

Chiseled to perfection

Schlanke und Sichere Container-Images mit .NET bauen





Tobias Richling

Teamleiter Digital Operations @ apetito AG
Passionate about .NET, Docker, Kubernetes and
Software Architecture

 @trichling

 trichling



tobias.richling@apetito.de

Agenda

- Ein Blick in die dunklen Ecken der Softwareentwicklung
- Ein praktischer Leitfaden zur Sauberkeit und Hygiene im eigenen Container
 - Je weniger Geschirr man hat, desto weniger muss man spülen
 - Wer das alte Service von Oma rauschmeißt, muss es gar nicht mehr spülen
- Was wenn beim aufräumen was kaputt geht?



Was ist das Problem?

Ein Blick auf den Dachboden der Softwareentwicklung

Probleme erkennen

- Microsoft Defender For Cloud (oder ähnliches) aktivieren
- Neue Images pushen
- Ein bisschen warten

The screenshot shows the Microsoft Azure portal with the URL https://portal.azure.com/#view/Microsoft_Azure_DevOps/ResourceGroups. The user is signed in as [trichling@gmx.de](#). The page title is "Microsoft Defender for Cloud | Environment settings". The left sidebar lists "General" sections: Overview, Getting started, Recommendations, Attack path analysis, Security alerts, Inventory, Cloud Security Explorer, Workbooks, Community, and Diagnose and solve problems. Below these are sections for Cloud Security (Security posture, Regulatory compliance, Workload protections, Data security, Firewall Manager, DevOps security) and Management (Environment settings, Security solutions, Workflow automation). The "Environment settings" section is currently selected. The main content area displays various connectivity and resource status metrics:

Category	Value	Description
Azure subscriptions	1	Number of Azure subscriptions.
AWS accounts	0	Number of AWS accounts.
AzureDevOps connectors	0	Number of Azure DevOps connectors.
GitLab connectors	0	Number of GitLab connectors.
Total issues	0	Number of total issues.

At the bottom, there are filters for "Environments == All" and "Standards == All", and a link to "More (2)". The footer shows "Name ↑", "Total resources ↑", "Connectivity status ↑", and "De".

Probleme erkennen

- Zur Container Registry gehen
- Alerts anschauen (speziell die für Container Images)
- Diese unterteilen sich in 2 große Kategorien
 - Betriebssystem Pakete (z. B. deb)
 - Anwendungspakete (z. B. NuGet)

The screenshot shows the Microsoft Azure portal interface for a container registry named 'thinkexception'. The left sidebar lists various management options like Overview, Activity log, and Settings. Under Settings, 'Microsoft Defender for Cloud' is selected. The main content area displays a summary of security findings:

- A banner at the top right encourages upgrading to 'Containers, Cloud Posture' plans.
- A 'Visit Microsoft Defender for Cloud' link is present.
- A 'Recommendations' section shows 3 findings:
 - Azure registry container images should have vulnerabilities resolved (Severity: High)
 - Container registries should not allow unrestricted network access (Severity: Medium)
 - Container registries should use private link (Severity: Medium)
- A note below the recommendations states: "Microsoft Defender for Cloud scans your container registry for security vulnerabilities and exposes detailed findings per image. To improve the security posture of your containerized environment and protect it from attacks, enable the optional Microsoft Defender for Containers plan and remediate the findings in your registry."
- An option to "View additional recommendations in Defender for Cloud >" is available.
- A "Security incidents and alerts" section at the bottom notes: "Defender for Cloud uses advanced analytics and global threat intelligence to alert you to malicious activity."

Ein normales Docker-Image...

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 as base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/sdk:8.0 as build
WORKDIR /app
COPY Lab.MicroToDo.Frontend.Api/Lab.MicroToDo.Frontend.Api.csproj Lab.MicroToDo.Frontend.Api/
COPY Lab.MicroToDo.Frontend.Contracts/Lab.MicroToDo.Frontend.Contracts.csproj Lab.MicroToDo.Frontend.Contracts/

RUN dotnet restore "Lab.MicroToDo.Frontend.Api/Lab.MicroToDo.Frontend.Api.csproj"
COPY ..
RUN dotnet publish "Lab.MicroToDo.Frontend.Api/Lab.MicroToDo.Frontend.Api.csproj" -c Release -o /app/publish

FROM base as final
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "Lab.MicroToDo.Frontend.Api.dll"]
```

Docker Image bauen:

```
docker build -f Dockerfile -t thinkexception.azurecr.io/microtodo-frontendapi:dev-net8 .\..
```

... und was da drin ist

- um das herauszufinden, gibt es syft

```
syft `  
thinkexception.azurecr.io/  
microtodo-frontendapi:dev-net8
```

- und wie viel davon ist vom Betriebssystem?
- ratet mal... und bedenkt, dass es sich um ein Runtime-Image handelt - kein SDK-Image!

```
syft thinkexception.azurecr.io/  
microtodo-frontendapi:dev-net8 `  
| sls "deb" `  
| measure
```

- Es sind 94 Betriebssystempakete (bitte merken!)



Beißt der oder will der nur spielen?

- Um das herauszufinden gibt es grype

```
grype thinkexception.azurecr.io/  
  microtodo-frontendapi:dev~net8
```

- Wie viel davon ist vom Betriebssystem mit Priorität High?

