Huawei HCCDA - Cloud Native Lab 1: Complete Container Guide 🐳

1. What Are Containers? (The Simple Explanation)

Think of containers like lunch boxes:

- Each lunch box contains a complete meal (your app + all its dependencies)
- They're portable - work the same on your laptop, your friend's computer, or in the cloud
- They're isolated - if one lunch box spills, it doesn't affect the others
- They're lightweight - unlike virtual machines that need a whole operating system each

Why we love containers:

- "It works on my machine" problem solved! 🎉
- Easy to share and deploy applications
- Consistent environment from development to production

---

2. Getting Started: Running Your First Container

Starting a Web Server Container

```
docker run -d -p 80:80 httpd
```

Let's break this down like a recipe:

- `docker run` = "Hey Docker, please make me a new container and start it!"
- `-d` = "Run it in detached mode" (like starting a background music player)
- `-p 80:80` = "Connect port 80 on my computer to port 80 in the container"
  - First `80` = your computer's door number
  - Second `80` = container's door number
- `httpd` = The Apache web server software package

What actually happens behind the scenes:

1. Docker checks if you have the `httpd` image locally
2. If not, it downloads it from Docker Hub (like an app store for containers)
3. Creates a new container from that image
4. Starts the Apache web server inside the container
5. Maps port 80 so you can access it via web browser

Try it! Open your browser and go to `http://localhost` - you should see "It works!"

Useful Inspection Commands

```
# See what containers are running
docker container ls
```

Output shows:

- CONTAINER ID: Unique ID for each container (like a serial number)
- IMAGE: What software it's running
- STATUS: Is it running or stopped?
- PORTS: How ports are mapped

```
# See what software packages (images) you have downloaded
docker image ls
```

---

3. Container Management (The Remote Control)

Basic Lifecycle Commands

```
# Stop a container (graceful shutdown)
docker stop container-id

# Start a stopped container
docker start container-id

# Remove a container permanently
docker rm container-id
```

Pro Tip: You can use the first few characters of the container ID instead of the full ID:

```
docker stop a1b2  # instead of docker stop
a1b2c3d4e5f6
```

Why Stop vs Remove?

- **Stop** = Like putting your computer to sleep (you can wake it up later)
- **Remove** = Like throwing your computer in the trash (it's gone forever!)

---

4. Exploring Inside Containers (The Adventure Begins!)

Downloading Different Operating Systems

```
# Download CentOS Linux (a specific version)
docker pull centos:centos7

# Download Ubuntu Linux (latest version)
docker pull ubuntu

# Download a specific Ubuntu version
docker pull ubuntu:24.04
```

Think of it like: Downloading different "flavors" of Linux to experiment with!

Going Inside a Container

```
# Start a CentOS container in background
docker run -t -d centos:centos7

# Get inside the running container
docker exec -it container-id bash
```

What do these flags mean?

- `-i` = "Keep the input open" (so you can type commands)
- `-t` = "Give me a virtual terminal" (so you see the output nicely)
- `bash` = "Start the bash shell" (the command interpreter)

Once inside, you can:

- Run commands like `ls`, `pwd`, `whoami`
- Install software with `yum install vim` (on CentOS)
- Create files and folders
- Type `exit` to leave when done

Important: Changes you make inside disappear when the container is deleted! (Unless you save them)

---

5. Saving Your Changes (Creating Custom Images)

The Manual Way: Docker Commit

```
# After making changes inside a container (like installing vim)
docker commit container-id centos-vim
```

What this does:

- Takes a "snapshot" of your current container
- Creates a new image called `centos-vim`
- This new image will always have vim installed when you create containers from it

Analogy: Like taking a photo of your perfectly configured computer to make identical copies later.

The Better Way: Dockerfile (Recipe Files)

```
docker build -t httpd-centos -f dockerfile /root
```

Breakdown:

- `docker build` = "Build an image following a recipe"
- `-t httpd-centos` = "Tag/name the resulting image as httpd-centos"
- `-f dockerfile` = "Use this file as the recipe"
- `/root` = "Build context" (files available during build)

Sample Dockerfile content:

```
FROM centos:centos7
RUN yum install -y httpd
COPY my-website/ /var/www/html/
CMD ["httpd", "-D", "FOREGROUND"]
```

Why Dockerfile is better than commit:

- Repeatable and consistent
- Version controllable (you can track changes)
- Easy to share and automate

---

6. Your Private Container Registry (Company App Store)

Setting Up Your Registry

```
docker run -d -p 5000:5000 registry
```

What this does:

- Starts a container that acts as your private image storage server
- Port 5000 is the standard port for Docker registries
- Now you have your own “private app store” for company images

Storing Your Custom Images

```
# Step 1: Tag your image for the registry
docker tag httpd-centos localhost:5000/http:V1

# Step 2: Push to the registry
docker push localhost:5000/http:V1
```

Tagging explained:

- `httpd-centos` = Your local image name
- `localhost:5000` = Address of your registry
- `http` = Repository name in the registry
- `V1` = Version tag (like “version 1”)

Using Images from Your Registry

```
# Remove local copy
docker rmi localhost:5000/http:V1

# Download from your registry
docker pull localhost:5000/http:V1
```

Real-world use: Companies use this to store their custom applications securely, rather than putting them on public Docker Hub.

