`docker create`

The `docker create` command is used to create a new container without starting it. Here's the basic syntax and some common options:

Basic Syntax

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Common Options

- `--name` : Assign a name to the container.
- `-p` : Map a container port to a host port (`hostPort:containerPort`).
- `-e` : Set environment variables.
- `-v` : Bind mount a volume.
- `--network` : Connect the container to a specified network.
- `--restart` : Define the restart policy (e.g., `always` , `on-failure` , `no`).
- `--memory` : Set a memory limit (e.g., `512m` , `1g`).
- `--cpus` : Limit the number of CPUs (e.g., `0.5` , `2`).

Examples

1. **Create a container from an image:**

```
docker create --name my_container ubuntu
```

2. **Create a container with port mapping:**

```
docker create -p 8080:80 nginx
```

3. **Create a container with an environment variable:**

```
docker create -e MY_ENV_VAR=value alpine
```

4. **Create a container with a volume mount:**

```
docker create -v /host/path:/container/path busybox
```

5. **Set a restart policy:**

```
docker create --restart always redis
```

To start the container after creation, use:

```
docker start <container_name_or_id>
```

`docker network`

The `docker network` command is used to manage networks in Docker. Here's an overview of the basic syntax and commonly used subcommands:

Basic Syntax

```
docker network [COMMAND] [OPTIONS]
```

Commonly Used Subcommands

1. **`docker network create`** : Creates a new network.
2. **`docker network ls`** : Lists all networks.
3. **`docker network inspect`** : Shows details about a network.
4. **`docker network connect`** : Connects a container to a network.
5. **`docker network disconnect`** : Disconnects a container from a network.

6. **`docker network rm`**: Removes a network.

Examples

1. **List all networks:**

```
docker network ls
```

2. **Create a new bridge network:**

```
docker network create my_bridge_network
```

3. **Create an overlay network (useful for Docker Swarm):**

```
docker network create --driver overlay my_overlay_network
```

4. **Inspect a network:**

```
docker network inspect my_bridge_network
```

5. **Connect a container to a network:**

```
docker network connect my_bridge_network my_container
```

6. **Disconnect a container from a network:**

```
docker network disconnect my_bridge_network my_container
```

7. **Remove a network:**

```
docker network rm my_bridge_network
```

Common Network Drivers

- **`bridge`**: Default network driver for standalone containers.
- **`host`**: Uses the host's network stack (no network isolation).
- **`overlay`**: Used for multi-host networking with Docker Swarm.
- **`macvlan`**: Allows containers to appear as physical devices on the network.
- **`none`**: Disables all networking for a container.

You can specify the driver when creating a network:

```
docker network create --driver bridge my_custom_network
```

`docker run`

The `docker run` command is used to create and start a new container from a specified image. Here's the basic syntax and some commonly used options:

Basic Syntax

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Common Options

- `--name`: Assign a name to the container.
- `-d`: Run the container in detached mode (background).
- `-p`: Map a host port to a container port (`hostPort:containerPort`).
- `-e`: Set environment variables.
- `-v`: Bind mount a volume.
- `--network`: Connect the container to a specified network.
- `--restart`: Set the container restart policy (`no`, `on-failure`, `always`, `unless-stopped`).
- `--rm`: Automatically remove the container when it exits.
- `-it`: Run the container in interactive mode with a TTY (useful for debugging).
- `--memory`: Set a memory limit for the container.
- `--cpus`: Limit the number of CPUs available to the container.

Examples

1. **Run a basic container:**

   ```
   docker run ubuntu
   ```

2. **Run a container in detached mode:**

   ```
   docker run -d nginx
   ```

3. **Run a container with a specific name:**

   ```
   docker run --name my_container nginx
   ```

4. **Run a container with port mapping:**

   ```
   docker run -p 8080:80 nginx
   ```

5. **Run a container with an environment variable:**

   ```
   docker run -e MY_ENV_VAR=value alpine
   ```

6. **Run a container with a volume mount:**

   ```
   docker run -v /host/path:/container/path busybox
   ```

7. **Run a container in interactive mode with a TTY:**

   ```
   docker run -it ubuntu /bin/bash
   ```

8. **Run a container with a restart policy:**

   ```
   docker run --restart always redis
   ```

9. **Automatically remove the container when it exits:**

   ```
   docker run --rm ubuntu
   ```

10. **Limit memory and CPU usage:**

    ```
    docker run --memory 512m --cpus 1.0 nginx
    ```

The `docker run` command combines creating a container (`docker create`) and starting it (`docker start`).

