chainguard

Getting Started with Chainguard's Dockerfile Converter

Streamlining Dockerfile Conversion with DFC

Hi, I'm Erika!

- Developer Experience Engineer at Chainguard
  - Docs, Demos, Workshops
- Background in devOps and software development
- Wrote the public-facing user guide for DFC

Agenda

- Introduction to Chainguard Containers
- How DFC Works
- Installation and Usage
- Advanced Use Cases

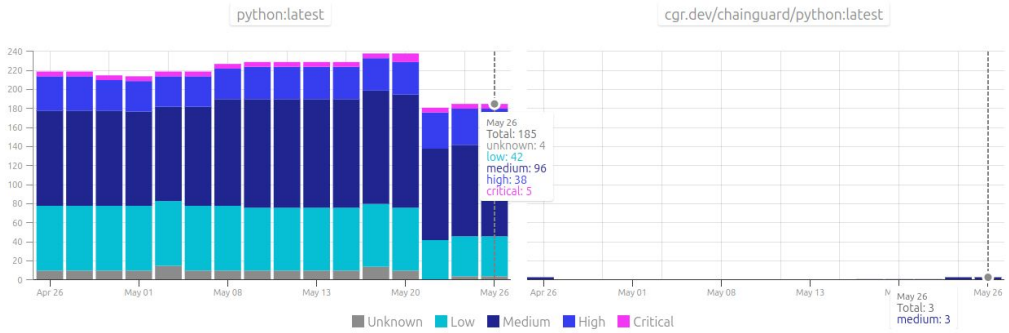
Introduction

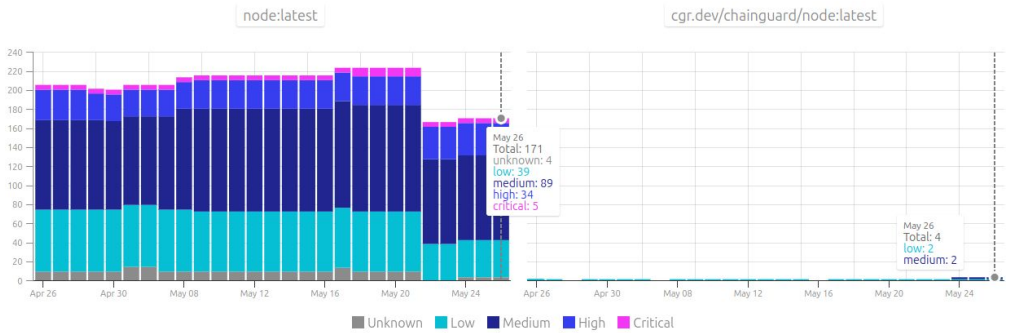
DFC and Chainguard Containers

What is dfc

- Dockerfile Converter is a convenient CLI tool to convert Dockerfiles to use Chainguard Container Images
- Supports Debian/Ubuntu (apt), Alpine (apk), Red Hat (yum / dnf / microdnf)
- Converts entire Dockerfiles or isolated FROM / RUN lines
- "Offline" Operation
- Supports Chainguard Organizations and Custom Registries
- Ability to create custom mappings
- Can be incorporated into internal Go tooling as a library

Why Convert to Chainguard Images?





Minimal, low-to-zero CVE Container Images

APK-based Linux Distro created specifically for containers

Images rebuilt every day to incorporate latest patches

Include SBOMs and provenance attestation for all images

How it Works

Conversion explained

How dfc Works

- FROM
  - dfc will change the registry in FROM lines to match Chainguard's registry format, using your organization's private registry
  - dfc may change the image name based on a mappings file
- APK
  - dfc will modify package manager instructions to use APK instead of apt, dnf, etc
- USER
  - Chainguard Containers run as a non-root user by default; a USER root directive is added before package installations to provide suitable permissions
  - useradd -> adduser, groupadd -> addgroup

