Linux-Anwendungen unter Windows 10 und Chrome OS nutzen

Pinguin auf Reisen

von Dr. Christian Knermann

Wer Linux-Anwendungen nutzen oder selbst Skripte sowie Anwendungen entwickeln möchte, muss nicht zwingend ein vollwertiges virtuelles oder gar physisches System aufsetzen. Microsoft und Google haben ihre hauseigenen Betriebssysteme Windows 10 und Chrome OS mit Subsystemen zur Ausführung von Linux ertüchtigt. IT-Administrator stellt die höchst unterschiedlichen Ansätze vor.

Quelle: Aleksandr Baidin – 123RF

Wenngleich Linux der große Durchbruch auf dem Desktop bislang verwehrt blieb, ist doch der Erfolg des Open-Source-Projekts unbestritten. Als Unterbau für zahllose Server bildet es das Rückgrat des Internets. In Verbindung mit dem Raspberry Pi und artverwandten Kleinstcomputern ist Linux ebenso eine feste Größe im Internet of Things wie auch Smart Home.

WSL-Version 2

Der ersten Ausgabe von Microsofts Windows Subsystem for Linux (WSL) hatten wir uns bereits gewidmet [1] und dabei gezeigt, wie Sie dabei auch grafische Anwendungen nutzen können. Allerdings war WSL1 im Hinblick auf Funktionalität und Performance noch nicht der große Wurf, denn es stellte lediglich eine Kompatibilitätsschicht dar, die die Systemaufrufe von Linux auf ihre Pendants unter Windows übersetzen, also emulieren, musste. Im Mai letzten Jahres änderte Microsoft dann mit Windows 10 Version 2004 den technischen Unterbau und portierte die neue Funktionalität wenige Monate später auch auf die Windows 10 Versionen 1903 und 1909 zurück. Letzteres gilt allerdings nur für die x64-Plattform. Auf ARM-Systemen bleibt WSL2 den Windows-Ausgaben ab 2004 aufwärts vorbehalten.

Im Gegensatz zu seinem Vorfahren setzt WSL2 nicht auf Emulation, sondern nutzt Microsofts hauseigene Plattform für virtuelle Computer, um einen nativen Linux-Kernel in einer leichtgewichtigen VM auszuführen. Aktualisierungen für diesen Linux-Kernel serviert Microsoft nun im Rahmen der üblichen Windows-Updates – allerdings erst, nachdem Sie einmalig manuell ein Update installiert haben. WSL1 verschwindet damit jedoch nicht komplett, es bleibt Ihnen erhalten und Sie können pro Linux-Distribution flexibel per Up- und Downgrade zwischen den Welten wechseln.

Grundsätzlich empfiehlt Microsoft, die neuere Version zu verwenden, und benennt nur wenige Gründe, die dafür sprechen, dem Vorgänger treu zu bleiben. So geht WSL1 in bestimmten Szenarien sparsamer mit dem Hauptspeicher um und bietet eine höhere Performance, wenn Sie Bedarf haben, aus Linux heraus häufig auf das Dateisystem von Windows zuzugreifen. Sobald die Ordner und Dateien aber im Linux-Dateisystem liegen, sind Sie nach Angaben von Microsoft mit WSL2 bis um den Faktor 20 schneller unterwegs.

Installation mit Handarbeit

Microsoft und Canonical hatten zwar bereits im vergangenen Jahr im Rahmen des Insider-Programms angekündigt, die Installation des WSL mit Hilfe des simplen Befehls *wsl.exe --install* deutlich zu vereinfachen. Bis in die finale Version hatte es dieses Kommando jedoch bis zum Redaktionsschluss leider immer noch nicht geschafft. Und so müssen Sie weiterhin manuell tätig werden. Bis einschließlich Windows 10 20H2 aktivieren Sie WSL am einfachsten, indem Sie in einer mit administrativen Rechten gestarteten Kommandozeile die folgenden Befehle ausführen:

```
dism.exe /online /enable-feature
  /featurename:Microsoft-Windows-
  Subsystem-Linux /all /norestart

dism.exe /online /enable-feature
  /featurename:VirtualMachinePlat-
  form /all /norestart
```

Daraufhin starten Sie Ihr System neu, installieren anschließend das nötige Ker-

nel-Update [2] und definieren zu guter Letzt WSL2 als Standard:

```
wsl --set-default-version 2
```

Distribution auswählen

Damit haben Sie das grundlegende Subsystem erfolgreich eingerichtet und es fehlen nur noch die eigentlichen Linux-Distributionen, die über den Microsoft Store ihren Weg auf das System finden. Dort dürfen Sie zwischen diversen Distributionen, wie etwa Debian, Fedora, mehreren Versionen von Ubuntu oder auch Kali Linux, wählen.

