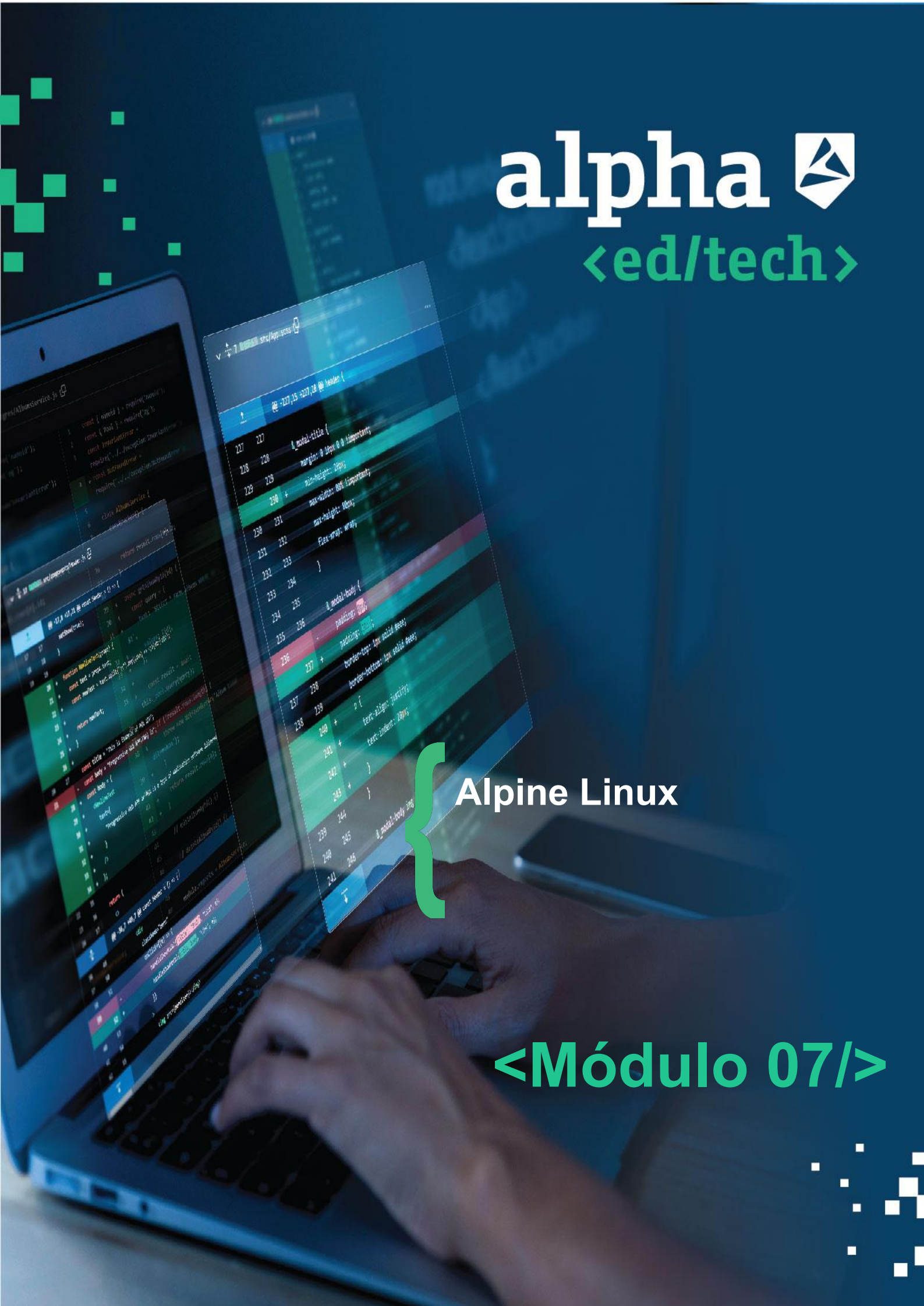
alpha
<ed/tech>
Alpine Linux
<Módulo 07/>

Alpine Linux

1. Introdução ao Alpine Linux

O que é Alpine Linux?