---

7. Resource Management (Preventing Greedy Containers)

Memory Limits

```
# Limit to 200MB RAM
docker run -it -m 200M polinux/stress stress --vm 1
--vm-bytes 150M
```

What happens:

- Container can use maximum 200MB memory
- The stress tool tries to use 150MB
- If it tries to use more than 200MB, Docker kills the container

Memory + Swap Limits

```
docker run -it -m 300M --memory-swap=400M
polinux/stress stress --vm 2 --vm-bytes 100M
```

Memory math:

- RAM limit: 300MB
- Total memory (RAM + Swap): 400MB
- Therefore, Swap = 400MB - 300MB = 100MB

CPU Limits

```
# Use only 60% of one CPU core
docker run -it --cpus=0.6 polinux/stress stress --
vm 1

# Relative CPU sharing (when multiple containers
compete)
docker run -itd --cpu-shares 2048 polinux/stress
stress --cpu 1
```

CPU shares explained:

- Default is 1024 shares per container
- Container with 2048 shares gets twice as much CPU time as one with 1024
- Only matters when CPU is overloaded

---

8. Container Networking

Network Modes

```
# Use host's network directly (fastest, less
secure)
docker run -itd --network=host centos

# Use default bridge (isolated, more secure)
docker run -itd --network=bridge centos
```

Bridge network explained:

- Docker creates a virtual network switch

- Each container gets its own IP address on this virtual network
- Containers can talk to each other but are isolated from the host

Creating Custom Networks

```
docker network create --driver bridge --subnet 173.18.0.0/16 --gateway 173.18.0.1 servicebridge01
```

Why create custom networks?

- Group related containers together
- Better organization and security
- Automatic DNS resolution between containers on same network

---

9. Storage and Data Persistence

Volume Mounts (Sharing Folders)

```
docker run -d -p 80:80 -v /home/container/htdocs/:/usr/local/apache2/htdocs/ httpd
```

What this does:

- Connects `/home/container/htdocs/` on your computer to `/usr/local/apache2/htdocs/` in the container
- Changes in either location are instantly visible in both
- Your data survives even if the container is deleted

Use cases:

- Website files that you edit frequently
- Database files you want to backup
- Configuration files

---

10. Monitoring and Logs

Real-time Monitoring

```
docker stats
```

Shows live CPU, memory, and network usage for all containers - like Task Manager for Docker!

Viewing Logs

```
journalctl -b CONTAINER_NAME=container-name
```

Alternative methods:

```
# View logs for a specific container
docker logs container-id

# Follow logs in real-time
docker logs -f container-id
```

Key Technology Concepts

Cgroups (Control Groups)

- **What:** Linux feature that limits resource usage
- **Why:** Prevents one container from hogging all CPU/memory
- **Where:** `/sys/fs/cgroup/cpu/docker/container-id/`

Namespaces

- **What:** Linux feature that provides isolation
- **Types:** PID, Network, Mount, IPC, UTS, User namespaces
- **Result:** Each container thinks it's alone on the system

Common Pitfalls & Best Practices

Don't Forget:

1. **Data in containers is temporary** - use volumes for important data
2. **Stop containers before removing them**
3. **Use descriptive names and tags for images**
4. **Set resource limits in production**

Useful Aliases for Your Shell:

```
# Add to your ~/.bashrc file
alias dps='docker ps --format "table
{{.Names}}\t{{.Image}}\t{{.Status}}\t{{.Ports}}"'
alias dimg='docker images'
alias dstop='docker stop $(docker ps -aq)'
alias drm='docker rm $(docker ps -aq)'
```

Quick Reference Commands:

```
# See running containers
docker ps

# See all containers (including stopped)
docker ps -a

# Remove all stopped containers
docker container prune

# Remove unused images
docker image prune

# See disk usage
docker system df
```

Real-World Scenarios

Development Environment:

```
# Perfect setup for web development
docker run -d -p 80:80 -v $(pwd):/var/www/html
httpd
```

Database with Persistent Storage:

```
# MySQL with data that survives container restarts
docker run -d -p 3306:3306 -v
/data/mysql:/var/lib/mysql mysql
```

Multi-Container Application:

```
# Web app + database talking to each other
docker run -d --name database mysql
docker run -d --name website -p 80:80 --link
database httpd
```

Remember: This is just the beginning! Containers are the foundation for Kubernetes and modern cloud applications. Practice these commands until they become second nature! 🚀