

How to Use Vultr's Docker Marketplace Application

Install Vultr's Docker Marketplace app and learn secure, scalable container management best practices.

Contents

01	Introduction	3
02	Deploy Vultr's Docker Marketplace Application	3
03	Initial Setup and Configuration	4
04	Manage Containers	7
05	Manage Images	8
06	Work with Volumes and Networks	9
07	Install and Use Docker Compose	10
08	Best Practices and Configuration	13
09	Troubleshooting	14
010	Use Cases	16
011	Conclusion	17

Introduction

Docker is a platform for developing, shipping, and running applications in isolated containers. Containers package an application with its dependencies, ensuring consistent behavior across different environments. Docker simplifies deployment, scaling, and management of applications through lightweight, portable containers. The Vultr Docker Marketplace Application deploys Docker Community Edition (CE) with a pre-configured environment, enabling you to begin deploying containers immediately.

This guide explains deploying and using [Vultr's Docker Marketplace Application](#). You will deploy an instance, verify the installation, configure security, manage containers and images, work with volumes and networks, install Docker Compose for multi-container applications, and implement best practices for production Docker deployments.

Deploy Vultr's Docker Marketplace Application

1. Log in to your [Vultr Customer Portal](#) and click the **Deploy Server** button.
2. Select your preferred server type.
3. Choose a server location.
4. Select a server plan based on your workload:
 - **1GB RAM, 1 CPU**: Testing and small containers
 - **2GB RAM, 2 CPU**: 2-5 light containers
 - **4GB RAM, 2 CPU**: Multiple containers or resource-intensive apps
 - **8GB RAM, 4 CPU**: Production workloads with multiple services
5. Click the **Configure** button to proceed.
6. Under **Marketplace Apps**, search for `Docker` and select it as the Marketplace Application.

7. Select the **Limited Login** option from the **Additional Features** section to create a limited user with sudo access.
8. Review your configurations and click the **Deploy Now** button to start deployment.

 **Note**

It may take up to 10 minutes for your server to finish installing Docker and dependencies.

9. After the instance shows the status of Running, navigate to the **Server Overview** page and copy the SSH connection details.

Initial Setup and Configuration

After deployment, verify the installation and configure security before deploying containers.

Verify Docker Installation

1. Connect to your Vultr server instance over [SSH](#) using the connection details from the Server Overview page.
2. Check the Docker service status.

CONSOLE

```
$ sudo systemctl status docker
```

The service should show as `active (running)`.

3. Verify the Docker version.

CONSOLE

```
$ docker --version
```

Output:

```
Docker version 29.0.0, build 3d4129b
```

Configure Docker User Permissions

To run Docker commands without `sudo`, your user must be in the `docker` group.

1. Add your current user to the docker group.

CONSOLE

```
$ sudo usermod -aG docker $USER
```

2. Log out and log back in for the group membership to take effect.

CONSOLE

```
$ exit
```

Then SSH back into the server.

3. Verify group membership.

CONSOLE

```
$ groups
```

You should see `docker` in the list.

4. Test Docker access without sudo.

CONSOLE

```
$ docker run hello-world
```

 Note

The Vultr Docker Marketplace also includes a preconfigured `docker` user for running containers. You can switch to it with `sudo su - docker` if needed.

Configure Firewall Security

Secure your server by configuring the firewall to allow only necessary traffic.

1. Allow SSH connections.

CONSOLE

```
$ sudo ufw allow OpenSSH
```

2. Allow HTTP and HTTPS traffic (if running web containers).

CONSOLE

```
$ sudo ufw allow 80/tcp  
$ sudo ufw allow 443/tcp
```

3. Enable the firewall.

CONSOLE

```
$ sudo ufw enable
```

4. Verify firewall status.

CONSOLE

```
$ sudo ufw status
```

Warning

Only expose ports required by your containers. Each exposed port is a potential security risk.