Alpine Linux é uma distribuição Linux ultra-leve, orientada à segurança e baseada em musl libc e BusyBox. Seu tamanho reduzido (cerca de 5MB) e foco em segurança o tornam ideal para containers, servidores e dispositivos embarcados.

Características Principais

- **Tamanho mínimo**: Imagem base de apenas 5MB
- **Segurança**: Kernel com proteções PaX/grsecurity
- **Simplicidade**: Sistema minimalista com apenas o essencial
- **Gerenciador de pacotes**: APK (Alpine Package Keeper)
- **Init system**: OpenRC (mais simples que systemd)

Quando usar Alpine?

✅ **Use Alpine quando:**

- Criar containers Docker;
- Precisar de segurança máxima;
- Trabalhar com recursos limitados;
- Construir microsserviços;
- Criar appliances de rede;

❌ **Evite Alpine quando:**

- Precisar de compatibilidade total com glibc;
- A equipe não tem experiência com musl libc;
- Necessitar de pacotes específicos não disponíveis;

2. Instalação e Configuração Inicial

Instalação Básica

Opção 1: Usando Docker (Recomendado para iniciantes)

```
# Baixar e executar Alpine
docker run -it alpine:latest /bin/sh

# Verificar versão
cat /etc/alpine-release
```

Opção 2: Instalação em VM ou Hardware

1. Baixe a ISO em https://alpinelinux.org/downloads/
2. Grave em pendrive ou monte em VM
3. Boot e execute:

```
# Iniciar instalador
setup-alpine

# Siga os prompts:
# - Selecione teclado (br para português brasileiro)
# - Configure hostname
# - Configure rede
# - Defina senha root
# - Configure timezone (America/Sao_Paulo)
# - Selecione disco e particionamento
```

Primeiro Acesso

```
# Login como root
login: root
password: [sua senha]

# Atualizar sistema
apk update
apk upgrade

# Verificar informações do sistema
uname -a
cat /etc/os-release
```

3. Gerenciamento de Pacotes com APK

O APK (Alpine Package Keeper) é o gerenciador de pacotes do Alpine, similar ao apt (Debian) ou yum (RedHat).

Comandos Essenciais

```
# Atualizar índice de repositórios
apk update

# Atualizar todos os pacotes instalados
apk upgrade

# Buscar pacotes
apk search python
apk search -d python  # com descrição

# Instalar pacotes
apk add python3
apk add python3 py3-pip  # múltiplos pacotes

# Remover pacotes
apk del python3

# Listar pacotes instalados
apk info
```

```
# Informações sobre um pacote específico
apk info python3

# Listar arquivos de um pacote
apk info -L python3

# Descobrir qual pacote fornece um arquivo
apk info --who-owns /usr/bin/python3
```

Cache e Repositórios

```
# Limpar cache
apk cache clean

# Adicionar repositórios (editar /etc/apk/repositories)
vi /etc/apk/repositories

# Repositórios comuns:
# http://dl-cdn.alpinelinux.org/alpine/v3.18/main
# http://dl-cdn.alpinelinux.org/alpine/v3.18/community
# http://dl-cdn.alpinelinux.org/alpine/edge/testing
```

Pacotes Virtuais (Organização)

```
# Criar pacote virtual para dependências temporárias
apk add --virtual build-deps gcc make

# Remover todas as dependências de uma vez
apk del build-deps
```

4. Configuração do Sistema

Editor de Texto

```
# Instalar editor (escolha um)
apk add nano      # mais simples
apk add vim       # mais poderoso
# Ou usar vi (já vem instalado)
```

Configuração de Timezone

```
# Listar timezones disponíveis
ls /usr/share/zoneinfo/
# Configurar timezone
setup-timezone -z America/Sao_Paulo
# Ou manualmente:
ln -sf /usr/share/zoneinfo/America/Sao_Paulo /etc/localtime
```

Configuração de Locale

```
# Alpine usa musl, que tem suporte limitado a locales
# Para a maioria dos casos, UTF-8 funciona bem
export LANG=C.UTF-8
export LC_ALL=C.UTF-8
```

```
# Tornar permanente (adicionar ao /etc/profile)
echo 'export LANG=C.UTF-8' >> /etc/profile
```

