ChromiumOS Docs

Running Custom Containers Under ChromeOS

Welcome to the containers project where we support running arbitrary code inside of VMs in ChromeOS. This is a heavily-technical document; for user-friendly information, see Set up Linux on your Chromebook documentation and the ChromeOS developer site.

If you're interested in hacking on Crostini itself, take a look at the Crostini developer guide

We won't get into technical details for specific projects as each one already has relevant documentation. We instead will link to them for further reading.

Contents

Overview

There are many codenames and technologies involved in this project, so hopefully we can demystify things here.

Crostini is the umbrella term for making Linux application support easy to use and integrating well with ChromeOS. It largely focuses on getting you a Terminal with a container with easy access to install whatever developer-focused tools you might want. It's the default first-party experience.

The Terminal app is the first entry point to that environment. It takes care of kicking off everything else in the system that you'll interact with.

crosvm is a custom virtual machine monitor that takes care of managing KVM, the guest VM, and facilitating the low-level (virtio-based) communication.

Termina is a VM image with a stripped-down ChromeOS linux kernel and userland tools. Its only goal is to boot up as quickly as possible and start running containers. Many of the programs/tools are custom here. In hindsight, we might not have named it one letter off from “Terminal”, but so it goes.

Concierge is a daemon that runs in ChromeOS which handles lifecycle management of VMs and containers and uses gRPC over vsock to communicate with Maitred.

Maitred is our init and service/container manager inside of the VM, and is responsible for communicating with Concierge (which runs outside of the VM). Concierge sends it requests and Maitred is responsible for carrying those out.

Tremplin is a daemon that runs in the VM to provide a gRPC wrapper for LXD. This includes basic functionality such as creating and starting containers, but also provides other Crostini-specific integration such as setting up a container's primary user, and setting up apt repositories in the guest to match the ChromeOS milestone.

Cicerone is a daemon that runs in ChromeOS which handles all communication directly with the VM and container once the container starts running. Specifically, it communicates with Tremplin (which runs inside

of the VM), and Garcon (which runs in a container inside the VM).

Garcon runs inside the container and provides integration with Cicerone/Chrome for more convenient/natural behavior. For example, if the container wants to open a URL, Garcon takes care of plumbing that request back out.

Seneschal is a daemon that runs in ChromeOS that handles lifecycle management of 9P servers. When Concierge starts a VM, it sends a message to Seneschal to also start a 9s instance for that VM. Then, while configuring the VM, Concierge sends a message to Maitred instructing it to connect to the 9s instance and mount it inside the VM.

9s is a server for the 9P file system protocol. There is one instance of 9s for each VM and it provides that VM with access to the user's data stored outside the VM. This includes things like the Downloads folder, Google Drive, and removable media. The lifecycle of each 9s instance is managed by Seneschal. Each 9s instance starts with no access to any files. Access to specific paths is granted by sending a message to Seneschal, which makes the requested path available to the specified 9s instance. Requests to share paths can only be triggered by some user action.

Sommelier is a Wayland proxy compositor that runs inside the container. Sommelier provides seamless forwarding of contents, input events, clipboard data, etc... between Wayland applications inside the container and Chrome.

Exo is the Chrome Wayland server implementation. Sommelier communicates with Exo using the Wayland protocol.

Chrome does not run an X server or otherwise support the X protocol; thus Sommelier is also responsible for starting up XWayland (in rootless mode), acting as the X window manager to the clients, and translating the X protocol inside the container into the Wayland protocol for Exo.

You can launch crosh and use the vmc command to create new VMs manually. It will only run Termina at this point in time. You can then use vsh to connect to a VM instance and use LXC to run arbitrary containers.

The default container launched via Terminal is Debian with custom packages. See cros-container-guest-tools for more details.

In this flow, the VM is named `termina` and the container is `penguin`.

Quickstart

> See the official Set up Linux on your Chromebook documentation too.

Here's a quick run down of how to get started.

