docker

````
# Docker Cheatsheet

## Images

```bash
docker images                          # lister les images locales
docker pull nginx:alpine               # télécharger une image
docker build -t monapp:v1 .            # build depuis Dockerfile
docker build -t monapp:v1 -f Dev.dockerfile .  # Dockerfile custom
docker rmi <image_id>                  # supprimer une image
docker image prune -a                  # virer les images non utilisées
````

Containers

```
docker ps                              # containers running
docker ps -a                           # tous les containers
docker run -d --name web nginx         # run en background
docker run -it ubuntu bash             # run interactif
docker run -d -p 8080:80 nginx         # port mapping host:container
docker run -d -v /host/path:/container/path nginx  # bind mount
docker run -d -v myvolume:/data nginx  # named volume
docker run --rm nginx                  # auto-delete après stop
docker run -e NODE_ENV=prod node       # variable d'env
docker run --env-file .env node        # fichier d'env

docker exec -it <container> bash       # shell dans container running
docker logs -f <container>             # logs en live
docker stop <container>                # SIGTERM graceful
docker kill <container>                # SIGKILL brutal
docker rm <container>                  # supprimer
docker rm -f <container>               # stop + rm
```
```

Volumes & Networks

```
docker volume ls
docker volume create mydata
docker volume inspect mydata
docker volume rm mydata

docker network ls
docker network create mynet
docker network connect mynet <container>
docker run -d --network mynet nginx    # attach au run
```

Docker Compose

```
docker compose up -d                   # démarrer la stack
docker compose down                    # stop + rm containers
docker compose down -v                 # + supprime les volumes
docker compose logs -f                 # logs de tous les services
docker compose ps                      # état des services
docker compose exec web bash           # exec dans un service
docker compose build                   # rebuild les images
docker compose pull                    # pull les nouvelles images
docker compose up -d --build           # rebuild + restart
```

Debugging

```
docker inspect <container>             # JSON complet du container
docker stats                           # CPU/RAM/IO en live
docker top <container>                 # processes dans le container
docker diff <container>                # fichiers modifiés vs image
docker cp <container>:/path ./local    # copier fichier depuis container
docker history <image>                 # layers de l'image
```

Cleanup

```
docker system df                       # espace disque utilisé
docker system prune                    # containers stoppés + images dangling
docker system prune -a --volumes       # TOUT nettoyer (attention)
```

Dockerfile essentials

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3000
USER node
CMD ["node", "server.js"]
```

Compose essentials

```
services:
  web:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
    volumes:
      - ./src:/app/src
    depends_on:
      - db
    restart: unless-stopped

  db:
    image: postgres:16-alpine
    volumes:
      - pgdata:/var/lib/postgresql/data
    environment:
      POSTGRES_PASSWORD: secret
```

```
volumes:
  pgdata:
```

Docker Swarm Cheatsheet

Concepts clés

```
┌─────────────────────────────────────────────────────────┐
│                      SWARM CLUSTER                        │
│                                                           │
│   ┌─────────────┐  ┌─────────────┐  ┌─────────────┐      │
│   │   Manager   │  │   Manager   │  │   Manager   │      │
│   │  (leader)   │◄─┤  (replica)  ├──┤  (replica)  │      │
│   └──────┬──────┘  └─────────────┘  └─────────────┘      │
│          │              Raft consensus                     │
│   ┌──────┴─────────────────┬───────────────────┐         │
│   ▼                        ▼                   ▼         │
│ ┌──────────┐         ┌──────────┐        ┌──────────┐   │
│ │  Worker  │         │  Worker  │        │  Worker  │   │
│ │ [Task]   │         │ [Task]   │        │ [Task]   │   │
│ │ [Task]   │         │ [Task]   │        │ [Task]   │   │
│ └──────────┘         └──────────┘        └──────────┘   │
└─────────────────────────────────────────────────────────┘

Stack    = groupe de services (1 docker-compose.yml = 1 stack)
Service  = définition d'une app (image + config + replicas)
Task     = instance d'un service (= 1 container qui tourne)
```

Initialisation & Nodes

```
docker swarm init                              # init swarm (devient manager)
docker swarm init --advertise-addr 192.168.1.10  # si plusieurs IPs

docker swarm join-token worker                 # affiche token pour ajouter worker
```

```
docker swarm join-token manager               # token pour ajouter manager
docker swarm join --token <token> <manager-ip>:2377  # rejoindre le swarm

docker node ls                                # lister les nodes
docker node inspect <node>                    # détails d'un node
docker node promote <node>                    # worker → manager
docker node demote <node>                     # manager → worker
docker node update --availability drain <node> # vider un node (maintenance)
docker node update --availability active <node> # remettre en service
docker node rm <node>                         # retirer un node

docker swarm leave                            # quitter le swarm (depuis worker)
docker swarm leave --force                    # quitter (depuis manager)
```

Services