Laden Sie zunächst die gewünschte Geschmacksrichtung von Linux herunter. Anschließend finden Sie passend zur jeweiligen Distribution eine Verknüpfung im Startmenü, über die Sie die Distribution initialisieren und einen Linux-Benutzer nebst Passwort anlegen, der nicht identisch mit Ihrem Benutzerkonto unter Windows sein muss. Daraufhin sollten Sie zunächst die Linux-Instanz mit Updates versorgen. Das erledigen Sie auf Grund der ähnlichen Architektur für Ubuntu, Debian sowie für Kali Linux mit den bekannten Shell-Kommandos:

```
sudo apt-get update

sudo apt-get dist-upgrade

sudo apt-get clean
```

Alle vorhandenen Linux-Distributionen listet unter Windows dieser Befehl auf:

```
wsl --list –verbose
```

In der Ausgabe ist die zuerst installierte Linux-Distribution als Standard mit einem Stern markiert. Diese startet automatisch, wenn Sie den Befehl *wsl* ohne Parameter aufrufen. Beispielsweise mittels

```
wsl --set-default kali-linux
```

ändern Sie den Standard. Schon vor der Veröffentlichung von WSL2 installierte Linux-Instanzen aktualisieren Sie ebenfalls mit Hilfe der Kommandozeile, etwa per:

```
wsl --set-version Ubuntu 2
```

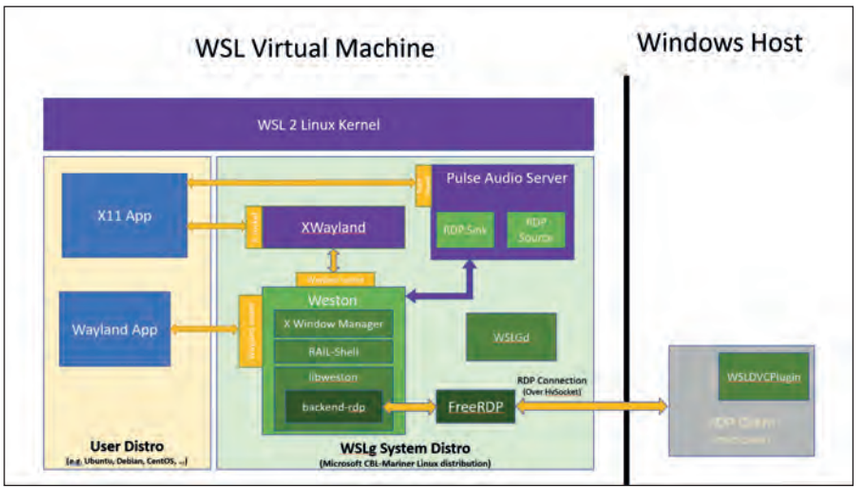

Bild 1: Microsoft arbeitet an einer eigenen GUI-Lösung, die Wayland- und X11-Apps auf RDP übersetzt.

Analog dazu gelingt bei Bedarf auch ein Downgrade auf WSL1.

GUI mit Hindernissen

Native Unterstützung für grafische Linux-Anwendungen ab Werk hatte Microsoft zwar bereits im letzten Jahr in Aussicht gestellt, aber bis zum Redaktionsschluss leider nur als Preview im Rahmen des Insider-Programms realisiert. Interessant ist der technische Ansatz von Microsoft, der eigentlichen Linux-VM eine zweite interne VM an die Seite zu stellen, die Wayland- und X11-Anwendungen unter Linux auf RDP übersetzt (Bild 1). Wann diese Funktion ihren Weg vom Insider-Programm in einen produktiven Build von Windows 10 finden wird, ist noch nicht bekannt.