Tag Mappings

- For chainguard-base, always use the **latest** tag
- When no tag is specified, or when a tag is not semantic, use **latest**
- When a tag is specified, truncate semantic to major.minor (e.g.: **1.2.3** to **1.2**)
- For any version except **chainguard-base**, add **-dev** suffix when there are **RUN** commands (e.g.: latest-dev,1.2-dev)

- `FROM node:14` → `FROM cgr.dev/ORG/node:14-dev` (if stage has RUN commands)
- `FROM node:14.17.3` → `FROM cgr.dev/ORG/node:14.17-dev` (if stage has RUN commands)
- `FROM debian:bullseye` → `FROM cgr.dev/ORG/chainguard-base:latest` (always)
- `FROM golang:1.19-alpine` → `FROM cgr.dev/ORG/go:1.19-dev` (if stage has RUN commands)
- `FROM node:${VERSION}` → `FROM cgr.dev/ORG/node:${VERSION}-dev` (if stage has RUN commands)

Installation and Usage

Installing and using dfc

Installation

With Homebrew

```
$ brew install chainguard-dev/tap/dfc
```

From Source (needs a Go environment)

```
$ go install github.com/chainguard-dev/dfc@latest
```

Running with Docker

```
$ docker run --rm -v "$PWD":/work cgr.dev/chainguard/dfc
```

Basic Usage

Outputs converted file to stdout

```
$ dfc ./Dockerfile
$ cat ./Dockerfile | dfc - #both work the same way
```

Converts single FROM line

```
$ echo "FROM node" | dfc -
FROM cgr.dev/ORG/node:latest
```

Converts single RUN line

```
$ echo "RUN apt-get update && apt-get install -y nano" | dfc -
RUN apk add --no-cache nano
```

chainguard

DEMO 1

Basic DFC usage

Customizing Behavior

Custom repositories and custom mappings

Customizing DFC Output

Using alternate registry

```
$ dfc --registry="r.example.com/cgr-mirror" ./Dockerfile
```

Use custom mappings file

```
$ dfc --mappings="./custom-mappings.yaml" ./Dockerfile
```

Using JSON / jq to get a list with all packages detected

```
$ dfc -j ./Dockerfile | jq -r '.lines[].run.packages' | grep '"' | cut -d'"' -f
2 | sort -u | xargs
```

chainguard

DEMO 2

Customizing DFC Output

Advanced Usage

Using dfc as a Go library, integration via MCP server

Using as a Go Library

```
ctx := context.Background()

// Parse the Dockefile bytes
dockerfile, err := dfc.ParseDockerfile(ctx, raw)
if err != nil {
	log.Fatalf("ParseDockerfile(): %v", err)
}

// Convert
converted, err := dockerfile.Convert(ctx, dfc.Options{
	Organization: org,
	// Registry: "r.example.com/cgr-mirror", // Optional: registry override
	// Update:   true,                       // Optional: update mappings before conversion
	// ExtraMappings: myCustomMappings,      // Optional: overlay mappings on top of builtin
	// NoBuiltIn: true,                      // Optional: skip built-in mappings
})
if err != nil {
	log.Fatalf("dockerfile.Convert(): %v", err)
}

// Print converted Dockerfile content
fmt.Println(converted)
```

Adding the MCP server to Claude Code

Building the MCP server

```
$ go build -o mcp-server .
```

Adding the MCP server to Claude Code (per project)

```
$ claude mcp add dfc -- ~/dfc/mcp-server/mcp-server
```

Adding the MCP server to Claude Code (system-wide for user)

```
$ claude mcp add dfc -s user -- ~/dfc/mcp-server/mcp-server
```

chainguard

DEMO 3

Using DFC's MCP server with Claude Code

chainguard

go.chainguard.dev/slack

Sign up to join Chainguard's Slack Community!

Q&A

Thank you!

erika@chainguard.dev

edu.chainguard.dev