- Buy a Chromebook!
  - All devices launched in 2019 or later will support Crostini.
  - See the Device Support section for more details.
- Make sure your device is up to date.
  - Check for a system updates and reboot if needed.

    - You do **not** need to put the device into developer mode.
- Enable support.
    - Open ChromeOS Settings
        - Click the time, usually in the bottom-right corner of the screen, and then click the cog icon.
        - Alternatively, press the Search/Launcher key and search for the Settings app.
    - Scroll down and click “Advanced”, then find the “Linux development environment” section under the “Developers” heading. After installation, Crostini-related settings can also be found here.
    - Turn it on!
- Profit!

If you're interested in Android Studio, check out their documentation too.

You may also want to pin to the shelf the Terminal or any graphical applications you have installed for ease of access in the future. This can be done by right clicking the app in the app launcher (press the Search/Launcher key) or the shelf for an already-running application.

Features

OK, so you've got your container going, but what exactly can you expect to work?

- Outbound network connections and inbound network connections when port forwarding is enabled.
- Graphical applications, under either Wayland (preferred; via Sommelier) or X (compatibility via Sommelier and XWayland).
- Accelerated Graphics.
- Sound output and microphone capture.
- Bidirectional file sharing with the host OS.
- And more!

Missing Features

There are still many features we're working on fleshing out.

- Peripheral access (USB, Bluetooth, Camera/etc...).
- IMEs.

There are more things we’re thinking about, but we’re being very careful/cautious in rolling out features as we want to make sure we aren't compromising overall system security in the process. The (large) FAQ below should hopefully hit a lot of those topics.

Security

While running arbitrary code is normally a security risk, we believe we've come up with a runtime model that sufficiently mitigates & contains the code. The VM is our security boundary, so everything inside of the

VM is considered untrusted. Our current VM guest image is also running our hardened kernel to further improve the security of the containers, but we consider this a nice feature rather than relying on it for overall system security.

In this model, the rest of the ChromeOS system should remain protected from arbitrary code (malicious or accidental) that runs inside of the containers inside of the VM.

The only contact with the outside world is via crosvm, and each channel talks to individual processes (each of which are heavily sandboxed).

User Data In The Container

With the shift to cloud services, current security thinking highlights the fact that getting account credentials (e.g. your Google/Facebook passwords) is way more interesting than attacking your desktop/laptop. They are not wrong. The current VM/container ChromeOS solution does not currently improve on this. Put plainly, anything entered into the container is the responsibility of the user currently. So if you run an insecure/compromised container, and then type your passwords into the container, they can be stolen even while the rest of the ChromeOS system remains secure.

Extracting Disk Images

Sometimes it's useful to be able to extract the disk image from a VM and move it to another machine, either to recover the data if the VM won't start the normal way, or to track down the source of a bug. This can be done from crosh, even on non-dev mode devices, using the following command:

```
crosh> vmc export <vm name> <file name> [removable storage name]
```

Where `<vm name>` is e.g. `termina`, `<file name>` can be any file name, and `[removable storage name]` is the drive label. Remember to quote this if the name contains spaces e.g. `"My USB Drive"`. The backup will be stored either on the removable storage specified, or in your Downloads folder if you didn't specify one. The file is a gzipped tar archive of the raw VM disk image and can be extracted and mounted on another system as follows:

```
tar -xzf <file name>
sudo mount -o loop <img file> /path/to/mount
```

The container data is then available at `/path/to/mount/lxd/storage-pools/default/containers/penguin/rootfs`.

Persistence

Processes in VMs and containers do not survive logout (since they live in the user's encrypted storage) and are killed automatically. They also do not automatically start at login (to avoid persistent attacks), nor can they automatically run at boot (without a login session) since they wouldn't be accessible (as they're in the user's encrypted storage).

Executable & Writable Code

The Termina VM disk image is downloaded to the writable stateful partition like other Chrome components. In order to make sure the contents aren't modified, we use dm-verity. This also means only images signed by Google may be loaded, and the image is always read-only.

