin your application up in a single command



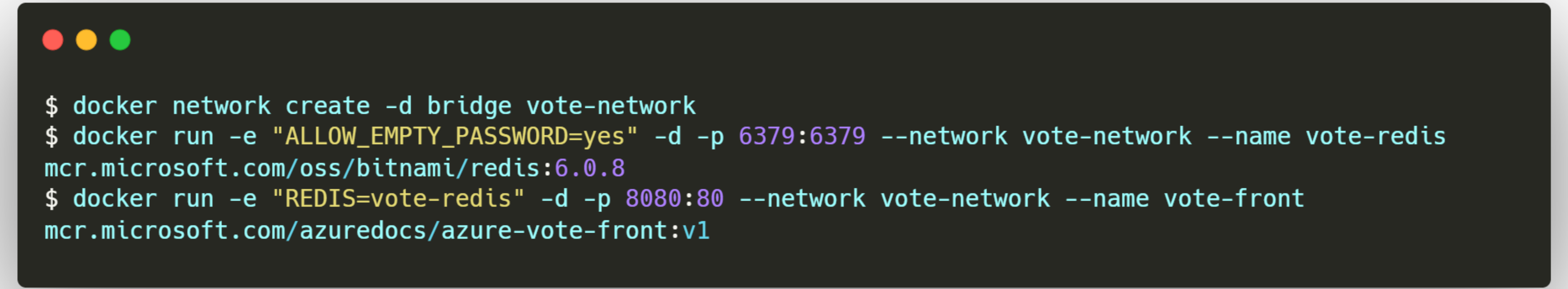
Features

Docker compose

- Multiple isolated environments on a single host
- Preserve volume data when containers are created
- Only recreate containers that have changed
- Variables and moving a composition between environments
- Multiple compose files
- Allow to handle containers lifecycle

Before…

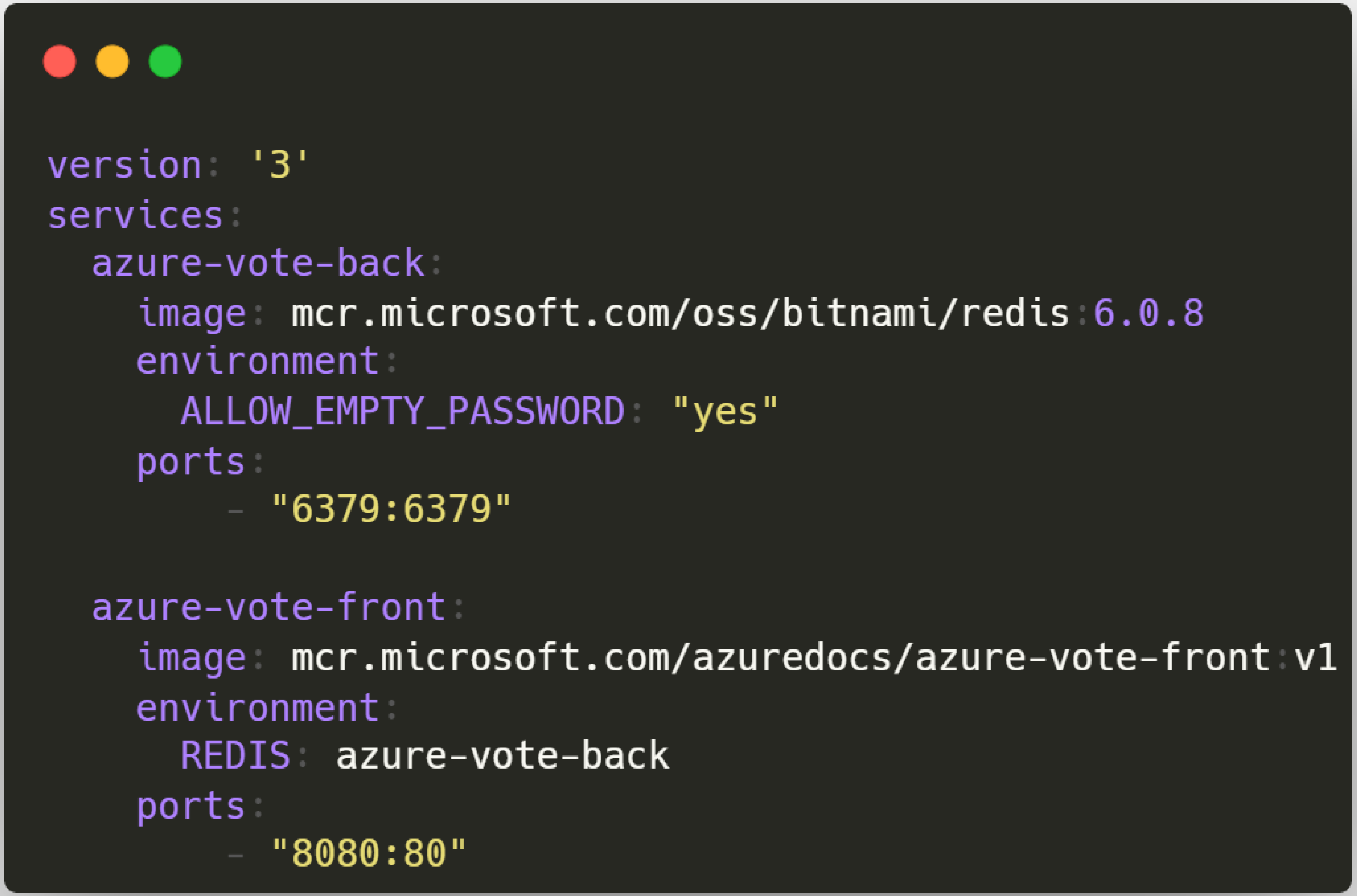
Docker compose



```
$ docker network create -d bridge vote-network
$ docker run -e "ALLOW_EMPTY_PASSWORD=yes" -d -p 6379:6379 --network vote-network --name vote-redis
mcr.microsoft.com/oss/bitnami/redis:6.0.8
$ docker run -e "REDIS=vote-redis" -d -p 8080:80 --network vote-network --name vote-front
mcr.microsoft.com/azuredocs/azure-vote-front:v1
```