Auch ohne eine Insider-Version zu installieren, müssen Sie bis dahin nicht auf ein GUI verzichten. Gegenüber WSL1 führen allerdings andere Wege zum Ziel. Hatten die Entwickler von Kali Linux bereits für die frühere Ausgabe ein passendes Skript parat, macht "Win-KeX" das Ganze nun noch einfacher [3]. Mit dem Kommando

```
sudo apt install -y kali-win-kex
```

ertüchtigen Sie das System, sowohl eine komplette Desktopumgebung als auch einzelne Anwendungen darzustellen.

Anderen Distributionen hilft ein X-Server für Windows auf die Sprünge, wie etwa die im Microsoft Store verfügbare App "X410" [4]. Die ist zwar nicht kostenlos, integriert sich aber sehr gut ins System. Starten Sie X410 aus dem Startmenü, erscheint der Server zunächst nur mit einem "X" im System-Tray der Task-Leiste. Über das Kontextmenü des Symbols legen Sie den Modus fest. Für "Windowed Apps" kümmert sich X410 selbst um das Windows-Management und wartet auf Linux-Anwendungen, um diese in separaten Fenstern darzustellen. Ein "Floating Desktop" stellt einen Linux-Desktop in einem skalierbaren Fenster dar, der "Full Desktop" beansprucht die komplette Bildfläche für sich.

Als Besonderheit gegenüber dem Vorgänger verwendet eine Linux-Instanz in Verbindung mit WSL2 intern nicht mehr die lokale Loopback-Adresse 127.0.0.1 oder den Namen "localhost", sondern eine von Windows für virtuelle Computer zugewiesene dynamische IP-Adresse.

Der Start einer grafischen Umgebung erfordert daher zusätzliche Schritte. Aktivieren Sie im Kontext-Menü von X410 die Option "Allow Public Access" und lassen Sie eine passende Ausnahme in der Windows-Firewall zu, wenn X410 Sie darum bittet. Weiter geht es dann unter Linux, wo Sie am Beispiel von Ubuntu den Window-Manager Xfce installieren:

```
sudo apt-get install -y xfce4
  xfce4-terminal
```

Das folgende Shell-Kommando ermittelt dynamisch zur Laufzeit die IP-Adresse

der Linux-Instanz und schreibt diese in die Umgebungsvariable DISPLAY:

```
export DISPLAY=$(cat
  /etc/resolv.conf | grep nameserver
  | awk '{print $2; exit;}'):0.0
```

Anschließend starten Sie mit *xfce4-session* eine Desktop-Sitzung. Der X-Client unter Linux exportiert die grafische Ausgabe an die dynamische IP-Adresse und X410 als X-Server unter Windows stellt diese dar. Schneller kommen Sie ans Ziel, indem ein Batch-Skript unter Windows alle nötigen Schritte in einem Rutsch erledigt:

```
start /B x410.exe /desktop

ubuntu.exe run "if [ -z \"$(pidof
  xfce4-session)\" ]; then export
  DISPLAY=$(cat /etc/resolv.conf |
  grep nameserver | awk '{print $2;
  exit;}'):0.0; xfce4-session; pkill
  '(gpg|ssh)-agent'; fi;"
```

Dabei lädt der Parameter "/desktop" einen "Floating Desktop". Mit "/wm" starten Sie alternativ eine einzelne Anwendung, etwa den Browser Firefox statt einer kompletten XFCE4-Session. Das bis hierher Beschriebene ist vor allem dann nützlich, wenn Sie grafische Linux-Anwendungen nutzen möchten, die es unter Windows nicht gibt.

Linux als Server, Windows als Client

Ein ebenso valides und für viele praktische Anwendungsfälle passendes Szenario ist es, unter Linux nur ein Server-Backend ohne GUI und die grafischen Werkzeuge unter Windows auszuführen. Installieren und starten Sie etwa unter Ubuntu den Webserver nginx:

```
sudo apt-get install -y nginx

sudo service nginx start
```

In der Richtung von Windows zu Linux funktionieren Zugriffe via "localhost" weiterhin. Entsprechend führt Sie unter Windows die Eingabe "http://localhost" im Browser zur Startseite von nginx. Allerdings werden Sie feststellen, dass Sie den Webserver nicht auf herkömmlichem Weg zum automatischen Start bewegen können.

