Elaboration on Atomic Images and BlueBuild

with demo

by Armin Bade (arminmiau)
originally created for my Lightning Talk in class at HTL Grieskirchen

published 6th January, 2025 on arminmiau.vercel.app

Table of Contents

1 Demo Prerequisites

- GitHub account
- Ability to run VM

2 Explanation

2.1 General idea of Atomic Images

The concept of atomic images, also known as immutable images, refers to Linux distributions packaged similarly to container images. These images are often called "bootable containers." They allow for atomic updates to the distribution and programs in a single operation—hence the name "atomic"—while enabling layering of additional programs.

2.1.1 Reliability & Consistency

Atomic images prioritize reliability and consistency by mounting their root file system as read-only. This prevents "configuration drift" over time, as user changes that could lead to unpredictability are restricted.

2.1.2 Simplified Updates

Traditional update mechanisms risk leaving systems partially updated or broken if interruptions occur. Atomic images mitigate this through transactional updates, where changes take effect only upon successful completion and can be rolled back if needed. For instance, Fedora images using `rpm-ostree` perform automatic background updates.

2.1.3 Improved Security

Writable root file systems are significant attack vectors in Linux. Atomic images reduce this risk by employing a read-only root file system, making them less vulnerable to exploitation.

2.1.4 Container-Optimized Workflow

Linux systems were not traditionally optimized for containerized workloads. Atomic images address this gap by offering minimal, consistent base images designed for hosting containerized applications. Examples include Fedora CoreOS and Flatcar Container Linux.

2.1.5 Easy Reproducibility

Reproducing exact environments across systems is error-prone. Atomic images, akin to OCI and Docker images, use a layered approach, ensuring reliable and consistent setups across multiple systems. This enables large-scale operating system deployments. Paired with containerized applications, they create a fully reproducible stack.

2.1.6 Types and examples

- Minimal base images:
  - Fedora CoreOS
  - RHCOS (similar to CoreOS, but RHEL)
  - Fedora IoT
  - Flatcar Container Linux
  - Universal Blue uCore (based on Fedora CoreOS)
  - openSUSE MicroOS
- Atomic Desktops:
  - Fedora Silverblue (GNOME)
  - Fedora Kinoite (KDE Plasma)
  - Universal Blue Bluefin (based on Silverblue)
  - Universal Blue Aurora (based on Kinoite)
  - Universal Blue Bazzite (gaming orientated)

2.2 Projects

2.2.1 libostree

`libostree` (formerly OSTree) is a shared library and suite of command-line tools for committing and downloading bootable file system trees.

2.2.2 rpm-ostree

`rpm-ostree` integrates `libostree` with `libdnf` (the RPM package system). It powers Fedora/Red Hat images and derivatives like Universal Blue, enabling RPM package layering on base images.

Note: As of Fedora 41, focus has shifted to `DNF5` and `bootc`. Features of `rpm-ostree` are gradually being migrated to `DNF5`.

2.2.3 Build Tools

Specific tools are required to build atomic images. Fedora uses `coreos-assembler` for CoreOS-style systems, while Universal Blue employs Red Hat's `buildah` to build OCI-compatible containers compatible with Docker and Kubernetes.

2.3 Custom Images

Linux is renowned for its customization capabilities, but sharing customizations is often cumbersome. Atomic images simplify this by encapsulating adjustments within the image itself. These adjustments can include style, kernel optimizations, drivers, and preinstalled applications.

2.3.1 Universal Blue

Universal Blue images are built on Fedora images, layering kernel optimizations and drivers (e.g., NVIDIA graphics drivers) on top. They offer a GitHub template for building custom images.

2.3.2 BlueBuild

BlueBuild simplifies the creation of custom Universal Blue images. It provides a workshop that sets up a GitHub repository to jump-start image creation. Future updates aim to enable programmatic image configuration.

2.3.3 Use cases

2.3.3.1 Application Deployment

Custom images streamline deployment by pre-configuring services, security, and compliance requirements. This approach reduces risks and accelerates product rollouts at scale.

2.3.3.2 Developer-Team environments

In teams where everyone uses Linux, a shared image can eliminate "works on my machine" issues by standardizing the development environment.

2.3.3.3 Education and Training environments

Pre-configured sandbox images provide students with secure, ready-to-use environments for practicing DevOps, security, or containerization, with minimal setup.

