Docker

Core Concepts

- **Docker:** A platform for building, shipping, and running applications using containerization. It simplifies software installation, distribution, and management.
- **Containers:** Isolated runtime environments that package an application and its dependencies. They are a lightweight alternative to virtual machines (VMs).
- **Images:** Read-only templates used to create containers. They contain the application, libraries, and dependencies.
- **Docker Hub:** A public registry for storing and sharing Docker images.
- **Daemon:** A background process or service that runs continuously.

Key Benefits

- **Isolation:** Containers isolate applications from each other and the host system, preventing conflicts.
- **Portability:** Docker images can run on any system with Docker installed.
- **Efficiency:** Containers consume fewer resources than VMs.
- **Organization:** Docker helps manage dependencies and keeps the system clean.
- **Security:** Containers limit the scope of potential security breaches.

Basic Docker Commands

- docker run <image>: Creates and starts a container from an image.
  - -d or --detach: Runs the container in the background (detached mode).
  - --name <container_name>: Assigns a name to the container.
  - -i or --interactive: Keeps STDIN open even if not attached.
  - -t or --tty: Allocates a pseudo-TTY.
  - --link <container_name>:<alias>: Creates a link to another container (legacy).
- docker ps: Lists running containers.
  - -a: Lists all containers (running and stopped).
- docker logs <container_name>: Displays the logs of a container.
  - -f or --follow: Follow log output in real-time.
- docker stop <container_name>: Stops a running container.
- docker restart <container_name>: Restarts a stopped container.
- docker exec <container_name> <command>: Executes a command inside a running container.
- docker create <image>: Creates a container but does not start it.
- docker rename <old_name> <new_name>: Renames a container.
- docker inspect <container_name>: Displays detailed information about a container.
- docker help <command>: Shows help information for a specific Docker command.

Important Concepts & Techniques

- **PID Namespace:** Isolates process IDs within a container.
- **Container Naming:** Use meaningful names or IDs for container management.
- **Container State:** Containers can be in various states (created, running, stopped, etc.).
- **Environment Independence:** Minimize dependencies on the host environment.
- **Read-only Filesystems:** Improves security and prevents unintended modifications. Use the --read-only flag.
- **Volumes:** Used to persist data and share files between the host and containers. -v <host_path>:<container_path>
- **tmpfs:** Creates temporary, in-memory filesystems. --tmpfs <container_path>
- **Environment Variables:** Inject configuration data into containers. -e <VARIABLE>=<value>

Troubleshooting

- **Port Conflicts:** Occur when multiple containers try to use the same port.
- **Dependency Order:** Ensure containers are started in the correct order based on dependencies.
- **Image Installation:** Verify that the image is installed correctly.

Workflow Summary

1. **Pull/Build Image:** Obtain a Docker image from a registry or build your own.
2. **Create Container:** Create a container from the image using docker run or docker create.
3. **Start Container:** Start the container using docker start.
4. **Manage Container:** Monitor, stop, restart, or execute commands within the container.

Networking

Key Concepts

- **Protocols:** Languages for network communication (e.g., HTTP).
- **Network Interface:** A point of connection to a network, having an IP address.
- **Port:** A specific channel on an interface for a process (e.g., port 80 for HTTP).
- **Bridge:** Connects multiple networks.
- Docker abstracts host networks from containers.
- Containers get unique IP addresses within a Docker network.
- Networks are first-class entities, managed with docker network commands.

Default Networks

- bridge: Default network, inter-container connectivity on a single host (Legacy, not recommended).
- host: No network isolation; container uses the host's network directly.
- none: No external network connectivity for the container.

Network Scope

- local: Confined to a single host.
- global: Created on all cluster nodes, but doesn't route between them.
- swarm: Spans all hosts in a Docker Swarm cluster.

User-Defined Bridge Networks

- Use docker network create to create custom bridge networks.
- Enable modern Docker features like service discovery and load balancing.