```
grype thinkexception.azurecr.io/  
  microtodo-frontendapi:dev~net8  
  | sls deb  
  | sls High  
  | measure
```

- Es sind 3 Verwundbarkeiten mit hoher Dringlichkeit (bitte merken!)



grype

Was kann man tun?

Chiseled Images eilen zur Rettung

Chiseled Images to the rescue

- Microsoft kündigt Chiseled Containers als GA an.
- Diese enthalten wesentlich weniger Abhängigkeiten und sind deshalb kleiner und sicherer
- Vergleichbar mit Google Distroless
- Werden aber vom Upstream Image Provider (ubuntu) bereitgestellt
- Microsoft stellt Chiseled Images für die runtime und aspnet bereit.



Will ich haben!

```
--FROM mcr.microsoft.com/dotnet/aspnet:8.0 as base
+FROM mcr.microsoft.com/dotnet/aspnet:8.0-jammy-chiseled as base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/sdk:8.0 as build
WORKDIR /app
COPY Lab.MicroToDo.Frontend.Api/Lab.MicroToDo.Frontend.Api.csproj Lab.MicroToDo.Frontend.Api/
COPY Lab.MicroToDo.Frontend.Contracts/Lab.MicroToDo.Frontend.Contracts.csproj Lab.MicroToDo.Frontend.Contracts/

RUN dotnet restore "Lab.MicroToDo.Frontend.Api/Lab.MicroToDo.Frontend.Api.csproj"
COPY . .
RUN dotnet publish "Lab.MicroToDo.Frontend.Api/Lab.MicroToDo.Frontend.Api.csproj" -c Release -o /app/publish

FROM base as final
WORKDIR /app
COPY --from=build /app/publish .
ENV DOTNET_ROOT /app/publish
ENTRYPOINT ["dotnet", "Lab.MicroToDo.Frontend.Api.dll"]
```

Docker Image bauen:

```
docker build -f Dockerfile -t thinkexception.azurecr.io/microtodo-frontendapi:dev-net8-chiseled .\..
```

Welche Images gibt es als Chiseled? 1

Und was macht das mit dem Image?

Syft

Statt 94 Paketen nur noch **9**

Grype

Statt 3 High-Vulnerabilities nur noch **0**

Size

Statt 223 MB nur noch **115 MB**



Wie machen die das?

Was ist das kleinste Docker Base Image?

FROM scratch

If you need to completely control the contents of your image, you can create your own base image from a Linux distribution of your choosing, or use the special `'FROM scratch'` base

[1][2][3]

Was kann schiefgehen?

Zu Risiken und Nebenwirkungen

... fragen Sie Ihren Arzt oder Apotheker

Chiseled Images haben

- keine Shell
- keinen Paketmanager
- keinen root-User
- nur die minimale Menge an Paketen, die für .NET-Apps benötigt werden
- keine Codepages (ICU) und Zeitzonen (TZData)

Was muss man da beachten?

"Shell" vs "Exec" Form

DONT: "Shell" form

```
RUN dotnet --list-runtimes
ENTRYPOINT dotnet myapp.dll
CMD dotnet myapp.dll -- args
```

DO: "Exec" form

```
RUN ["dotnet", "--list-runtimes"]
ENTRYPOINT ["dotnet", "myapp.dll"]
CMD ["dotnet", "myapp.dll", "--", "args"]
```

wget, apt und Co

DONT: wget in der final stage

```
# build stage
FROM mcr.microsoft.com/dotnet/sdk:8.0-jammy AS build
...
# final stage/image
FROM mcr.microsoft.com/dotnet/runtime-deps:8.0-jammy-chiseled
RUN wget -O somefile.tar.gz <URL> \
    && tar -oxzf aspnetcore.tar.gz -C /somefile-extracted
```

DO: wget in der build stage

```
# build stage
FROM mcr.microsoft.com/dotnet/sdk:8.0-jammy AS build
RUN wget -O somefile.tar.gz <URL> \
    && tar -oxzf aspnetcore.tar.gz -C /somefile-extracted
...
# final stage/image
FROM mcr.microsoft.com/dotnet/runtime-deps:8.0-jammy-chiseled
...
```

Ablage von Dateien

DONT: Schreiben in das Verzeichniss der Applikation

```
File.WriteAllText("myFile.txt", myText);
```

DO: Dateien im Benutzerprofil ablegen

```
var path = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile), "myFile.txt");
File.WriteAllText(path, myText);
```

Globalisierung und Zeitzonen

- Z. B. Microsoft.SqlClient benötigt Globalization support für die Konvertierung von Unicode in andere Zeichensätze [1]
- Dies verursacht eine Abhängigkeit auf ICU APIs [2] [3]
- Für die Konvertierung von Datum und Uhrzeit werden auch Zeitonendaten benötigt (TZData) [4]

DO - In case you need it

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0-jammy-chiseled-extra as base
```

Dockerfile [5]

Wie wäre es mit aufräumen?

