Docker and NetBird User Guide

This document is divided into two parts, covering the usage of Docker (containerized application deployment) and NetBird (secure remote access), respectively. The functionalities are independent, allowing you to choose based on your needs.

Part 1: Deploying Applications with Docker Containers

Docker allows you to package your application and its dependencies into a standalone container, enabling easy deployment and operation on RK3566 devices. The following uses an HTTP server written in C as an example to demonstrate the complete containerization process.

1. Preparation
Ensure your RK3566 device is flashed with firmware that includes Docker, and that the Docker service is running. Verification method:

bash
docker --version
If version information is output, Docker is ready.

2. Write the C Program
Create a directory mycapp on your PC and create a new file hello.c with the following content:

c
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 8080

void handle_request(int client_fd) {
    char buffer[1024];
    read(client_fd, buffer, sizeof(buffer) - 1);
    const char *response =
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: text/plain\r\n"
        "\r\n"
        "Hello from Docker on RK3566!";
    write(client_fd, response, strlen(response));
    close(client_fd);
```

```
}

int main() {
    int server_fd, client_fd;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    bind(server_fd, (struct sockaddr *)&address, sizeof(address));
    listen(server_fd, 3);

    printf("HTTP server listening on port %d\n", PORT);
    while (1) {
        client_fd = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
        handle_request(client_fd);
    }
    return 0;
}
```

3. Write the Dockerfile

In the mycapp directory, create a file named Dockerfile (no extension) with the following content:

dockerfile

```
# Use multi-stage build to reduce image size
# Stage 1: Compile
FROM arm64v8/alpine:latest AS builder
RUN apk add --no-cache gcc musl-dev
WORKDIR /app
COPY hello.c .
RUN gcc -static -o hello hello.c

# Stage 2: Runtime image
FROM arm64v8/alpine:latest
WORKDIR /app
COPY --from=builder /app/hello .
```

EXPOSE 8080
CMD ["./hello"]
Explanation:

FROM arm64v8/alpine:latest specifies the base image as Alpine Linux (ARM64 version), which is small in size.

RUN apk add ... installs the compilation tools.

COPY hello.c . copies the source code into the image.

gcc -static -o hello hello.c performs static compilation to avoid runtime dependencies on dynamic libraries.

The second stage only copies the compiled executable. The final image contains only the hello program and the Alpine base system, resulting in a size of just a few MB.

4. Copy Files to the RK3566 Device
Copy the mycapp directory to a directory on the device (e.g., /userdata/mycapp) using scp or a USB drive.

5. Build the Docker Image on the Device
Log in to the device and navigate to the directory:

bash
cd /userdata/mycapp
docker build -t mycapp .
The build process may take a few minutes. Once complete, view the image:

bash
docker images
6. Run the Container
bash
docker run -d --name mycapp -p 8080:8080 --restart always mycapp
Parameter explanation:

-d: Run in the background.
--name mycapp: Specify the container name.
-p 8080:8080: Map the host's port 8080 to the container's port 8080.
--restart always: Automatically restart the container if it exits, enabling startup on boot.

7. Test the Service
Execute on the device locally:

bash
curl http://localhost:8080
The output should be Hello from Docker on RK3566!.

If the device has network connectivity, other devices on the same local network can access it via http://<Device_IP>:8080.

8. Other Common Docker Commands
bash
View running containers
docker ps

View container logs
docker logs mycapp

Stop container
docker stop mycapp

Start container
docker start mycapp

Delete container (must be stopped first)
docker rm mycapp

Delete image
docker rmi mycapp
9. Custom Programs
You can package your own programs (C, C++, Python, Go, etc.) using a similar method. Simply modify the compilation steps in the Dockerfile to ensure the resulting executable can run on ARM64.

Part 2: Secure Remote Access with NetBird

NetBird is a zero-trust VPN based on WireGuard that allows you to securely access your RK3566 device from anywhere without needing a public IP address or port forwarding.

1. Prerequisites
The device is flashed with firmware that includes NetBird.
You need a NetBird account (free registration at https://app.netbird.io) or a self-hosted management platform.

2. Start NetBird on the RK3566 Device
Log in to the device and execute:

bash
netbird up
On first run, you will be prompted for authentication. You can complete this in one of two ways:

Option A: Using a Setup Key (Recommended)

Log in to the NetBird console (https://app.netbird.io).
Go to the "Peers" page and click "Add peer" to generate a Setup Key (valid for 24 hours by default).
Copy the key and enter it on the device when prompted:
text
Please enter setup key: <Paste your key>
Wait a few seconds; the device will join the network.
Option B: Browser Login

The device will print a URL (e.g., https://app.netbird.io/setup?token=xxx).
Open this URL in your computer's browser and follow the prompts to log in.
The device will automatically establish the connection.
3. Check Connection Status
bash
netbird status
Example output:

text
Daemon version: 0.27.2
CLI version: 0.27.2
Management: Connected
Signal: Connected
NetBird IP: 100.64.0.2
Note the NetBird IP; this is the device's virtual IP within the VPN.

4. Install the NetBird Client on Your Computer
Go to https://app.netbird.io/download and download the client for your operating system (Windows, macOS, Linux).
Install and run the client, logging in with the same account. Upon login, your computer will also receive a VPN IP (e.g., 100.64.0.1).

5. Access the Device Remotely via NetBird
5.1 SSH Access
If the device has an SSH service enabled (like dropbear), execute in your computer's terminal:
bash
ssh root@100.64.0.2
Enter the device's root password to log in.

5.2 Accessing a Web Service
If an HTTP service is running on the device (for instance, the Docker container's exposed port 8080), enter in your computer's browser:

text
http://100.64.0.2:8080
You will see the service response.

5.3 Other Services
Any TCP/UDP service, as long as the program listens on 0.0.0.0 (all interfaces), can be accessed via <VPN_IP>:<port>.

6. Manage the NetBird Network
View all online devices: Log in to the NetBird console; the "Peers" page shows the list of devices and their VPN IPs.

Set firewall rules: ACLs can be configured in the console to restrict which devices can communicate with each other.

Disconnect: Execute netbird down on the device to disconnect from the VPN.

7. Enable NetBird Auto-Start on Boot
The NetBird service typically runs at system startup (via init scripts). To ensure it is enabled, you can check /etc/init.d/rcS or manually add the start command:

bash
echo "netbird up" >> /etc/init.d/rcS
8. Troubleshooting

Connection failure: Check if the device's network connection is functioning. Verify if Management and Signal show Connected in netbird status. If they show Disconnected, there might be network issues or account configuration problems.

Setup Key expired: Generate a new Setup Key and run netbird up again.
Cannot SSH: Confirm that the SSH service is installed and running on the device (ps aux | grep dropbear), and that the firewall is not blocking port 22.