Hardware Attacks

The Meltdown/Spectre vulnerabilities have implications for safely using VMs. We've applied fixes/mitigations to make sure VMs can't attack the host system or other VMs. See the ChromiumOS wiki page for more details.

Lifecycles

Once you've got the Terminal installed (which takes care of installing all the other necessary components like Termina), the system is ready to use.

By virtue of having things installed, nothing starts running right away. In that regard, when you log out, everything is shut down and killed, and when you log in, nothing is automatically restarted.

When you run the Terminal, the Termina VM will be started automatically, and the default Crostini container will be started in that. You can now connect to the container via SSH or SFTP (via the Files app).

Similarly, if you run a Linux application directly (e.g. pinned to your shelf or via the launcher), the Termina VM will be started automatically, and the container that application belongs to will be launched. There's no need to run Terminal manually in these situations.

When you close all visible applications, the VM/containers are not shut down. If you want to manually stop them, you can do so via the Terminal app's context menu, or via crosh.

Similarly, if you want to spawn independent VMs, or more containers, you can do so via crosh and the vmc and vsh commands.

Persistence

All the VMs and containers created, and the data within those containers, will persist across user sessions (logout/login). They are kept in the same per-user encrypted storage as the rest of the browser's data.

If a VM or container are stopped or killed ungracefully (e.g. powerloss), then data might be lost and need recovery like anything else in the system.

Device Support

All devices launched in 2019 or later, as well as some devices launched earlier, support Crostini. Any device which is still receiving updates currently supports Crostini. There are no plans to backport support to devices which are no longer receiving updates.

Enterprise admins have the ability to disable Crostini access for devices which otherwise support it.

Glossary

- **9s**: Server for the 9p file system protocol.
- **ARC++** (Android Runtime for Chrome [plus plus]): The current method for booting Android in a container under ChromeOS.
- **ARCVM** (Android Runtime for Chrome in a VM): The latest method for booting Android under ChromeOS. Unlike ARC++, ARCVM runs Android inside crosvm.
- **Cicerone**: ChromeOS daemon that communicates with containers.
- **Concierge**: ChromeOS daemon that manages VM/container life cycles.
- **Container**: A package (tarball/filesystem image/etc...) full of programs ready to be executed with some levels of isolation.
- **crosh** (ChromeOS shell): A restricted developer shell for running a handful of commands.
- **Crostini**: An umbrella name for providing a polished UI experience to run Linux apps.
- **crosvm**: The ChromeOS Virtual Machine Monitor (akin to QEMU).
- **FUSE**: Filesystem handling in userland which enables a wider variety of formats, remote filesystems, and improves overall security/stability.
- **Garcon**: Daemon in the container for passing requests between the container and Chrome via Cicerone.
- **KVM** (Kernel Virtual Machine): The Linux interface for managing virtual machines.
- **LXC/lxd**: Linux container solution.
- **Maitred**: Agent that runs inside the VM and manages containers.
- **QEMU**: A large/complete virtual machine emulator.
- **Seneschal**: ChromeOS daemon that manages 9p servers.
- **Sommelier**: Wayland proxy compositor in the container that provides seamless forwarding of contents, input events, clipboard data, etc... between Linux apps and Chrome, and seamless X integration.
- **Termina**: Codename for the custom VM that we boot.
- **Terminal**: A shell that acts as the default entry point to Crostini.
- **userland**: Everything not running inside of the kernel. Also known as user space.
- **VM** (Virtual Machine): A way to boot a different operating system in a strongly isolated environment.
- **vmc**: crosh command to manually manage custom VM instances via Concierge.
- **vsh**: Shell that runs inside the VM (not inside of the container).
- **Wayland**: The new graphics stack in the Linux world.
- **WM** (Window Manager): Program responsible for managing windows that other programs create. e.g. window borders, maximizing/minimizing, etc...
- **X**: Umbrella term for the large classical project tasked with making graphics and inputs work in UNIX environments. May refer to the server, client, protocol, WM, or many other facets depending on context. a.k.a. X11, X.Org, and XFree86.
- **XWayland**: An X server that outputs to Wayland.