Hostname

```
# Configurar hostname
setup-hostname meu-alpine

# Ou manualmente:
echo "meu-alpine" > /etc/hostname
hostname -F /etc/hostname
```

5. Usuários e Permissões

Gerenciamento de Usuários

```
# Criar novo usuário
adduser joao
# Será solicitada a senha e informações

# Criar usuário do sistema (sem home)
adduser -S -D -H nginx

# Deletar usuário
deluser joao

# Modificar usuário
passwd joao  # mudar senha
```

Grupos e Sudo

```
# Instalar sudo
apk add sudo

# Adicionar usuário ao grupo wheel (administradores)
adduser joao wheel

# Configurar sudoers
visudo

# Descomentar ou adicionar:
%wheel ALL=(ALL) ALL
```

Permissões de Arquivo

```
# Comandos padrão do Linux
chmod 755 script.sh
chown joao:joao arquivo.txt
chgrp developers projeto/
```

6. Rede e Conectividade

Configuração de Rede

```
# Configuração automática via setup
setup-interfaces
# Configuração manual (/etc/network/interfaces)
vi /etc/network/interfaces
```

Exemplo de configuração estática:

```
auto eth0
iface eth0 inet static
    address 192.168.1.100
    netmask 255.255.255.0
    gateway 192.168.1.1
```

Exemplo DHCP:

```
auto eth0
iface eth0 inet dhcp
```

Reiniciar Rede

```
# Reiniciar serviço de rede
rc-service networking restart
# Ou usar ifup/ifdown
ifdown eth0
ifup eth0
```

Ferramentas de Rede

```
# Instalar ferramentas úteis
apk add curl wget net-tools bind-tools
# Testar conectividade
ping google.com
curl https://api.github.com
# DNS lookup
nslookup google.com
dig google.com

# Informações de rede
ifconfig
ip addr show
ip route show
```

SSH

```
# Instalar OpenSSH
apk add openssh
# Habilitar e iniciar
rc-update add sshd
rc-service sshd start

# Configuração (/etc/ssh/sshd_config)
vi /etc/ssh/sshd_config

# Recomendações de segurança:
# PermitRootLogin no
# PasswordAuthentication no (após configurar chaves)
# Port 2222 (mudar porta padrão)
```

7. Alpine em Containers Docker

Criando Dockerfile com Alpine

Exemplo 1: Aplicação Python Simples

```
FROM alpine:3.18

# Instalar Python e dependências
RUN apk add --no-cache python3 py3-pip

# Criar diretório de trabalho
WORKDIR /app

# Copiar arquivos
COPY requirements.txt .
RUN pip3 install --no-cache-dir -r requirements.txt

COPY . .

# Expor porta
EXPOSE 5000

# Comando de inicialização
CMD ["python3", "app.py"]
```

Exemplo 2: Build Multi-Stage (Node.js)

```
# Stage 1: Build
FROM alpine:3.18 AS builder

RUN apk add --no-cache nodejs npm

WORKDIR /build
COPY package*.json ./
RUN npm ci --only=production

# Stage 2: Runtime
FROM alpine:3.18

RUN apk add --no-cache nodejs

WORKDIR /app
COPY --from=builder /build/node_modules ./node_modules
COPY . .

EXPOSE 3000
CMD ["node", "server.js"]
```

Exemplo 3: Aplicação com Compilação (Go)

```
# Build stage
FROM golang:1.21-alpine AS builder

RUN apk add --no-cache git

WORKDIR /build
COPY go.mod go.sum ./
RUN go mod download
```

```
COPY . .
RUN CGO_ENABLED=0 go build -o app

# Runtime stage
FROM alpine:3.18

RUN apk add --no-cache ca-certificates

WORKDIR /app
COPY --from=builder /build/app .

EXPOSE 8080
CMD ["./app"]
```

Boas Práticas com Alpine no Docker

```
# 1. Use versões específicas
FROM alpine:3.18  # não use 'latest'

# 2. Combine comandos RUN para reduzir layers
RUN apk add --no-cache \
    python3 \
    py3-pip \
    gcc \
    && pip3 install --no-cache-dir flask \
    && apk del gcc  # remover dependências de build

# 3. Use --no-cache para não armazenar índice APK
RUN apk add --no-cache python3

# 4. Use pacotes virtuais para build dependencies
RUN apk add --no-cache --virtual .build-deps \
    gcc \
    musl-dev \
    && pip install mysqlclient \
    && apk del .build-deps

# 5. Crie usuário não-root
RUN adduser -D appuser
USER appuser
```