`docker exec`

The `docker exec` command allows you to run a command inside an already running Docker container. Here's the basic syntax and commonly used options:

Basic Syntax

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Common Options

- `-d` : Run the command in the background.
- `-i` : Keep STDIN open even if not attached.
- `-t` : Allocate a pseudo-TTY (useful for interactive commands).
- `-u` : Run the command as a specific user (`username` or `uid`).

Examples

1. **Run a command inside a container:**

   ```
   docker exec my_container ls /app
   ```

2. **Run a command with interactive terminal (`-it`):**

   ```
   docker exec -it my_container /bin/bash
   ```

   This opens an interactive Bash shell inside the container.

3. **Run a command as a specific user:**

   ```
   docker exec -u username my_container whoami
   ```

4. **Run a command in the background:**

   ```
   docker exec -d my_container touch /tmp/newfile
   ```

5. **Run a command while keeping STDIN open:**

   ```
   docker exec -i my_container cat
   ```

The `docker exec` command is particularly useful for debugging, testing, or executing maintenance tasks inside running containers.

To remove various Docker objects, you can use the following commands for containers, images, networks, volumes, and other related items. Here's a comprehensive guide to Docker removal commands, including options for bulk deletion.

1. Remove Containers

- **Remove a specific container:**

  ```
  docker rm <container_id_or_name>
  ```

- **Remove all stopped containers:**

  ```
  docker container prune
  ```

- **Remove multiple containers by specifying their IDs or names:**

  ```
  docker rm <container_id1> <container_id2> <container_id3>
  ```

- **Force-remove a running container (stops and removes it):**

  ```
  docker rm -f <container_id_or_name>
  ```

- **Remove all containers (both running and stopped):**

  ```
  docker rm -f $(docker ps -aq)
  ```

2. Remove Images

- **Remove a specific image:**

  ```
  docker rmi <image_id_or_name>
  ```

- **Remove multiple images by specifying their IDs or names:**

  ```
  docker rmi <image_id1> <image_id2> <image_id3>
  ```

- **Remove all dangling (untagged) images:**

  ```
  docker image prune
  ```

- **Remove all images, including those in use by containers:**

  ```
  docker rmi -f $(docker images -q)
  ```

3. Remove Networks

- **Remove a specific network:**

```
docker network rm <network_name_or_id>
```

- **Remove all unused networks:**

```
docker network prune
```

- **Remove multiple networks by specifying their names or IDs:**

```
docker network rm <network_id1> <network_id2> <network_id3>
```

4. Remove Volumes

- **Remove a specific volume:**

```
docker volume rm <volume_name_or_id>
```

- **Remove all unused volumes:**

```
docker volume prune
```

- **Remove multiple volumes by specifying their names or IDs:**

```
docker volume rm <volume_id1> <volume_id2> <volume_id3>
```

5. Remove Docker System Objects

- **Remove all unused data (containers, networks, images, and volumes):**

```
docker system prune
```

- **Remove all unused data including stopped containers, networks, and images (forceful cleanup):**

```
docker system prune -a
```

- **Clean up Docker system completely (including unused volumes):**

```
docker system prune -a --volumes
```

Bulk Removal Tips

- To **remove everything** (containers, images, networks, volumes), you can use a combination of the above commands in a sequence:

```
docker rm -f $(docker ps -aq)           # Remove all containers
docker rmi -f $(docker images -q)       # Remove all images
docker network rm $(docker network ls -q)  # Remove all networks
docker volume rm $(docker volume ls -q)    # Remove all volumes
```

Use these commands with caution, especially with `-f` (force) and bulk removal commands, as they can delete important data and configurations.