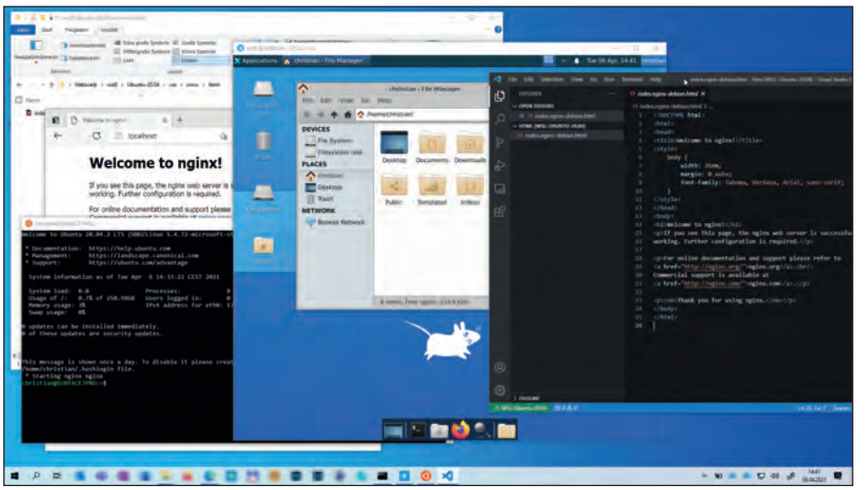

Bild 2: Shell, GUI per X-Server, Dateisystem oder auch Visual Studio Code – Windows und Linux integrieren sich auf vielfältige Weise.

WSL verwendet auch in der zweiten Version nicht das Init-System Systemd. Ein Aufruf des Kommandos *systemctl* zur Verwaltung von Systemd-Diensten funktioniert folglich nicht. Öffnen Sie daher mit *nano ~/.bashrc* die Konfiguration der Shell und fügen Sie am Ende der Datei die folgenden Zeilen ein:

```
# Start Nginx
sudo /etc/init.d/nginx start
```

Zu guter Letzt sorgen Sie mit *sudo visudo* nun noch dafür, dass der Webserver automatisch startet, ohne zuvor ein Passwort zu fordern. Am Ende dieser Konfiguration tragen Sie Folgendes ein:

```
# Start Nginx
%sudo ALL=NOPASSWD:
  /etc/init.d/nginx
```

Sobald Sie das nächste Mal die Linux-Distribution starten, wird der Webserver automatisch den Betrieb aufnehmen. Nach diesem Rezept können Sie nun weitere Dienste, wie etwa PHP und MySQL, ergänzen und so Ihre Entwicklungsumgebung unter Linux ausbauen. Das Editieren von Webseiten und Skripten gelingt anschließend bequem von Windows aus.

Dateisysteme ohne Grenzen

Auch WSL2 erlaubt bidirektionale Dateisystem-Operationen zwischen Windows und Linux. Wie beim Vorgänger finden Sie in der Linux-Umgebung alle unter Windows vorhandenen Laufwerke automatisch mit ihren jeweiligen Laufwerksbuchstaben unter Pfaden wie "/mnt/c" und "/mnt/d" wieder. Umgekehrt greifen Sie von Windows aus per Netzwerkfreigabe auf das Linux-Dateisystem zu. Im Windows-Explorer erscheinen alle installierten Distributionen mit ihrem jeweiligen Namen innerhalb der Freigabe "\\wsl$" und stellen auf diesem Weg ihre Root-Dateisysteme bereit.

So können Sie Ihre gewohnten grafischen Tools unter Windows in Verbindung mit der Zielplattform entsprechenden Serverkomponenten unter Linux verwenden. Microsofts beliebter Editor Visual Studio Code sorgt mit dem Erweiterungspaket "Remote Development" und der darin enthaltenen Komponente "Remote – WSL" für noch mehr Komfort und nahtlose Integration [5]. So öffnet das Shell-Kommando *code <Dateiname>*" unter Linux die jeweilige Datei direkt im Editor unter Windows. Im Gegenzug installiert der unter Windows laufende Editor für die Entwicklung nützliche Erweiterungen in die Linux-Umgebung. Beide Welten arbeiten so optimal zusammen (Bild 2).

