Caddy with Cloudflare DNS Plugin - Corrected Deployment Guide

Updated: September 23, 2025
Issue Resolution: acmez library version conflicts and build failures

Problem Analysis

The original deployment encountered version compatibility issues when building Caddy v2.7.6 with the Cloudflare DNS plugin using golang:1.23. The specific errors:

```
unknown field PropagationTimeout in struct literal of type certmagic.DNS01Solver
cannot use iss.ExternalAccount (variable of type *"github.com/mholt/acmez/acme".EAB) as *"github.com/mholt/acmez/v3/acme".EAB
```

Root Cause

1. **acmez Library Evolution**: Caddy transitioned from the “lego” ACME library to their own “acmez” fork for better performance and features
2. **API Breaking Changes**: The transition from acmez v1/v2 to acmez v3 introduced breaking changes in the External Account Binding (EAB) interface
3. **Plugin Lag**: The caddy-dns/cloudflare plugin may be using older acmez APIs while newer Caddy core expects acmez v3
4. **Build Environment**: Using raw golang images instead of Caddy’s official builder images can cause dependency conflicts

Solutions Overview

We provide **three tested solutions**, listed by preference:

Solution 1: Pre-built Images (RECOMMENDED)

- **Pros**: Zero compilation, tested compatibility, automatic updates
- **Cons**: Less customization control
- **Best for**: Most users, production deployments

Solution 2: Multi-stage xcaddy Build

- **Pros**: Full control, custom modules, reproducible builds
- **Cons**: Longer build times, requires maintenance
- **Best for**: Advanced users, custom requirements

Solution 3: Version-pinned Builds

- **Pros**: Predictable builds, compatibility testing
- **Cons**: Manual version management, potential security lag
- **Best for**: Environments requiring specific versions

Step-by-Step Deployment

Prerequisites

1. **Cloudflare Account** with domains managed by Cloudflare
2. **Cloudflare API Token** with the following permissions:
   - `Zone:Zone:Read`
   - `Zone:DNS:Edit`
3. **Docker & Docker Compose** installed on your Raspberry Pi 5
4. **Static IP** configured for your Pi

Step 1: Create API Token

1. Go to Cloudflare API Tokens (https://dash.cloudflare.com/profile/api-tokens)
2. Click “Create Token”
3. Use “Custom token” template
4. Set permissions:
   - **Zone → Zone → Read**
   - **Zone → DNS → Edit**
5. Set Zone Resources to include your domains
6. Save the token securely

Step 2: Prepare Directory Structure

```
# Create project directory
mkdir -p /opt/homelab/caddy
cd /opt/homelab/caddy

# Download the corrected files
wget -O Dockerfile https://example.com/Dockerfile.caddy-cloudflare
wget -O docker-compose.yml https://example.com/docker-compose.caddy.yml
wget -O .env.example https://example.com/.env.example

# Copy your existing Caddyfile
cp /path/to/your/Caddyfile ./Caddyfile
```

Step 3: Configure Environment

```
# Copy and edit environment variables
cp .env.example .env
nano .env

# Set your Cloudflare API token
CLOUDFLARE_API_TOKEN=your_token_here
TZ=America/New_York
```

Step 4: Update Caddyfile

Ensure your Caddyfile uses the correct DNS challenge syntax:

```
{
  # Global DNS challenge configuration
  acme_dns cloudflare {env.CLOUDFLARE_API_TOKEN}
}

# Your existing site configurations work as-is
example.com {
  reverse_proxy service:port
}

# For site-specific DNS challenges (alternative)
site2.com {
  reverse_proxy service2:port
  tls {
    dns cloudflare {env.CLOUDFLARE_API_TOKEN}
  }
}
```

Step 5: Deploy with Docker Compose

```
# Pull pre-built image (Solution 1 - RECOMMENDED)
docker compose pull

# OR build custom image (Solution 2)
# docker compose build

# Start services
docker compose up -d

# Check logs
docker compose logs -f caddy
```

Step 6: Verify Deployment

```
# Check container status
docker compose ps

# Test certificate issuance
curl -I https://your-domain.com

# Check Caddy config
docker compose exec caddy caddy validate --config /etc/caddy/Caddyfile
```

Troubleshooting Common Issues

DNS Challenge Failures

Symptoms: `timeout waiting for record to fully propagate`

Solutions:

```
{
  acme_dns cloudflare {env.CLOUDFLARE_API_TOKEN}
  resolvers 1.1.1.1 8.8.8.8
}
```