FAQ

Where can I chat with developers?

All ChromiumOS development discussions happen in our chromium-os-dev Google Group. Feel free to ask anything!

Where can I file bugs and feature requests?

Please check the existing bug list as the issue may already be known or fixed. You can star existing issues to receive updates.

If you want to send feedback, you can file a feedback report and include `#crostini` in the description. Feedback about any part of ChromeOS can be filed via “Alt-Shift-I”.

If you want to file a bug, use this link to route to the right people. You may also want to also file a feedback report and mention the bug number to send device logs that may aid debugging.

When will my device be supported?

If your device does not already support Crostini, it is likely no longer receiving updates and Crostini support will not be added. See the Device Support section for more details.

Do I need to enable developer mode?

There is no need to enable developer mode (where you see the scary screen at boot about OS verification being turned off). These features are all designed to run securely while your system is running in normal/verified mode.

Unrelated to developer mode, some users may want to switch to the dev channel to receive updates quicker.

Can I boot another OS like Windows, macOS, Linux, *BSD, etc...?

Currently, no, you can only boot our custom Linux VM named Termina. See also the next few questions.

Can I run my own VM/kernel?

Currently, no, you can only boot Termina which uses our custom Linux kernel and configs. Stay tuned!

Can I run a different Linux distro?

Of course! The full LXD command line is available, and the included images remote has lots of other distros to choose from. However, we don't test with anything other than the default container that we ship, so things may be broken when running another distro.

I'm running <insert distro here>, how do I get {GUI apps, launcher icons, etc...}?

Sommelier and Garcon binaries are bind-mounted into every container, so no need to install or cross-compile. The systemd units and config files from cros-container-guest-tools will start these daemons in a

systemd user session. It's also a good idea to run `loginctl enable-linger <user>` to allow these to remain running in the background.

Am I running Crostini?

If you're using the Terminal app, or programs in the default container we provide that includes our programs to ease integration (e.g. Sommelier), then yes.

If you're running your own container or VM, then no.

How do I share files between ChromeOS & the container?

The default Crostini container's storage is accessible under “Linux Files” in the ChromeOS Files app. Using Secure Shell, you can set up a SFTP mount to the other remote containers and then browse via the Files app as well.

Folders in ChromeOS can be shared to the container by right clicking them in the Files app and selecting “Share with Linux”. These will be shared under the directory `/mnt/chromeos`.

Can I access files when the container isn't running?

Currently, the container must be running in order to access its content. The default Crostini container will be started automatically when “Linux Files” is accessed from the Files app.

Can I install custom kernel modules?

Currently, no, Termina does not include module support. That means trying to use software that requires building or loading custom kernel modules (e.g. VirtualBox) will not work. See the next question too.

Can I mount filesystems?

Currently, no (*). The containers are implemented using Linux user namespaces and those are quite restricted (by design).

See the FUSE support entry for alternatives.

(*): Technically you can mount a few limited pseudo filesystems (like memory-backed tmpfs), but most people aren't interested in those.

Is FUSE supported?

Yes! Note that unprivileged containers can’t set up loopback mounts (see the next question), so your FUSE driver of choice can’t require a block device.

Can I use loop devices?

Currently, no. See the previous question about mounting filesystems.

Currently, no. See the previous question about mounting filesystems.

Specifically, we're referring to `losetup` and `mount -o loop` which use `/dev/loop-control` and nodes like `/dev/loop0` via the `loop` kernel module.

If you have a use case that wouldn't be solved by supporting FUSE, please file a bug for us.

Can I run a VM inside the VM?

Currently, no, nested KVM is not supported. You could run qemu-system to emulate the hardware and boot whatever OS you want inside of that. Unfortunately, it'll be quite slow as QEMU won't be able to utilize KVM for hardware acceleration.

Can I run a container inside the container?

Yes! You'll probably need to install the relevant packages first for whatever container format you want to run.

What container formats are supported?