Möchten Sie eine nach Ihren Wünschen konfigurierte Linux-Umgebung sichern und auf ein anderes System übertragen, so hilft Ihnen auch hierbei das Kommandozeilen-Tool "wsl". Der folgende Befehl sichert eine Installation von Ubuntu:

```
wsl --export Ubuntu
  c:\temp\ubuntu.tar
```

Mittels des Befehls

```
wsl --import Ubuntu <Zielverzeich-
  nis> c:\temp\ubuntu.tar
```

spielen Sie das Backup auf demselben oder einem anderen System wieder ein. Standardmäßig liegen die Linux-Distributionen im persönlichen Profil des ausführenden Windows-Benutzers unter "%userprofile%\AppData\Local\Packages". Ein frisch installiertes Ubuntu 20.04 LTS ohne Extras belegt dort etwas mehr als 1 GByte, die Sicherungsdatei ist ähnlich groß. Mit Server-Daemons und GUI werden daraus schnell mehrere GByte.

Containervirtualisierung unter Chrome OS

Nicht nur Microsoft Windows eignet sich als Plattform für Linux-Entwicklungen, auch Google Chrome OS bringt sich hier in Stellung. Dass Chrome OS längst mehr ist als eine besonders schlanke Basis für den Browser Chrome, hatten wir bereits kürzlich festgestellt [6]. Native Erweiterungen, Android-Apps und plattformunabhängige Progressive Web Apps (PWA) steigern den Funktionsumfang. Und falls dies nicht reichen sollte, wartet auch Chrome OS mit einem Linux-Subsystem auf. Aufgrund seines Familienstammbaums fällt Chrome OS das Ganze sogar leichter, ist doch das Open-Source-Projekt Chromium OS als Basis für Googles proprietäres Betriebssystem ein Abkömmling der Distribution Gentoo Linux.

Mit einem intern als "Crostini" bezeichneten Projekt hat Google eine Linux-Umgebung in Chrome OS integriert. Wenngleich Google im Rahmen der virtuellen Hausmesse I/O 2021 das System ab Chrome OS 91 als stabil und produktiv verwendbar deklariert hat, ist es aktuell in den Einstellungen von Chrome OS noch als Beta gekennzeichnet und die zentrale Verwaltung per Google Admin Console sowie die Hardware-Integration sind noch nicht ausgereift. So fehlt der Linux-Umgebung aktuell noch Support für Webcams, hardwarebeschleunigte Grafik, Bluetooth sowie Zugriff auf USB-Geräte.

Im Gegensatz zum strikten Sicherheitskonzept von Chrome OS, das jede App und sogar jeden Tab des Browsers Chrome in eine separate Sandbox sperrt, teilen sich alle Anwendungen innerhalb von Linux einen gemeinsamen Sicherheitskontext.

Unter der Haube verwendet Chrome OS einen Hypervisor namens "Crosvm", der als Gast eine VM mit dem Namen "termina" beherbergt. Die führt wiederum den Container-Manager LXD aus, der als Basis für einen Container mit der simplen Bezeichnung "Penguin" dient. Darin läuft die Distribution Debian 10 "Buster". Im Gegensatz zu WSL in Verbindung mit dem Microsoft Store bietet Chrome OS also offiziell keine Unterstützung für verschiedene Distributionen. Fortgeschrittene dürfen sich daran wagen, selbst mittels LXD alternative Container aufzubauen [7]. Im Folgenden wollen wir uns aber auf die Möglichkeiten des von Google bereitgestellten Debian-Containers konzentrieren.

Einrichten und ausbauen

Auf unterstützten Chromebooks finden Sie in den Einstellungen den Bereich "Entwickler" und darunter den Punkt "Linux Entwicklungsumgebung (Beta)". Sollte dieser fehlen, kann dies zwei Gründe haben. Entweder Ihr Chromebook unterstützt die Ausführung einer Linux-Umgebung nicht, was bei älteren oder leistungsschwächeren Geräten der Fall sein kann. Oder es wirkt im Rahmen der zentralen Verwaltung eine Richtlinie auf das Gerät, die den Einsatz von Linux verbietet.