Creating a Network

docker network ls # List networks
docker network create <network_name> # Create a network

Bridge Networks (Single-Host)

- Default: docker0 (virtual interface).
- Containers get private IPs, routed via Docker network.
- User-defined bridges: docker network create --driver bridge ...
  - --attachable: Allows attaching/detaching containers.
  - --subnet, --ip-range: Custom IP addressing.
- docker network connect: Attaches container to a network.

Exploring Networks

- ip addr: Shows container network interfaces (lo, eth0, etc.).
- nmap: Scans network for connected devices.
- DNS-based service discovery: Containers discover each other by name (hostname = container name).

Beyond Bridge Networks

- **Underlay Networks (macvlan, ipvlan):** First-class network addresses, routable from host network. Linux hosts only.
- **Overlay Networks (Swarm Mode):** Multi-host aware bridge, inter-container connections across swarm nodes.

Special Networks

- --network host: Container shares host's network namespace (no isolation). Access to localhost services.
- --network none: Container is network-isolated (only loopback interface).

Inbound Traffic (NodePort Publishing)

- -p host_port:container_port: Forwards traffic from host to container.
- docker port container_name: Lookup port mappings.

Networking Caveats

- **No Firewalls:** Containers on same network have unrestricted access to each other. Application-level security is crucial.
- **Custom DNS:**
  - --hostname: Sets container hostname (resolves to container IP).
  - --dns: Specifies DNS servers for container.
  - --dns-search: Specifies DNS search domain (appends to hostnames).
  - --add-host: Adds custom hostname-to-IP mapping.

Image Packaging

Building from a Container

1. docker run: Create container from existing image.
2. Modify filesystem (install software, create files, etc.).
3. docker commit: Create new image from container's changes.

Reviewing Changes

- docker container diff container_name (A=added, C=changed, D=deleted).

Committing Images

- docker container commit -a "author" -m "message" container_name new_image_name
  - -a: Author.
  - -m: Commit message.
- Image attributes, such as environment variables, working directory, exposed ports, volumes, entrypoint, and command are all saved in the image.

Union Filesystems (UFS)

- Images are built in layers.
- Changes are written to new layers.
- Reading a file: UFS searches layers top-down.
- File deletion: Adds a "delete" record to the top layer.
- File changes: Adds the changed file to the top layer.

Image Distribution

Key Concepts

- **Distribution Spectrum:** Ranging from simple (hosted public registries) to complex (custom distribution).
- **Selection Criteria:**
  - **Cost:** Free to expensive.
  - **Visibility:** Public vs. private.
  - **Transport Speed/Bandwidth:** Installation speed.
  - **Longevity Control:** Risk of third-party changes.
  - **Availability Control:** Ability to resolve availability issues.
  - **Access Control:** Protection from unauthorized modification.
  - **Artifact Integrity:** Ensuring image authenticity.
  - **Artifact Confidentiality:** Protecting sensitive content.
  - **Requisite Expertise:** Skill level needed.

Distribution Methods

1. **Hosted Public Registries (e.g., Docker Hub):** Simple, high visibility. Limited control over longevity and availability.
2. **Hosted Private Registries:** Offer access control and confidentiality. Still rely on a third party for availability and longevity.
3. **Private Registries:** Run your own registry using registry software. Greater control over all aspects, but increased responsibility. Suitable for local, corporate, or private cloud networks.
4. **Custom Image Distribution:** Maximum flexibility. Requires building your own infrastructure (e.g., SFTP, HTTP downloads).
5. **Image Source Distribution:** Distribute Dockerfiles instead of images. Requires users to build images themselves. Suitable for open-source projects.

Services

Key Concepts

- **Service:** A discoverable and available process/functionality over a network. Abstraction simplifies management.
- **Task:** A unit of work within a service, often a container.
- **Swarm Mode:** Enables Docker service management features. Requires initialization (docker swarm init).
- **Orchestrator (e.g., Docker Swarm):** Automates service lifecycle, ensuring desired state is maintained. Tracks desired state vs. current state.
- **Replication:** Running multiple instances (replicas) of a service for availability.