Nur ein gelösches Image ist ein sicheres Image

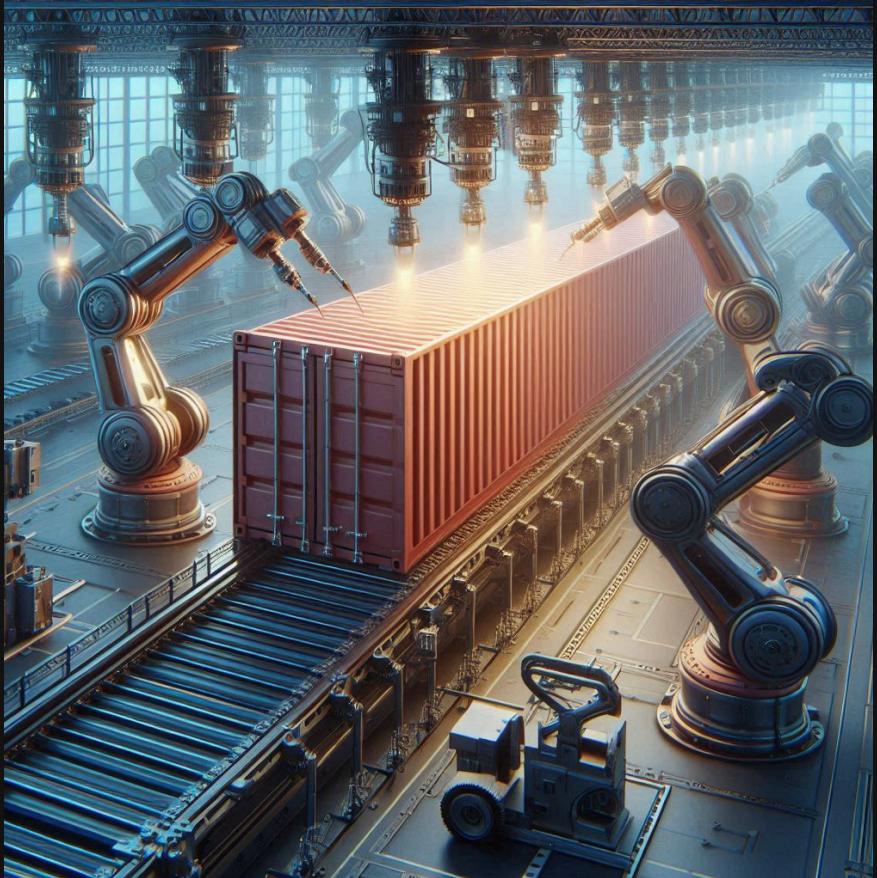
Retention Policy

- Eingebauter Mechanismus, um nicht getaggte Manifeste zu löschen
- Nur für Premium-Tier ACRs verfügbar



ACR Tasks

- Automatisierungs-Layer innerhalb der Azure Container Registry
- Vergleichbar mit DevOps Pipelines / GitHub Actions
- Images bauen, pushen, ausführen, testen, etc.
- Task-Definitionen in YAML
- Ausführung anhand von Triggern (z. B. Push oder Zeitplan)
- Es stehen Befehle zur Verfügung die über die Azure CLI nicht zur Verfügung stehen



ACR Purge

Löscht Images / Manifests anhand bestimmter Filter

Verwaltung von Tasks

Ad hoc ausführung

```
az acr run -r thinkexception `  
    --cmd 'acr purge --help' `  
    /dev/null
```

Anlegen

```
az acr task create -n purgeTest -r thinkexception `  
    --cmd 'acr purge --help' `  
    -c /dev/null
```

Anzeigen

```
az acr task list -r thinkexception  
az acr task show -n purgeTest -r thinkexception
```

Löschen

```
az acr task delete -n purgeTest -r thinkexception
```

Ausführung mit Schedule

YAML-File mit Taskdefinition

```
version: v1.1.0 # do not put this in your yaml file
steps:
- cmd: acr purge --filter '^.*:[^F|f]eature[-0-9]*$' --ago 30d --untagged --keep 1 --dry-run
  disableWorkingDirectoryOverride: true
  timeout: 3600
```

Task aus dem YAML-File erstellen

```
az acr task create -r thinkexception -n purgeFeatureImages ` 
  --file purgeFeatureImages.yaml ` 
  --schedule "0 1 * * Sun" ` 
  --context /dev/null
```

Fazit

Fazit

- Alte Images sind wie alte Socken: Sie stinken und sollten regelmäßig gewechselt werden.
- syft & grype sind gute Tools, um Schwachstellen in Images zu finden (auch in CI / CD Pipelines).
- Chiseled Images sind ein guter Weg, um die Sicherheit und Performance von Containern zu verbessern.
- Aufräumen alter Images ist zusätzlich erforderlich.
- Auch das permanente Überwachen der Container Registry / Kubernetes Cluster ist dringend zu empfehlen.



Vielen Dank für die Aufmerksamkeit!

Habt ihr noch Fragen?



@trichling



trichling



tobias.richling@apetito.de