Klicken Sie auf die Schaltfläche "Aktivieren", so beginnt Chrome OS mit der Einrichtung der Linux-Umgebung. Der Assistent fragt dazu nach einem Benutzernamen, der ähnlich wie beim WSL nicht mit Ihrem Benutzernamen unter Chrome OS übereinstimmen muss. Für die Größe des virtuellen Laufwerks schlägt Google zunächst 7,5 GByte vor. Sie können dies gefahrlos akzeptieren und das Laufwerk zu einem späteren Zeitpunkt erweitern, falls der Platz knapp werden sollte. Nach erfolgreicher Installation präsentiert Ihnen Chrome OS ein Terminal-Fenster mit der Linux-Shell des Debian-Systems. Das Terminal finden Sie auch im Launcher des Betriebssystems im Ordner "Linux-Apps" wieder. Wie beim WSL sollten Sie Linux zunächst mit Updates versorgen:

```
sudo apt-get update

sudo apt-get dist-upgrade

sudo apt-get clean
```

Zusätzlich zu Shell-Kommandos integriert Linux auch grafische Anwendungen und das im Vergleich zum WSL mit deutlich weniger Aufwand. Installieren Sie etwa den Dateimanager "Nemo", einen Klon des beliebten Gnome-Dateimanagers "Nautilus":

```
sudo apt-get install -y nemo
```

Grafische Anwendungen wie diese verankern sich analog zum Terminal im Launcher von Chrome OS, von wo Sie sie in der Ablage anpinnen können.

Auch Chrome OS und Linux tauschen bidirektional Daten miteinander aus. Dazu finden Sie in der App "Dateien" von Chrome OS den Ordner "Linux-Dateien", der auf das Verzeichnis "/home/<Username>" innerhalb der Debian-Umgebung verweist. Über die Option "Mit Linux teilen" im Kontextmenü eines jeden Ordners können Sie weitere Pfade an Linux durchreichen. Die Mount-Punkte finden Sie anschließend im Linux-Container jeweils unterhalb von "/mnt/chromeos/MyFiles/<Ordnername>" wieder.

Server-Daemons integrieren

Beim Ausbau der Entwicklungsumgebung hilft Ihnen alternativ zur Shell der grafische Paketmanager "Synaptic", den Sie

Backup und Restore

In den Einstellungen von Chrome OS erzeugen Sie unterhalb von "Entwickler / Linux-Entwicklungsumgebung (Beta) / Sicherung & Wiederherstellung" ein Backup Ihrer Arbeitsumgebung oder holen einen früheren Stand zurück. Chrome OS schreibt dazu den Inhalt des Containers in eine Crostini-Bilddatei, erkennbar an der Dateiendung "*.tini". Tatsächlich handelt es sich dabei um ein komprimiertes TAR-Archiv, sodass Sie daraus auch einzelne Objekte auslesen können. Unmittelbar nach der Installation belegt eine Sicherung nur ungefähr 400 MByte, mitsamt Server-Daemons und Entwickler-Tools werden daraus jedoch schnell mehrere GByte.

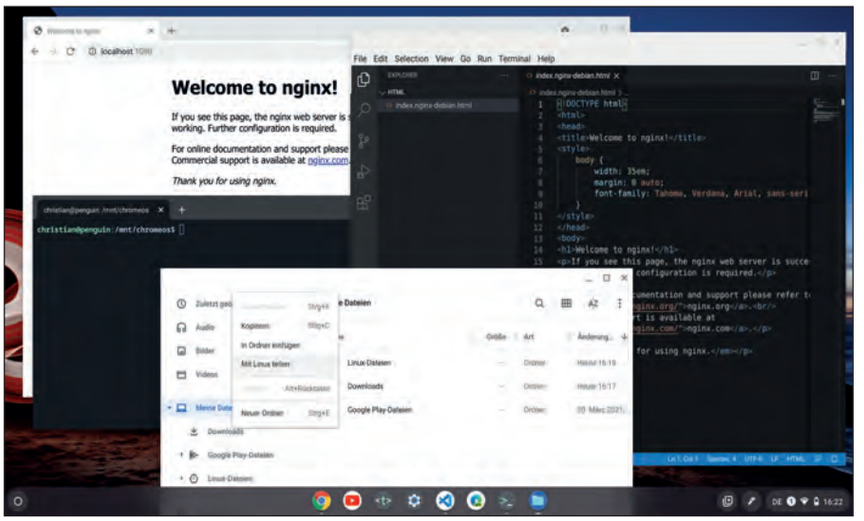

Bild 3: Chrome OS dient hier als Host für den Webserver nginx und den Editor Visual Studio Code.

mit folgenden Kommandos installieren und starten:

```
sudo apt-get install -y synaptic

xhost +si:localuser:root

sudo synaptic
```