After…

Docker compose



```yaml
version: '3'
services:
  azure-vote-back:
    image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
    environment:
      ALLOW_EMPTY_PASSWORD: "yes"
    ports:
        - "6379:6379"

  azure-vote-front:
    image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
    environment:
      REDIS: azure-vote-back
    ports:
        - "8080:80"
```

YAML file

Docker compose

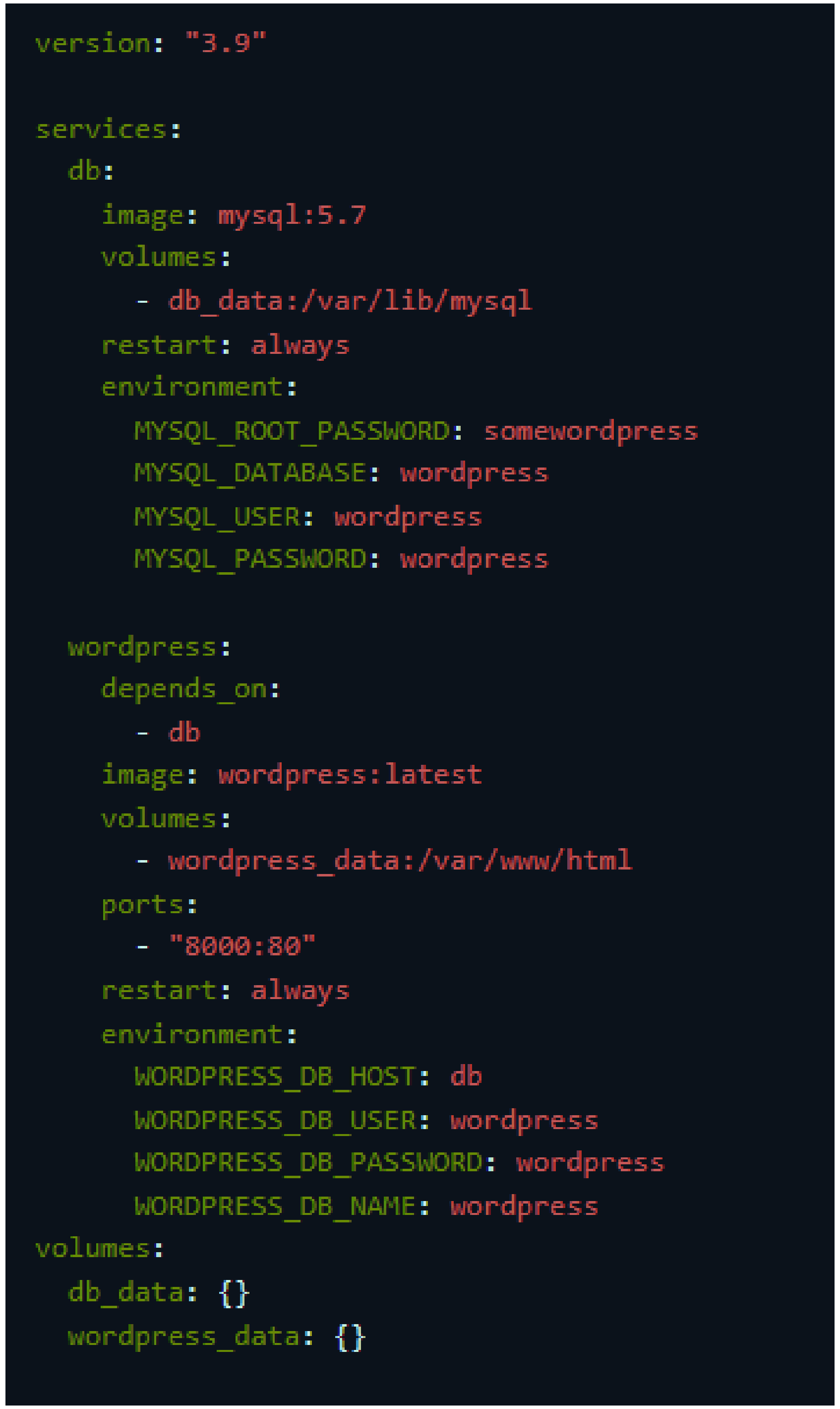
Version

Services

- Build
- Image
- Environment
- Ports
- Volumes

Volumes

Networks



```yaml
version: "3.9"

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - wordpress_data:/var/www/html
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
  wordpress_data: {}
```

Commands

Docker compose

Create and start all the containers listed in the "docker-compose.yml"

```
$ docker-compose up -d
```

List all the containers belong to the compose environment instance

```
$ docker-compose ps
```

Stop all containers started by docker-compose

```
$ docker-compose down
```

Demo: Docker Compose