Termina only supports running LXC directly. For Kubernetes/Docker/OCI/rkt/etc... containers, see the previous question.

What architecture works on my system?

Since everything is all native code execution, it depends on the device you have.

If you don't know what device you have, you can find this out in two different ways:

- Open chrome://settings/help/details and look at the Platform, then match the board name with our public device list. Look at the "User ABI" field to see what kind of CPU you have.
- Open up crosh and run `uname -m`. This will print the architecture of your current device.

If you see `x86_64`, you'll be able to run code compiled for Intel/AMD (32-bit/64-bit/x32 should all work).

If you see `arm` (or something similar like `armv7l`) or `aarch64`, you'll be able to run code compiled for ARM/ARM64.

Can I run other architectures?

There is currently no integrated support for running e.g. ARM code on an Intel system, or vice-versa. You could handle this yourself (e.g. by using qemu-user), but if you're familiar with qemu-user, then you already know that :).

How many VMs can I run?

You can spawn as many as your system can handle (RAM/CPU-wise). They are all independent of each other.

How can I manually start/stop VMs from crosh?

You can start and stop a Termina VM instance with `vmc start <VM name>` and `vmc stop <VM name>`, respectively. The name of the default VM instance is `termina`.

How can I create/destroy multiple VMs?

You can create and destroy a VM with `vmc create <VM name>` and `vmc destroy <VM name>`, respectively. Also, you can see the list of existing VMs with `vmc list`.

How can I run VMs with data images on external storage?

If your ChromeOS uses v4.19+ kernel or in developer mode, you can run a VM with a disk images on external storage such as an SD card or a USB stick.

If your USB stick is shown as `USB Drive` in the Files app, you can create an extra disk image there with the following command:

```
# Allocate a 1GB data image on the inserted USB stick.
crosh> vmc create-extra-disk --size=1G --removable-media "USB Drive/extra-disk.img"
# Or, you can specify the full path.
crosh> vmc create-extra-disk --size=1G "/media/removable/USB Drive/extra-disk.img"
```

Then, you can start a VM with the disk image mounted.

```
# Make sure the VM instance is stopped.
crosh> vmc stop termina
# Pass the image path with `--extra-disk` option to `vmc start`.
# If you're on kernel < 4.19, you need to be in developer mode and pass in the
# `--untrusted` flag.
crosh> vmc start termina --extra-disk "/media/removable/USB Drive/extra-disk.img"
```

The extra disk will be mounted at `/mnt/external/0/` in the guest.

How many containers can I run?

You can spawn as many as your system can handle (RAM/CPU-wise). Each VM instance can host multiple containers.

Can I run programs that keep running after logout?

Nope! All VMs (and their containers) are tied to your login session. As soon as you log out, all programs are shut down/killed by design.

Since all your data lives in your encrypted home, we wouldn't want that to possibly leak when you log out.

For more details, see the Security section in this doc.

Can I autorun programs when I log in to ChromeOS?

Nope! All VMs (and their containers) need to be manually relaunched. This helps prevent persistent exploits.

For more details, see the Security section in this doc.

Can I autorun programs when I boot?

Nope! See the previous questions, and the Security section.

Can I set environment variables for my container?

Sure! There are a few ways to do this.

- environment.d lets you set environment variables for your `systemd --user` session, which includes the Terminal and all GUI apps.
- If you just want environment variables in your Terminal, set those in your shell's config file, such as `~/.bashrc` or `~/.zshrc`.

Changes to environment variables only take effect for newly started programs. You may also need to restart programs or the entire container for any changes to take effect.

Is multiprofile supported?

No, Terminal is only supported in the primary profile (*). We don't plan on making secondary profiles more featureful.

If you're unfamiliar with multiprofile support, check out the general multiprofile documentation for more details.

(*): The Terminal and other installed GUI applications are disabled in secondary profiles. Users can manually start VMs via crosh and containers therein, but the UI and Files app integration may not work correctly.

Are child accounts supported?

No, Terminal is not supported in child accounts. We don't have plans to make this available to such accounts.