Service Lifecycle & Management

1. **Service Creation:** docker service create defines a service (name, image, port mapping).
2. **Resurrection:** Swarm automatically restarts failed containers to maintain the desired number of replicas.
3. **Scaling:** docker service scale adjusts the number of service replicas.

4. **Inspection:** docker service inspect shows the service's desired state definition.
5. **Replication Modes:**

- **Replicated:** Maintains a fixed number of replicas.
- **Global:** Runs one replica on each node in the swarm.

1. **Automated Rollout:** docker service update updates a service's configuration (e.g., image).

- --update-order: stop-first (stop old before starting new).
- --update-parallelism: Number of replicas updated simultaneously.
- --update-delay: Time between updates.

Image Build Patterns

- **All-in-One:** Includes build tools and runtime dependencies in a single image (simple, large).
- **Build Plus Runtime:** Separate build image and a smaller runtime image (better security, smaller size).
- **Build Plus Multiple Runtimes:** Slim runtime image with variations for debugging or specialized use cases (multi-stage builds).

Image Metadata

- **Labels:** Use LABEL instruction to add metadata (application name, version, build date, VCS commit).
- **Dockerfile & Manifests:** Include these in the image filesystem for traceability.
- **Orchestration with Make:** Use make to automate the build process (metadata gathering, artifact building, image building, testing, tagging).

Image Testing

- **Dockerfile Linting:** Use tools like hadolint to check for best practices.
- **Container Structure Test (CST):** Verifies file permissions, command execution, metadata.
- **Vulnerability Scanning:** Scan images for known vulnerabilities before publishing.

Image Tagging

- Tags are mutable, human-readable pointers to image IDs.
- **Continuous Delivery with Unique Tags:** Use unique build IDs for each image and promote them through stages.
- **Configuration Image per Deployment Stage:** Create a generic app image and separate config images for each environment.
- **Semantic Versioning:** Use Major.Minor.Patch for releases to communicate the level of change.

Swarm

Core Concepts

- **Swarm:** Docker's built-in clustering & orchestration. Enables deploying apps across multiple hosts for scalability & availability.
- **Nodes:** Machines in a Swarm cluster.
  - **Manager:** Orchestrates services, maintains cluster state. Requires a majority to be available.
  - **Worker:** Executes tasks (containers) as instructed by managers.
- **Services:** Define application processes. Swarm creates tasks to realize service definitions.
- **Tasks:** Containerized process scheduled & run once. Replaced if they fail based on restart policy.
- **Networks:** Overlay networks for service communication. Encrypted for security.
- **Volumes:** Persistent storage for service data. Local to a node by default.
- **Configs/Secrets:** Externalized configuration data. Secrets are handled securely.

Deployment & Management

- **Swarm Initialization:** docker swarm init (on a manager node).
- **Node Joining:** docker swarm join (using join tokens).
- **Docker Stack:** A named collection of services, networks, and volumes. Defined using Docker Compose (YAML).
  - docker stack deploy: Creates/updates a stack.
  - docker stack ps: Lists tasks in a stack.
  - docker stack rm: Removes a stack.
- **Service Management:**
  - docker service update: Modifies service configuration.
  - docker service ps: Lists tasks for a service.
  - docker service logs: Shows logs for a service.
- **Config Management:**
  - docker config create: Creates a config.
  - docker config inspect: Inspects a config.
  - docker config rm: Removes a config.
- **Secret Management:**
  - docker secret create: Creates a secret.
  - docker secret inspect: Inspects a secret.
  - docker secret rm: Removes a secret.

Compose File (docker-compose.yml)

- version: Specifies Compose file version.
- services: Defines services.
  - image: Container image to use.
  - ports: Port mappings (host:container).
  - networks: Networks to attach the service to.
  - volumes: Volume mounts (host_path:container_path).
  - environment: Environment variables.
  - secrets: Secrets to mount.
  - configs: Configs to mount.

