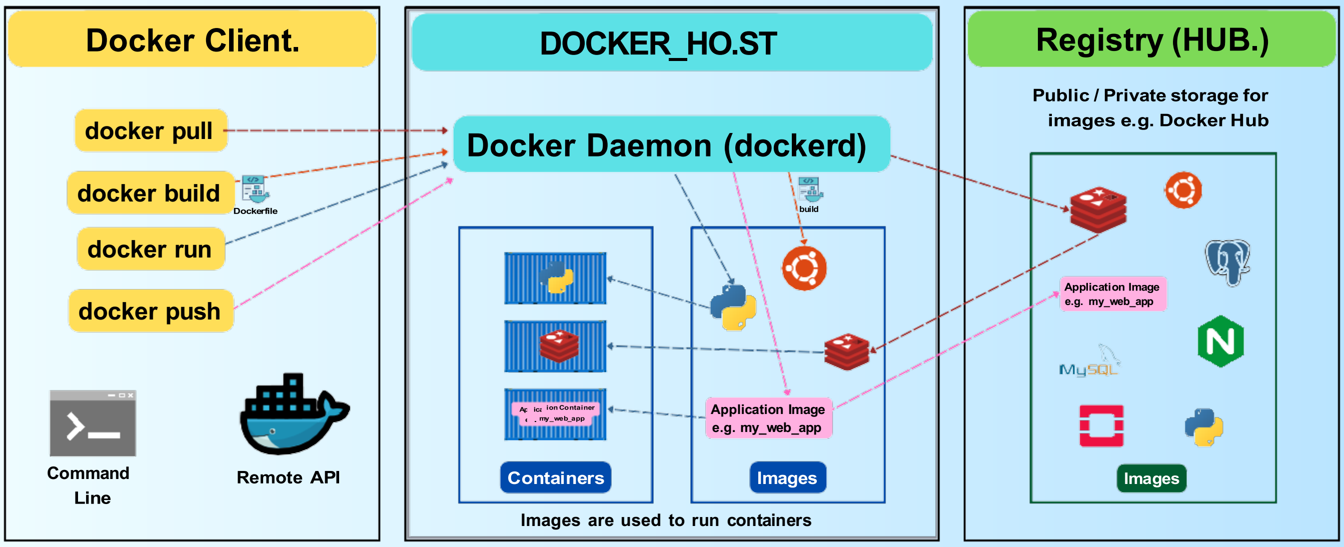
Docker CheatSheet

Docker Architecture


Docker Client.
docker pull
docker build
Dockerfile
docker run
docker push
Command Line
Remote API
DOCKER_HO.ST
Docker Daemon (dockerd)
build
Application Container e.g. my_web_app
Application Image e.g. my_web_app
Containers
Images
Images are used to run containers
Registry (HUB.)
Public / Private storage for images e.g. Docker Hub
Application Image e.g. my_web_app
MySQL
Images

⇢ **docker pull -** Pulls an image from a registry to the Docker Host.
⇢ **docker build -** Builds an image from a Dockerfile.
⇢ **docker run -** Creates and starts a container from an image.
⇢ **docker push -** Pushes an image from the Docker Host to a registry.

Docker uses a client-server architecture, consisting of several main components that work together to build, run, and manage containers. Here's a breakdown of each part:

Docker Client

- The Docker Client is the primary way users interact with Docker. It sends commands to the Docker Daemon using CLI commands like docker run, docker build, docker pull, and docker push.
- The client communicates with the daemon through a REST API or command-line interface (CLI).

Docker Daemon (dockerd)

- The Docker Daemon runs on the Docker Host and manages all container-related tasks.
- It listens to API requests and handles actions like building, running, and distributing Docker containers.
- The daemon also manages Docker images and container lifecycle operations, ensuring efficient resource usage on the host machine.

Docker Host

- This is the machine (local or cloud-based) where Docker Daemon runs, managing containers and images.
- **Images:** The Docker Daemon uses images to create containers. These images are built from Dockerfiles and can be pulled from the Docker Registry.
- **Containers:** Containers are instances of images that run applications. Containers are lightweight and isolated from each other but share the same OS kernel.

Docker Registry (Hub)

- A Docker Registry is a repository where Docker images are stored and managed. Docker Hub is the default public registry, but private registries can also be set up.
- The registry allows users to push images to share with others or pull images for local use.
- Images are versioned and stored in the registry, serving as blueprints for creating containers on any Docker Host.

Installation and Setup

Install Docker:

- **Linux:** Follow distribution-specific instructions.
- **Windows/Mac:** Use Docker Desktop.

Post-Installation:

- **docker --version** – Check installation.
- **docker info** – Display Docker system information.