If you're unfamiliar with child accounts, check out the general child accounts documentation for more details.

Are my VMs/containers/data synced/backed up?

Currently, no, nothing is synced or backed up. You're responsible for any data going into the containers.

We hope to improve this situation greatly.

How can I back up a VM?

How can I backup a VM?

The Crostini container can be backed up and restored via ChromeOS Settings. Manually created containers and VMs can be backed up with standard LXC commands and `vmc export` respectively.

Can I access the VM/container files directly (e.g. via the Files app)?

Currently, no, there is no way to access the image files used by the VM. There are no plans to change this.

If you want to back things up, you'll need to do so by hand.

Why is the time inside the VM/container out of sync?

The clock inside of the VM (and by extension, the containers) is automatically kept in sync with ChromeOS's clock. So you do not have to run time keeping services yourself (e.g. ntp). That clock is based off of UTC.

We attempt to sync timezone data into the container via timedatectl. If that doesn't work, we fallback with exporting the `TZ` environment variable.

We don't currently update the timezone details inside the VM itself. We also don't try to update any other timezone setting as they are non-standard across distros. So the time might appear to be wrong at a glance in those environments, or stale if the `TZ` environment variable is used.

See https://crbug.com/829934 for some extended technical details. It's more complicated than you might think!

What copy & paste, and drag & drop formats are supported?

While X/Wayland support an arbitrary number of MIME formats, Exo, the ChromeOS Wayland server, accepts only the formats that Chrome itself does for clipboard copy & paste, and drag & drop.

Clipboard items copied via Chrome, or drags started in Chrome will use formats:

- text as `text/plain;charset=utf-8`, `text/plain;charset=utf-16`, and `text/plain` (using UTF-8 encoding).
- html as `text/html;charset=utf-8`, and `text/html;charset=utf-16`.
- copy & paste images as `image/png`.
- drag & drop image file contents as `application/octet-stream;name="<filename>"`.
- files as `text/uri-list` which is a CRLF-separated list of `file:` URLs.

Exo accepts the following formats from VM Wayland apps:

- text as `text/plain;charset=utf-16`, `text/plain;charset=utf-8`, `text/plain`, and `UTF8_STRING`.
- RTF as `text/rtf`.
- html as `text/html;charset=utf-16`, `text/html;charset=utf-8`, `text/html`.
- images in many formats including `image/png` and `image/jpeg`.
- files as `text/uri-list` which is a CRLF-separated list of `file:` URLs.

You can see the details of what Chrome will offer in exo/data_offer.cc, and what is accepted by Exo in

exo/data_source.cc.

Can I read/write the clipboard automatically from inside the VM?

Currently, no.

From a security point of view, we don't want untrusted code silently or automatically extracting whatever the user has copied. Perhaps your browser session is copying personal data or passwords. It's the same problem that the web platform runs into.

This is not the same thing as users manually pasting data (e.g. Ctrl-V). This is only about programmatic reading.

We don't expect it to stay this way forever. Once we have a permission model and UI to manage these things, we can look into allowing users to grant this permission.

If you use `xclip` or X tools, they often have a local buffer (in XWayland), but it won't automatically sync with the rest of the system.

Do I have to manage VM updates?

Nope! The Termina VM is a component that is updated automatically.

Keep in mind that the VM is separate from the container.

How do I check the Termina version?

The Termina version is tied to the ChromeOS version and updated at the same time. ChromeOS's version can be seen at `chrome://version`, and will look something like `14324.72.0`.

You can also connect to a VM via vsh and run `cat /etc/lsb-release`.

Do I have to manage container updates?

The Google provided packages in the container that communicate with ChromeOS or are required for ChromeOS integration will be automatically updated on a regular basis. This will install/upgrade any necessary dependencies automatically.

Package updates due to security fixes will automatically be installed for you.

There is no automatic upgrading of other installed packages in the container. We'd rather avoid updating packages that might break programs already installed. The container is like any other Linux distro out there, so you'll need to update it from time to time if you want newer software. But if you're happy with things as they are, and stable is better than shiny, then you don't need to worry about ever having to manually manage the system.