API Token Errors

Symptoms: `HTTP 403 Forbidden` or `Invalid request headers`

Solutions:
1. Verify token permissions match requirements exactly
2. Check token isn't expired
3. Ensure domain is managed by Cloudflare
4. Test token with curl:

```
curl -X GET "https://api.cloudflare.com/client/v4/zones" \
  -H "Authorization: Bearer your_token_here"
```

Build Failures (if using custom builds)

Symptoms: Module not found, dependency conflicts

Solutions:
1. Switch to pre-built image (Solution 1)
2. Clear Docker build cache: `docker builder prune`
3. Check xcaddy version compatibility
4. Use specific Caddy version tags

Certificate Rate Limits

Symptoms: `too many certificates already issued`

Solutions:
1. Use staging environment for testing:

```
{
  acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
}
```

1. Wait for rate limit reset (weekly)
2. Use different certificate authority

Resource Requirements

Minimum System Requirements

- **RAM**: 1GB available (2GB total recommended)
- **Storage**: 10GB free space for Docker images and certificates
- **CPU**: ARM64 architecture (Raspberry Pi 5 64-bit OS)
- **Network**: Stable internet for certificate challenges

Docker Resource Allocation

```
deploy:
  resources:
    limits:
      memory: 512M
      cpus: '0.5'
    reservations:
      memory: 256M
      cpus: '0.25'
```

Security Best Practices

API Token Security

- Use **scoped tokens** with minimal permissions
- **Rotate tokens** every 90 days
- **Never commit** tokens to version control
- Consider **Docker secrets** for production

Caddy Security

- Enable **automatic security headers**
- Use **strong TLS configurations**
- Implement **rate limiting** where needed
- Regular **updates** of Docker images

Network Security

- Configure **firewall rules** (ports 80, 443)
- Use **internal networks** for service communication
- Implement **fail2ban** for additional protection

Integration with Existing Services

Your existing Caddyfile configuration remains compatible. The services you're currently proxying will work without modification:

- **n8n**: `n8n.sourcecodesamurai.com`
- **Vaultwarden**: `vault.carlparrish.com`
- **Jellyfin**: `movies.streetgeek.media`
- **Firefly III**: `budget.deeplydigital.net`
- **All other services** from your original Caddyfile

Simply deploy this corrected Caddy configuration and your services will receive automatic SSL certificates via Cloudflare DNS challenges.

Maintenance and Updates

Regular Maintenance Tasks

- **Weekly**: Check certificate renewal logs
- **Monthly**: Update Docker images

- **Quarterly**: Rotate API tokens
- **Annually**: Review and optimize configurations

Update Procedure

```
# Update images
docker compose pull

# Restart with new images
docker compose up -d

# Clean old images
docker image prune -f
```

Backup Strategy

```
# Backup certificates and configuration
docker run --rm -v caddy_data:/data -v $(pwd):/backup \
  alpine:latest tar czf /backup/caddy-backup-$(date +%Y%m%d).tar.gz /data

# Backup Caddyfile and compose files
cp Caddyfile docker-compose.yml .env /backup/location/
```

Performance Optimization for Raspberry Pi 5

Caddy-specific Optimizations

```
{
  # Optimize for ARM64
  admin localhost:2019
  auto_https off  # Enable only for domains that need it

  # Performance tuning
  order rate_limit before basicauth
  storage file_system {
    root /data
  }
}
```

System-level Optimizations

- **SSD Storage**: Use external SSD for Docker volumes
- **Memory Management**: Configure swap if needed
- **CPU Frequency**: Ensure proper cooling to prevent throttling
- **Network Optimization**: Use Gigabit Ethernet when possible

Conclusion

This corrected deployment resolves the acmez library version conflicts by using maintained, compatible Docker images. The pre-built image approach (Solution 1) is recommended for most users as it eliminates build complexity while providing reliable automatic HTTPS with Cloudflare DNS challenges.

The key differences from the problematic approach:
1. **Uses official Caddy builder images** instead of raw golang

2. **Leverages maintained compatibility** through CaddyBuilds project
3. **Avoids version conflicts** with pre-tested combinations
4. **Provides multiple fallback options** for different use cases

Your Raspberry Pi 5 homelab will now have robust, automatic SSL certificate management for all your self-hosted services.

Additional Resources

- Official Caddy Documentation (https://caddyserver.com/docs/)
- CaddyBuilds Cloudflare Repository (https://github.com/CaddyBuilds/caddy-cloudflare)
- Cloudflare API Documentation (https://developers.cloudflare.com/api/)
- Caddy Community Forums (https://caddy.community/)
- xcaddy Documentation (https://github.com/caddyserver/xcaddy)