    - deploy: Deployment configuration.
        - replicas: Number of task replicas.
        - restart_policy: Defines restart behavior.
        - update_config: Configures update strategy.
        - resources: Resource limits (CPU, memory).
- networks: Defines networks.
    - driver: Network driver (e.g., overlay).
    - driver_opts: Driver options (e.g., encryption).
    - attachable: Allows standalone containers to attach to this network.
- volumes: Defines volumes.
- secrets: Defines external secrets.
    - external: true: Indicates secret is managed outside the Compose file.
- configs: Defines external configs.
    - external: true: Indicates config is managed outside the Compose file.

Key Considerations

- **Service Discovery:** Services discoverable by name within a Docker network.
- **Load Balancing:** Swarm provides built-in load balancing across service replicas.
- **Security:** Use encrypted networks. Secure published ports with TLS.
- **Secrets Management:** Avoid environment variables for secrets. Use Docker secrets mounted as files.
- **Immutable Configs:** Configs are immutable. Create new configs on change.
- **Restart Policies:** Define robust restart strategies.
- **Update Configuration:** Control update rollout (parallelism, delay).
- **Resource Limits:** Set CPU and memory limits for services.
- **Dependencies:** Use depends_on to define service startup order.
- **Rolling Updates:** Swarm updates services in batches, stopping old tasks before starting new ones.

Core Concepts (Redux)

- **Nodes:** Machines in the Swarm cluster (managers & workers). docker node ls shows node status.
- **Services:** Application definitions (e.g., API, database). Defined in docker-compose.yml.
- **Tasks:** Individual container instances running as part of a service.
- **Stacks:** Collection of related services defined in a docker-compose.yml file. Deployed with docker stack deploy.
- **Secrets:** Secure way to manage sensitive data (passwords, keys). Created with docker secret create.
- **Configs:** Non-sensitive configuration data.
- **Networks:** Overlay networks enable communication between services.

Key Commands (Redux)

- docker node ls: List nodes in the Swarm.
- docker stack deploy --compose-file docker-compose.yml <stack_name>: Deploy a stack.
- docker service ls: List services.

- docker service logs <service_name>: View service logs.
- docker service ps <service_name>: List tasks for a service.
- docker secret create <secret_name> <file>: Create a secret.
- docker stack rm <stack_name>: Remove a stack.

Deployment Process (Redux)

1. Initialize a Swarm (on a manager node).
2. Define services in a docker-compose.yml file, specifying images, ports, networks, secrets, and configs.
3. Create secrets (if needed).
4. Deploy the stack using docker stack deploy.
5. Verify service status with docker service ls and docker service ps.
6. Access the application through published ports on any node in the Swarm.

Networking (Redux)

- **Ingress Network:** Special overlay network for external access to services.
- **Routing Mesh:** Exposes service ports on all nodes, routing traffic to available tasks.
- **Service Discovery:** Services use DNS to find each other by name.
- **Overlay Networks:** Isolate communication between services.

Task Placement (Redux)

- **Replicated Mode:** Swarm maintains the desired number of replicas.
- **Global Mode:** One task per node.
- **Placement Constraints:** Control where tasks run using node attributes (e.g., node.role == worker, node.labels.zone == public). Use docker service update --constraint-add.

Resource Management (Redux)

- **Limits:** Maximum resources a container can use (e.g., CPU, memory).
- **Reservations:** Guaranteed minimum resources.
- Use deploy.resources.limits and deploy.resources.reservations in docker-compose.yml.

Important Notes (Redux)

- Swarm requires all cluster-level resources (networks, secrets, configs) to exist before deploying services.
- Service tasks are ephemeral; updates replace containers.
- Data in default volumes is local to a node. Use volume plugins for shared storage.
- No firewalls between containers on a Docker overlay network.
- Application-level authentication is crucial for service-to-service communication.
- Swarm load balances connections, not HTTP requests, by default.