Configuration:

- **sudo usermod -aG docker $USER** - Add user to the Docker group.

Working with Images

Build, List, and Remove Images

- **docker build -t <image-name> .**
  # Build an image from Dockerfile
- **docker images**
  # List local images
- **docker rmi <image-id>**
  # Remove an image by ID
- **docker image prune -a**
  # Remove unused images

Pulling and Pushing Images

- **docker pull <image-name>**
  # Pull an image from a registry
- **docker push <image-name>**
  # Push an image to a registry

Container Interaction and Inspection

- **docker exec -it <container-id> /bin/bash**
  # Start an interactive bash session in a running container
- **docker attach <container-id>**
  # Attach to a running container's main process
- **docker logs <container-id>**
  # View container logs
- **docker stats <container-id>**
  # Display resource usage statistics for one or more containers
- **docker inspect <container-id>**
  # Display detailed configuration and state info about a container
- **docker top <container-id>**
  # Display running processes inside the container

Container Lifecycle Management

Starting, Stopping, and Managing Containers

- **docker run -d -p <host-port>:<container-port> <image>**
  # Start a container in detached mode with port mapping (e.g. 80:80)
- **docker stop <container-id>**
  # Gracefully stop a running container
- **docker start <container-id>**
  # Start a stopped container
- **docker restart <container-id>**
  # Restart a running or stopped container
- **docker kill <container-id>**
  # Forcefully stop (kill) a running container
- **docker rm <container-id>**
  # Remove a stopped container
- **docker rm -f <container-id>**
  # Force remove a running container
- **docker ps**
  # List running containers
- **docker ps -a**
  # List all containers (including stopped)
- **docker rename <container-id> <new-name>**
  # Rename a container

Common Run Options

- **docker run --name <name> -it <image>**
  # Assign a custom name and run in interactive mode
- **docker run -v <host-path>:<container-path> <image>**
  # Mount a volume from the host
- **docker run --env <env-var>=<value> <image>**
  # Set an environment variable
- **docker run --network <network-name> <image>**
  # Connect the container to a specified network
- **docker run --rm <image>**
  # Automatically remove the container when it stops

Advanced Dockerfile Directives

Key Dockerfile Instructions

FROM <image>	# Set base image
WORKDIR /app	# Set working directory
COPY . .	# Copy all files to container
RUN <command>	# Run commands in container
EXPOSE <port>	# Expose container port.

ENTRYPOINT ["executable", "param"] #Set container's main executable.

CMD ["executable", "param"] # Start container process

HEALTHCHECK --interval=30s --timeout=10s CMD curl -f http://localhost:<port> || exit 1 # Define container health check

Multi-Stage Build Example

```
# Stage 1 - Build
FROM node:14 AS builder
WORKDIR /app
COPY . .
RUN npm install && npm run build

# Stage 2 - Runtime
FROM  node:14-slim
WORKDIR /app
COPY --from=builder /app/dist /app/dist
CMD ["node", "dist/app.js"]
```

Optimization Best Practices

- **Layering:** Combine commands to reduce layers. Place stable commands at the top (e.g., apt-get update).
- **.dockerignore:** Exclude unnecessary files to reduce image size.
- Use **ARG** for build-time variables; **ENV** for runtime configuration.

Docker Compose Basic Commands

- **docker-compose up -d**
  # Start services
- **docker-compose down**
  # Stop and remove all services
- **docker-compose logs <service>**
  # View service logs
- **docker-compose up -d --scale <service>=3**
  #Scale services
- **docker-compose ps**
  # List running services

Data Persistence with Volumes

Creating and Managing Volumes

- **docker volume create my_volume**
  # Create volume
- **docker run -v my_volume:/data <image>**
  # Attach volume to container
- **docker volume inspect my_volume**
  # View volume details
- **docker volume rm my_volume**
  # Remove volume

Data Sharing

- **docker run -v shared_volume:/shared --name app1 busybox**
- **docker run -v shared_volume:/shared --name app2 busybox**

# Both app1 and app2 can access /shared, enabling data sharing.

docker-compose.yml Sample

```
version: '3.8'
services:
 web:
  image: nginx
  ports:
   - "8080:80"
  volumes:
   - web-data:/usr/share/nginx/html
  networks:
   - app-network
 db:
  image: mysql
  environment:
   MYSQL_ROOT_PASSWORD: rootpass
volumes:
 web-data:
networks:
 app-network:
```

# Health Checks can also be included in docker compose:

```
services:
 web:
  image: nginx
  healthcheck:
   test: ["CMD", "curl", "-f", "http://localhost"]
   interval: 30s
   timeout: 10s
   retries: 3
```