Manage Containers

Learn essential operations for running, monitoring, and controlling Docker containers.

Run Containers

- Run a simple detached container.

CONSOLE

```
$ docker run -d --name nginx-test nginx
```

- Run a container with port forwarding.

CONSOLE

```
$ docker run -d -p 8080:80 --name web nginx
```

Access the container at `http://YOUR_SERVER_IP:8080`.

- Run a container with environment variables.

CONSOLE

```
$ docker run -d -e MYSQL_ROOT_PASSWORD=mypassword --name mysql mysql:8.0
```

- Run a container with a volume mount.

CONSOLE

```
$ docker run -d -v /data:/var/lib/mysql --name mysql-data mysql:8.0
```

Monitor and Manage Containers

- List running and stopped containers.

CONSOLE

```
$ docker ps          # Running containers
$ docker ps -a      # All containers
```

- View container logs.

CONSOLE

```
$ docker logs web
$ docker logs -f web # Follow logs in real-time
```

- Execute commands inside a container.

CONSOLE

```
$ docker exec -it web /bin/bash
```

Type `exit` to leave the container shell.

- Stop and remove containers.

CONSOLE

```
$ docker stop web      # Stop a running container
$ docker start web     # Start a stopped container
$ docker restart web   # Restart a container
$ docker rm web        # Remove a stopped container
$ docker rm -f web     # Force remove a running container
```

Manage Images

- Pull and list images.

CONSOLE

```
$ docker pull nginx:latest # Pull from Docker Hub
$ docker images           # List downloaded images
```

- Build a custom image (optional).

CONSOLE

```
$ mkdir ~/my-app && cd ~/my-app
$ nano Dockerfile
```

DOCKERFILE

```
FROM nginx:latest
COPY index.html /usr/share/nginx/html/
```

Create an HTML file and build:

CONSOLE

```
$ echo "<h1>Hello from Docker!</h1>" > index.html
$ docker build -t my-nginx:v1 .
```

- Clean up unused resources.

CONSOLE

```
$ docker system prune -a # Remove unused images and
containers
```

Work with Volumes and Networks

- Create and use volumes for persistent data.

CONSOLE

```
$ docker volume create mydata
$ docker run -d -v mydata:/app/data nginx
$ docker volume ls           # List volumes
```

- Create networks for container communication.

CONSOLE

```
$ docker network create mynetwork
$ docker run -d --name web --network mynetwork nginx
$ docker run -d --name app --network mynetwork alpine
```

Containers on the same network can communicate using container names as hostnames.

Install and Use Docker Compose

Docker Compose simplifies managing multi-container applications using declarative YAML configuration files.

Install Docker Compose

1. Download Docker Compose.

CONSOLE

```
$ sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Make the binary executable.

CONSOLE

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

3. Verify the installation.

CONSOLE

```
$ docker-compose --version
```

Output:

```
Docker Compose version v2.40.3
```

Deploy a Multi-Container Stack

1. Create a project directory.

CONSOLE

```
$ mkdir ~/wordpress && cd ~/wordpress
```

2. Create a `docker-compose.yml` file.

CONSOLE

```
$ nano docker-compose.yml
```

YAML

```
version: '3.8'

services:
  db:
    image: mysql:8.0
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wpuser
      MYSQL_PASSWORD: wppass
    restart: always

  wordpress:
    image: wordpress:latest
```

```
depends_on:
  - db
ports:
  - "8080:80"
environment:
  WORDPRESS_DB_HOST: db:3306
  WORDPRESS_DB_USER: wuser
  WORDPRESS_DB_PASSWORD: wppass
  WORDPRESS_DB_NAME: wordpress
restart: always

volumes:
  db_data:
```

Save and close the file.

3. Start the stack.

CONSOLE

```
$ docker-compose up -d
```

4. View running services.

CONSOLE

```
$ docker-compose ps
```

5. View logs.

CONSOLE

```
$ docker-compose logs -f
```

6. Stop the stack.

CONSOLE

```
$ docker-compose down
```

7. Stop and remove volumes.

CONSOLE

```
$ docker-compose down -v
```

Best Practices and Configuration

Implement these recommendations to ensure your Docker environment runs securely and efficiently.

Security Hardening

1. Run containers securely with resource limits.

CONSOLE