Exemplo Completo: API Flask

Dockerfile:

```
FROM alpine:3.18

# Instalar dependências do sistema
RUN apk add --no-cache python3 py3-pip

# Criar usuário não-root
RUN adduser -D flaskuser

# Configurar diretório de trabalho
WORKDIR /app
RUN chown flaskuser:flaskuser /app

# Mudar para usuário não-root
USER flaskuser

# Instalar dependências Python
COPY --chown=flaskuser:flaskuser requirements.txt .
RUN pip3 install --user --no-cache-dir -r requirements.txt

# Copiar código da aplicação
COPY --chown=flaskuser:flaskuser . .

# Expor porta
EXPOSE 5000

# Comando de inicialização
CMD ["python3", "app.py"]
```

app.py:

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Hello from Alpine Linux!"})

@app.route('/health')
def health():
    return jsonify({"status": "healthy"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

requirements.txt:

```
Flask==2.3.0
```

Build e execução:

```
# Build
docker build -t minha-api-flask .

# Executar
docker run -p 5000:5000 minha-api-flask

# Testar
curl http://localhost:5000
```

8. Serviços e Inicialização

OpenRC (Init System)

Alpine usa OpenRC em vez de systemd. Comandos principais:

```
# Listar todos os serviços
rc-status

# Iniciar serviço
rc-service nginx start

# Parar serviço
rc-service nginx stop

# Reiniciar serviço
rc-service nginx restart

# Status do serviço
rc-service nginx status

# Adicionar serviço ao boot
rc-update add nginx default

# Remover serviço do boot
rc-update del nginx default

# Listar serviços no runlevel atual
rc-update show
```

Runlevels

```
# Runlevels do Alpine:
# - boot: serviços essenciais de boot
# - default: serviços padrão
# - shutdown: serviços de desligamento

# Adicionar serviço a runlevel específico
rc-update add sshd default

# Mudar runlevel
rc default
rc shutdown
```

Criando um Serviço Customizado

```
# Criar script do serviço em /etc/init.d/
vi /etc/init.d/meuapp
```

Exemplo de script:

```
#!/sbin/openrc-run

name="Minha Aplicação"
command="/usr/local/bin/meuapp"
command_args=""
command_user="appuser:appuser"
pidfile="/run/${RC_SVCNAME}.pid"
command_background="yes"
```

```
depend() {
    need net
    after firewall
}

start_pre() {
    checkpath --directory --owner $command_user /var/log/meuapp
}
```

Ativar o serviço:

```
# Dar permissão de execução
chmod +x /etc/init.d/meuapp

# Adicionar ao boot
rc-update add meuapp default

# Iniciar
rc-service meuapp start
```

9. Segurança e Hardening

Atualizações de Segurança

```
# Atualizar regularmente
apk update && apk upgrade

# Verificar pacotes com vulnerabilidades
apk audit
```

Firewall com iptables

```
# Instalar iptables
apk add iptables

# Regras básicas
# Permitir loopback
iptables -A INPUT -i lo -j ACCEPT

# Permitir conexões estabelecidas
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Permitir SSH
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Permitir HTTP/HTTPS
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Bloquear todo o resto
iptables -P INPUT DROP

# Salvar regras
/etc/init.d/iptables save

# Habilitar no boot
rc-update add iptables
```

Fail2Ban

```
# Instalar
apk add fail2ban

# Configurar
cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
vi /etc/fail2ban/jail.local

# Habilitar para SSH
[sshd]
enabled = true
port = ssh
logpath = /var/log/messages

# Iniciar e habilitar
rc-service fail2ban start
rc-update add fail2ban
```

Auditoria e Logs

```
# Instalar syslog
apk add syslog-ng

# Iniciar e habilitar
rc-service syslog-ng start
rc-update add syslog-ng

# Ver logs
tail -f /var/log/messages

# Rotação de logs
apk add logrotate
```