Docker Networking

Network Types

- **Bridge:** Default; internal communication between containers on the same host.
  docker network create -d bridge my_bridge_network
- **Host:** Shares the host's network directly (no isolation).
  docker run --network host nginx
- **Overlay:** Connects containers across Docker hosts in Swarm mode.
  docker network create -d overlay my_overlay_network
  docker service create --name web --network my_overlay_network nginx
- **Macvlan:** Assigns unique MAC addresses to containers, appearing as individual devices.

```
docker network create -d macvlan \
  --subnet=192.168.1.0/24 \
  --gateway=192.168.1.1 \
  -o parent=eth0 my_macvlan_network
docker run --network my_macvlan_network --ip
192.168.1.100 nginx
```

- **Ipvlan:** Like Macvlan but operates with a single MAC address for a network segment.

```
docker network create -d ipvlan \
  --subnet=192.168.2.0/24 \
  --gateway=192.168.2.1 \
  -o parent=eth0 my_ipvlan_network
docker run --network my_ipvlan_network --ip
192.168.2.100 nginx
```

- **None:** Completely isolated network mode with no connectivity.
  docker run --network none nginx

Common Network Commands

- **docker network ls**
  #List networks
- **docker network create my_network**
  #Create a network
- **docker network connect my_network <container>**
  #Connect container to network
- **docker network inspect <network-name>**
  # Inspect network

Security Essentials

- **docker scan <image-name>**
  # Scan an image for vulnerabilities.
- **docker run --user $(id -u):$(id -g) <image>**
  #Run containers as a non-root user.
- **docker run --memory="256m" --cpus="1" <image>**
  #Limit resource usage

Tips for Security

- Use minimal images (e.g., Alpine).
- Limit container privileges (--cap-drop).
- Regularly update images and avoid outdated versions.
- Environment Variables: Store sensitive information with docker secret in Swarm mode.

Orchestration with Docker Swarm

- **docker swarm init**
  # Initialize a swarm
- **docker node ls**
  # List nodes in the swarm
- **docker service create --name <service> --replicas 3 <image>**
  # Create replicated service
- **docker service ls**
  # List services
- **docker service scale <service>=5**
  # Scale service to 5 instances
- **docker service update --image <new-image> <service>**
  # Update service image

CI/CD Integration with Docker

Push Image to Docker Hub:

- **docker login**
- **docker tag <image> <username>/<repository>:<tag>**
- **docker push <username>/<repository>:<tag>**

Docker in Continuous Integration Pipelines

- **Automated Builds:** Build Docker images for each code commit to ensure compatibility.
- **Testing in Containers:** Run tests within containers for consistent environments.
- **Simulate Production:** Use Docker Compose to mirror production environments.
- **Push Tested Images:** Send images to a registry to simplify downstream deployments.

Docker in Continuous Deployment Pipelines

- **Automated Deployment:** Use orchestration tools (e.g., Swarm, Kubernetes) for production rollouts.
- **Versioning and Rollback:** Use tags to version images and enable rollbacks.

CI/CD Commands Example:

- **docker build -t myapp:$GIT_COMMIT .**
  # Build and tag image for each commit
- **docker push myapp:$GIT_COMMIT**
  # Push image to Docker Hub
- **docker pull myapp:$GIT_COMMIT**
  # Pull image from registry for deployment
- **docker-compose -f docker-compose.prod.yml up -d**
  # Deploy using Docker Compose

Docker System Maintenance & Clean-Up Commands

- **docker system df**
  #Check disk usage of Docker resources
- **docker events**
  #Monitor real-time events
- **docker system prune**
  #Remove unused containers, networks, images, and build cache
- **docker system prune -a**
  #Force remove all stopped containers, networks, and unused images
- **docker volume prune**
  #Remove unused volumes
- **docker image prune**
  #Remove dangling images
- **docker network prune**
  #Remove unused networks
- **docker image prune --filter "until=24h"**
  #Remove unused images based on filters
- **docker builder prune**
  #Clear Docker build cache

Additional Commands

- **docker commit <container> <new_image>**
  #Create a new image from a container's changes.
- **docker cp <container>:<path> <local_path>**
  #Copy files from a container to the host system.
- **docker diff <container>**
  #Show changes made to a container's filesystem.
- **docker export <container> > <file>.tar**
  #Export a container's filesystem as a tar archive.
- **docker import <file>.tar**
  #Import a tar archive as a new image.
- **docker tag <image> <tag>**
  #Add a tag to an existing image.
- **docker save -o <file> <image>**
  #Save an image to a tar archive.
- **docker load -i <file>**
  #Load an image from a tar archive.
- **docker network disconnect <network> <container>**
  #Disconnect a container from a network.
- **docker logout**
  #Log out from a Docker registry.
- **docker-compose exec <service> <command>**
  #Run a command in a running service container.