```
$ docker run -d --user 1000:1000 --memory="512m" --  
cpus="0.5" nginx  
$ docker run -d --read-only nginx # Read-only  
filesystem  
$ docker scan nginx:latest # Scan for  
vulnerabilities
```

2. Keep system and images updated, remove unused resources.

CONSOLE

```
$ sudo apt update && sudo apt upgrade -y  
$ docker pull nginx:latest  
$ docker system prune -a
```

Performance and Maintenance

1. Configure Docker daemon logging to prevent disk space issues.

CONSOLE

```
$ sudo nano /etc/docker/daemon.json
```

JSON

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

Save and close the file. Then restart Docker:

CONSOLE

```
$ sudo systemctl restart docker
```

2. Monitor disk usage.

CONSOLE

```
$ docker system df
```

3. Backup volumes and images.

CONSOLE

```
$ docker run --rm -v mydata:/data -v $(pwd):/backup alpine
tar czf /backup/backup.tar.gz /data
$ docker save -o nginx-backup.tar nginx:latest # Export
image
$ docker load -i nginx-backup.tar # Import
image
```

Troubleshooting

This section covers common issues and diagnostic commands.

Check Service Status and Logs

- Verify Docker service and view logs.

CONSOLE

```
$ sudo systemctl status docker
$ sudo journalctl -u docker -e
$ docker logs container-name
```

Common Issues

Permission Denied While Connecting to Docker Socket

This is the most common issue. Your user needs to be in the `docker` group.

- Add user to docker group and re-login.

CONSOLE

```
$ sudo usermod -aG docker $USER
$ exit # Log out
```

SSH back in and verify.

CONSOLE

```
$ groups # Should show 'docker'
$ docker run hello-world # Test without sudo
```

Cannot Connect to Docker Daemon

- Ensure the Docker service is running.

CONSOLE

```
$ sudo systemctl start docker
$ sudo systemctl status docker
```

Container Exits Immediately

- Check logs and exit code.

CONSOLE

```
$ docker logs container-name
$ docker inspect container-name --
format='{{.State.ExitCode}}'
```

Port Already in Use

- Find conflicting process and use a different port.

CONSOLE

```
$ sudo netstat -tulpn | grep :80
$ docker run -d -p 8080:80 nginx # Use alternate port
```

Out of Disk Space

- Check usage and clean up.

CONSOLE

```
$ docker system df
$ docker system prune -a --volumes
```

Use Cases

The Vultr Docker Marketplace application is ideal for various container-based workloads:

- **Microservices Architecture:** Deploy and manage distributed microservices with isolated dependencies and independent scaling.
- **Web Applications:** Run web servers, application stacks, and databases in containers for easy deployment and portability.

- **CI/CD Pipelines:** Execute reproducible builds, automated tests, and deployments in isolated container environments.
- **Development Environments:** Create consistent development and testing environments that mirror production configurations.
- **Self-Hosted Applications:** Deploy popular containerized applications like databases, message queues, monitoring tools, and content management systems.
- **Edge Computing:** Run lightweight, portable containers close to users or data sources for reduced latency.

Conclusion

In this guide, you deployed Vultr's Docker Marketplace Application and configured it for production use. You verified the installation, configured firewall security, learned essential container and image management operations, worked with volumes and networks for persistent storage and networking, installed Docker Compose for multi-container orchestration, and implemented security best practices. With this production-ready Docker environment, you can deploy, manage, and scale containerized applications efficiently on Vultr's infrastructure.



VULTR