```
# Créer un service
docker service create --name web nginx:alpine
docker service create --name web -p 8080:80 nginx          # port mapping
docker service create --name web --replicas 3 nginx        # 3 instances
docker service create --name api -e NODE_ENV=prod node     # env var
docker service create --name db --mount type=volume,src=dbdata,dst=/var/lib/mysql mysql

# Gérer les services
docker service ls                             # lister les services
docker service ps <service>                   # voir les tasks d'un service
docker service logs -f <service>              # logs (toutes les tasks)
docker service inspect <service>              # config complète JSON

# Scaling
docker service scale web=5                    # scale à 5 replicas
docker service scale web=5 api=3              # scale plusieurs services

# Update
docker service update --image nginx:1.25 web   # rolling update
docker service update --replicas 10 web        # changer replicas
docker service update --env-add DEBUG=1 web    # ajouter env var
docker service update --env-rm DEBUG web       # supprimer env var
```

```
docker service update --rollback web           # rollback dernière update

# Supprimer
docker service rm web                          # supprimer le service
```

Stacks (déploiement via Compose)

```
docker stack deploy -c docker-compose.yml myapp    # déployer une stack
docker stack ls                                    # lister les stacks
docker stack services myapp                        # services de la stack
docker stack ps myapp                              # tasks de la stack
docker stack rm myapp                              # supprimer la stack
```

Compose pour Swarm (v3+)

```
version: "3.8"

services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    deploy:
      mode: replicated          # ou 'global' (1 par node)
      replicas: 3
      update_config:
        parallelism: 1          # 1 task à la fois
        delay: 10s              # attendre entre chaque
        failure_action: rollback
        order: start-first      # new avant de kill old
      rollback_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
        delay: 5s
```

```
        max_attempts: 3
      resources:
        limits:
          cpus: '0.5'
          memory: 256M
        reservations:
          cpus: '0.25'
          memory: 128M
      placement:
        constraints:
          - node.role == worker
          - node.labels.region == eu-west
    networks:
      - frontend

  api:
    image: myapi:latest
    deploy:
      replicas: 2
    secrets:
      - db_password
    configs:
      - source: api_config
        target: /app/config.json
    networks:
      - frontend
      - backend

  db:
    image: postgres:16-alpine
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
          - node.role == manager    # ou un node spécifique
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
```

```
      - backend

networks:
  frontend:
    driver: overlay
  backend:
    driver: overlay
    internal: true              # pas d'accès externe

volumes:
  pgdata:

secrets:
  db_password:
    external: true              # créé manuellement avant

configs:
  api_config:
    file: ./config.json
```

Secrets & Configs

```
# Secrets (données sensibles, chiffrées)
echo "monsecret" | docker secret create db_password -
docker secret create ssl_cert ./cert.pem       # depuis fichier
docker secret ls
docker secret inspect db_password
docker secret rm db_password

# Configs (fichiers de config non sensibles)
docker config create nginx_conf ./nginx.conf
docker config ls
docker config inspect nginx_conf
docker config rm nginx_conf

# Utilisation dans service
docker service create --name db \
  --secret db_password \
```

```
  --config source=my_config,target=/etc/app/config.json \
  postgres
```

Réseau

```
docker network create -d overlay mynetwork     # réseau overlay pour swarm
docker network create -d overlay --internal private_net  # pas de sortie externe
docker network ls
docker network inspect mynetwork

# Le routing mesh expose automatiquement les ports sur TOUS les nodes
# Même si le container tourne sur node2, tu peux y accéder via node1:port
```

Debugging

```
docker service ps <service> --no-trunc         # voir erreurs complètes
docker service logs <service> 2>&1 | grep error
docker inspect <task_id>                       # inspecter une task spécifique
docker node ps <node>                          # tasks sur un node
docker events                                  # events en temps réel
```

Commandes utiles au quotidien

```
# Forcer re-déploiement (pull nouvelle image si même tag)
docker service update --force web

# Update avec nouvelle image
docker service update --image myapp:v2 --update-parallelism 2 --update-delay 30s web

# Voir où tournent les tasks
docker service ps web --format "{{.Node}}: {{.CurrentState}}"

# Drainer un node avant maintenance
docker node update --availability drain node-03
# ... maintenance ...
docker node update --availability active node-03
```

Révision #1
Créé 2026-01-15 15:32:43 UTC par tjiho
Mis à jour 2026-01-15 15:54:03 UTC par tjiho