10. Casos Práticos

Caso 1: Servidor Web Nginx

```
# Instalar Nginx
apk add nginx

# Criar diretório para site
mkdir -p /var/www/html

# Criar página de teste
cat > /var/www/html/index.html << 'EOF'
<!DOCTYPE html>
<html>
<head>
    <title>Alpine Linux - Nginx</title>
</head>
<body>
    <h1>Servidor funcionando!</h1>
    <p>Rodando em Alpine Linux</p>
</body>
</html>
EOF
```

```
# Configurar Nginx
vi /etc/nginx/http.d/default.conf

# Conteúdo básico:
server {
    listen 80;
    server_name _;
    root /var/www/html;
    index index.html;
}

# Testar configuração
nginx -t

# Iniciar Nginx
rc-service nginx start
rc-update add nginx default
```

Caso 2: Container Python com PostgreSQL

Dockerfile:

```
FROM alpine:3.18

RUN apk add --no-cache \
    python3 \
    py3-pip \
    postgresql-client

RUN apk add --no-cache --virtual .build-deps \
    gcc \
    python3-dev \
    musl-dev \
    postgresql-dev \
    && pip3 install --no-cache-dir \
    psycopg2-binary \
    sqlalchemy \
    && apk del .build-deps

WORKDIR /app
COPY . .

CMD ["python3", "main.py"]
```

Caso 3: Microsserviço com Health Check

Dockerfile:

```
FROM alpine:3.18

RUN apk add --no-cache python3 py3-pip curl
RUN pip3 install --no-cache-dir flask gunicorn

WORKDIR /app
COPY . .

EXPOSE 8000

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:8000", "app:app"]
```

Caso 4: Script de Backup Automatizado

```
#!/bin/sh
# /usr/local/bin/backup.sh

BACKUP_DIR="/backup"
DATE=$(date +%Y%m%d_%H%M%S)

# Criar backup
tar -czf "$BACKUP_DIR/backup_$DATE.tar.gz" /var/www/html

# Manter apenas últimos 7 backups
ls -t "$BACKUP_DIR"/backup_*.tar.gz | tail -n +8 | xargs rm -f

echo "Backup concluído: backup_$DATE.tar.gz"
```

Adicionar ao cron:

```
# Instalar cronie
apk add cronie

# Editar crontab
crontab -e

# Adicionar linha (backup diário às 2h)
0 2 * * * /usr/local/bin/backup.sh

# Iniciar crond
rc-service crond start
rc-update add crond
```

Comandos Rápidos - Cheat Sheet

```
# PACOTES
apk update              # Atualizar repositórios
apk upgrade             # Atualizar pacotes
apk add pacote          # Instalar pacote
apk del pacote          # Remover pacote
apk search termo        # Buscar pacote
apk info                # Listar instalados

# SERVIÇOS
rc-service nome start   # Iniciar serviço
rc-service nome stop    # Parar serviço
rc-update add nome      # Adicionar ao boot
rc-status               # Status de serviços

# REDE
ifconfig                # Interfaces de rede
ip addr show            # IPs configurados
ping host               # Testar conectividade

# SISTEMA
df -h                   # Espaço em disco
free -m                 # Memória
top                     # Processos
ps aux                  # Lista processos
```

Recursos Adicionais

- **Site Oficial**: https://alpinelinux.org
- **Wiki**: https://wiki.alpinelinux.org
- **Pacotes**: https://pkgs.alpinelinux.org
- **Docker Hub**: https://hub.docker.com/_/alpine
- **Fórum**: https://forum.alpinelinux.org

Conclusão

O Alpine Linux é uma distribuição poderosa e eficiente, ideal para containers, microsserviços e ambientes com recursos limitados. Sua curva de aprendizado pode ser um pouco íngreme inicialmente, mas os benefícios em termos de segurança, desempenho e tamanho compensam o investimento.

Próximos passos sugeridos:

1. Pratique criando containers Docker com Alpine
2. Configure um servidor web simples
3. Explore o gerenciamento de serviços com OpenRC
4. Implemente práticas de segurança
5. Crie seus próprios scripts de automação