Das xhost-Kommando sorgt in diesem Fall dafür, dass der Nutzer root den X-Client des Systems verwenden darf, was ihm standardmäßig nicht gestattet ist. Nach dem Start der gewünschten Anwendung entziehen Sie dem User die Berechtigung wieder mit

```
xhost -si:localuser:root
```

Installieren Sie nun den Webserver nginx und nach Belieben weitere Komponenten wie PHP oder auch MySQL. Im Gegensatz zum WSL bringt der Debian-Container die Startumgebung Systemd mit, sodass Sie Daemons hierüber für den automatischen Start konfigurieren können:

```
sudo systemctl enable nginx

sudo systemctl start nginx
```

Dabei müssen Sie allerdings beachten, dass der Linux-Container mit den Server-Daemons darin nach einem Reboot des Chromebooks nicht von allein wieder anlaufen wird. Das passiert erst, sobald Sie manuell das Terminal oder eine beliebige andere Linux-Anwendung starten. Berücksichtigen Sie weiterhin, dass Sie eine Portweiterleitung konfigurieren müssen, um von Chrome OS und anderen Geräten im Netzwerk auf die Daemons in der Entwicklungsumgebung zugreifen zu können. Hierfür unterstützt Chrome OS allerdings nur TCP- oder UDP-Ports ab 1024 aufwärts. In unserem Beispiel ändern Sie die Konfiguration von nginx mittels:

```
sudo vi /etc/nginx/
  sites-enabled/default
```

In der Konfiguration ändern Sie den TCP-Port des Webservers von 80 auf 1080 und speichern die Datei:

```
(…)
listen 1080 default_server;
listen [::]:1080 default_server;
(…)
```

Nun starten Sie den Webserver mit *sudo systemctl start nginx* neu.

Zu guter Letzt navigieren Sie in den Einstellungen von Chrome OS zum Punkt "Entwickler\ Linux-Entwicklungsumgebung (Beta)\ Portweiterleitung" und fügen dort den TCP-Port 1080 hinzu. Daraufhin können Sie unter Chrome OS per "http://localhost:1080" die Startseite des Webservers erreichen.

Jenseits des offiziellen Paket-Repositories von Debian können Sie problemlos weitere Software ergänzen. Laden Sie unter Chrome OS das Debian-Paket von Microsoft Visual Studio Code herunter und starten Sie die Datei im Dateimanager von Chrome OS. Das System erkennt selbständig, dass es sich um eine Linux-Anwendung handelt und öffnet den Dialog "App mit Linux installieren", sodass Sie die Einrichtung direkt aus Chrome OS heraus anstoßen können. Chrome OS informiert Sie über den Installationsfortschritt und danach erscheint die Anwendung automatisch im Launcher des Systems. Anders als beim WSL befindet sich in diesem Fall Ihre komplette Entwicklungsumgebung mitsamt Daemons und Quellcode-Editor innerhalb des Linux-Containers (Bild 3).

Fazit

Linux-Subsysteme gibt es in diversen Geschmacksrichtungen. Die Ansätze unter Microsoft Windows auf der einen und Google Chrome OS auf der anderen Seite unterscheiden sich zum Teil deutlich. Windows punktet mit der Vielzahl der im Store verfügbaren Distributionen, Chrome OS mit der ausgereifteren Integration grafischer Anwendungen. Beide haben gemeinsam, dass sich ihre Nutzer in vielen Fällen für Anwendungen und Entwicklungsaufgaben unter Linux den Overhead einer ausgewachsenen Linux-VM sparen können. Zudem gelingt der Austausch von Daten zwischen den Welten komplikationslos. Linux erweitert so den Funktionsumfang der jeweiligen Host-Systeme deutlich. *(ln)*

Link-Codes

[1] "Windows Subsystem for Linux" in IT-Administrator 08/2019
l7z71

[2] Updatepaket für den WSL2-Linux-Kernel
l7z72

[3] Win-KeX
l7z73

[4] X410 mit WSL2 verwenden
l7z74

[5] Visual Studio Code mit WSL nutzen
l7z75

[6] "Google Chrome OS im Unternehmen verwalten" in IT-Administrator 05/2021
l7z76

[7] Container unter Chrome OS laufen lassen
l7z77