If you're looking for a quick recipe to pull in the latest shiny versions, run `sudo apt-get update && sudo apt-get dist-upgrade`.

If you want to control any automatic package updates, edit the file `~/.config/cros-garcon.conf`. It is

automatically installed for new users, and includes documentation inline. **Be warned**: disabling automatic updates can break integration with CrOS.

When a new major version update to Debian is available and tested, users are prompted by ChromeOS and provided with UI to upgrade the container. If you want to upgrade before this is available, you can do so manually.

Can I use IPv6?

Yes. Both dual-stack and IPv6-only networks are supported.

ChromeOS only supports SLAAC; read more in the support page for IPv6 support on ChromeOS.

Can I access layer 2 networking?

Currently, no, networking access is only at layer 3 (i.e. IP). So you won't be able to do any bridging or lower level fun stuff.

It's not clear if/when this will change. Bridging with the outside world is difficult with WiFi, and not many devices have Ethernet connections. We could support layer 2 between containers, but it's not clear how many people want this in order to justify the effort involved.

Do VPNs set up by CrOS/Android (outside of the VM/containers) work?

Yes!

Is audio output supported?

Yes!

Is audio capture (e.g. microphone) supported?

Yes! There is a toggle in ChromeOS Settings to enable this.

Can I access hardware (e.g. USB/Bluetooth/serial)?

USB support for some devices, including Android devices and serial devices, is available via ChromeOS Settings. Direct Bluetooth access is not available but you could use the Web Bluetooth API with a web server running in Crostini.

Can I run graphical applications?

Of course!

Can I run Wayland programs?

Yes, and in fact, these are preferred! Chrome itself deals with Wayland clients heavily, and so you're much more likely to have things "just work" if you upgrade.

Sommelier provides this support seamlessly.

Can I run X programs?

Yes, although you might run into some compatibility kinks, and it probably will never be as perfect as running a traditional X server. However, with the wider community moving to Wayland, it should be good enough.

Sommelier takes care of launching XWayland, acting as the WM, and otherwise translating X and Wayland requests between Chrome and the X programs.

Why are windows sometimes tiny/fuzzy?

While Chrome supports high DPI displays, many Linux applications don't. When a program doesn't properly support DPI scaling, poor results follow.

Currently we expose the native resolution and DPI directly to applications. If they show up tiny or fuzzy, it's because they don't support scaling properly. You should report these issues to the respective upstream projects so that, hopefully someday, it'll "just work".

In the mean time, Sommelier exposes some runtime settings so you can set the scale factor on a per-program basis to work around the misbehavior. Check out Sommelier's documentation for more details.

If you're applying a system-wide zoom or otherwise changing the default display resolution, we attempt to scale the application output to match. This can lead to blurry results. You can adjust the resolution of your display, or tweak things via Sommelier (see above for more details).

Will synergy work?

Synergy will not work (as a client or server). It requires capturing and spoofing inputs (e.g. mouse/keyboard) for all windows. Since we're built on top of Wayland, by design, one client cannot get access to any other client on the system. This is a strong security boundary between clients as we don't want arbitrary code running inside of a container being able to break out and attack other clients (like the browser) and sending arbitrary keystrokes.

There are no plans to ever enable this kind of control from the container. This isn't to say a synergy-like solution will never happen in ChromeOS (e.g. something like CRD), just that the solution won't be synergy or any other tool in a container.

> You can run synergy, and probably get it to convey input events for the single window that it's running under, but that's as close as you'll get.

Can I run Windows programs?

Sure, give WINE a try. Compatibility will largely depend on WINE though, so please don't ask us for support.

Can I run Steam?

Sure, give Steam a shot. Just remember that without accelerated graphics or sound, it's probably not going to be too much fun.

Can I run macOS programs?

Probably not. You could try various existing Linux solutions, but chances are good that they are even rougher around the edges.

Can I develop Android apps (for ARC++)?

Check out the Android Studio site for more details on this.

Why implement crosvm from scratch (instead of using QEMU/kvmtool/etc...)?

We have nothing against any of these other projects. In fact, they're all pretty great, and their designs influenced ours. Most significantly, they did more than we needed and did not have as good a security model as we were able to attain by writing our own. While crosvm cannot do everything those other projects can, it does only what we need it to.

For more details, check out the crosvm project.

Why run VMs? Aren't containers secure?

While containers often isolate themselves (via Linux namespaces), they do not isolate the kernel or similar system resources. That means it only takes a single bug in the kernel to fully exploit the system and steal your data.

That isn't good enough for ChromeOS, hence we put everything inside a VM. Now you have to exploit crosvm via its limited interactions with the guest, and crosvm itself is heavily sandboxed.

For more details, see the Security section in this doc.

Don't Android apps (ARC++) run in a container and not a VM?

While Android apps currently run in a container, eligible devices are migrating to running inside a VM.

For ARC++ container, we try to isolate them quite a bit (using namespaces, seccomp, alt syscall, SELinux, etc...), but at the end of the day, they have direct access to many syscalls and kernel interfaces, so a bug in there is reachable via code compiled with Android's NDK.

If Android apps are in a container, why can't users run code too?

We don't usually accept a low security bar in one place as a valid reason to lower the security bar everywhere. Instead, we want to constantly raise the security bar for all code.

For example, devices that support Android 11+ are being migrated from ARC++ in a container to ARCVM.

Are Android apps (ARC++) going away?

There are no plans to merge the two projects. We share/re-use a lot of the Chrome bridge code though, so it's not like we're doing everything from scratch.

Don't VMs slow everything down?

It is certainly true that VMs add overhead when compared to running in only a container or directly in the system. However, in our tests, the overhead is negligble to the user experience, and well worth the strong gains in system security.

For more details, see the Security section in this doc.

Why run containers inside the VM? Why not run programs directly in the VM?

In order to keep VM startup times low, we need Termina to be as slim as possible. That means cutting out programs/files we don't need or are about.

We use dm-verity which requires the Termina image be read-only for Security, but it also means we can safely share it between VM instances.

Further, the versions of programs/libraries we ship are frequently newer than other distros (since we build off of Gentoo), and are compiled with extra security flags.

Allowing user modifications to the VM prevents a stateless image that always works and is otherwise immune from user mistakes and bugs in programs.

Altogether, it's difficult to support running arbitrary programs, and would result in a system lacking many desired properties outlined above. Forcing everything into a container produces a more robust solution, and allows users to freely experiment without worry.

Also, we love turtles.

Is Foreshadow (a.k.a. L1TF / CVE-2018-3646) handled?

Yes. For more details, see our public documentation.

Can I delete containers I no longer want?

Sure, feel free to delete whatever you want. However, there is no UI or commands currently to help with this.

Note: The default Crostini container is named `penguin`.

Can I delete VMs I no longer want?

Sure, feel free to delete whatever you want. The `vmc destroy` command can be used to delete them manually.

Note: The default Crostini VM is named `termina`.

Can I disable these features?

Administrators can control access to containers/VMs via the management console, so enterprise/education organizations that want to limit this can. If Crostini access is disallowed, Crostini's “Turn On” button in ChromeOS Settings will be greyed out.

Why the name Crostini?

It's a play off crouton which is a project to easily create full Linux environments (including developer tools) for users who turned on developer mode. Crostini aims to satisfy the majority of use cases covered by crouton, and is a larger & tastier snack than a crouton, hence the name.

How is Crostini related to Crouton?

crouton helped define many of the use cases that developers wanted with ChromeOS, so it helped guide Crostini from a requirements perspective. We wanted to make sure that the majority of crouton users would be able to use Crostini instead for their needs, but in a secure environment.

So crouton helped inspire the direction of Crostini, but no code has been shared or reused between the two. It’s not that crouton is bad, it’s simply a completely different model.