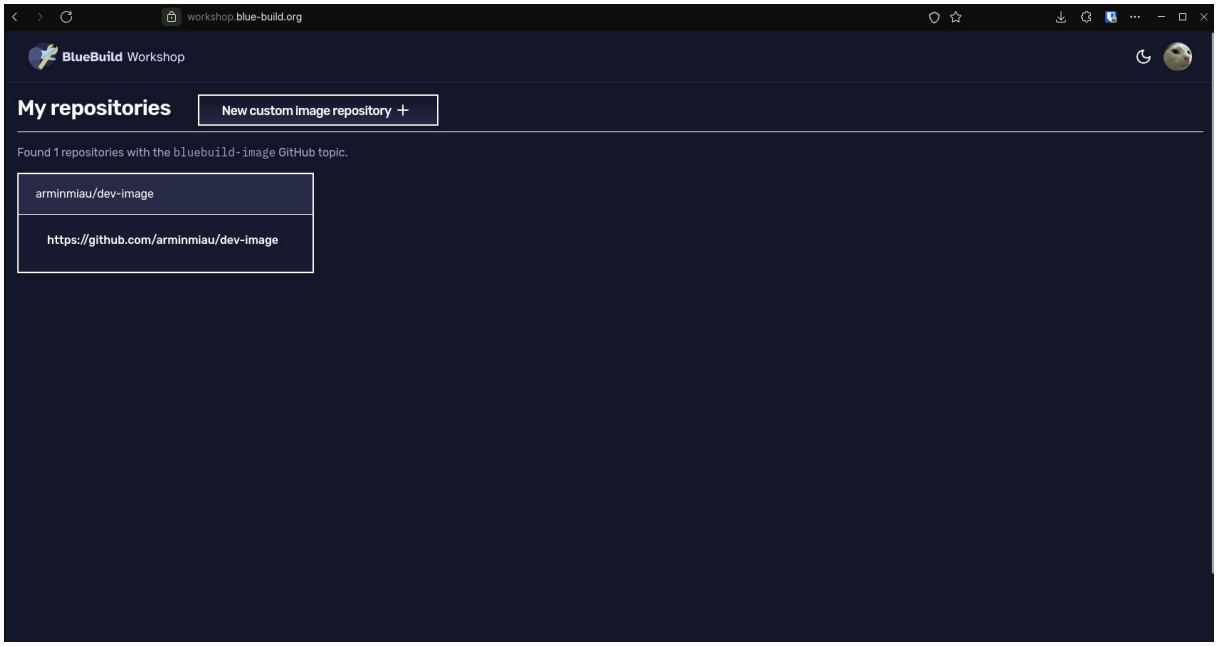
2.3.3.4 Experimentation and Learning

Atomic images offer a safe platform for testing different desktop environments, tools, and configurations. They allow users to experiment with Linux customizations and roll back changes as needed.

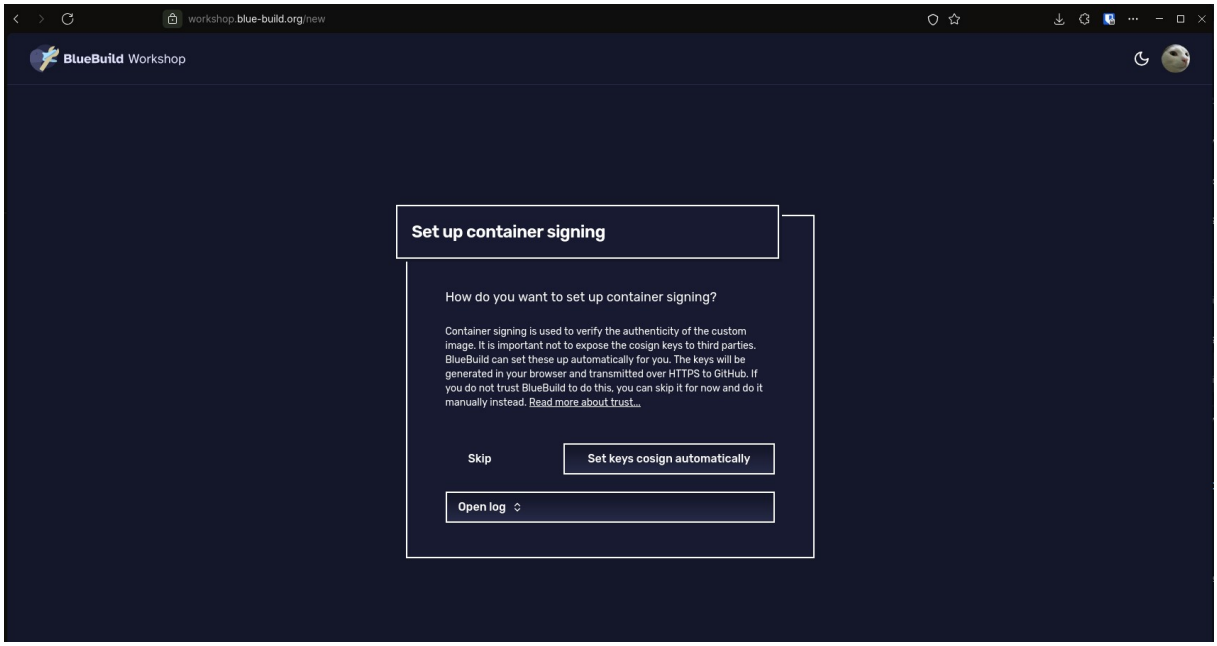
3 Demo

3.1 Setup

Head to https://workshop.blue-build.org/ and login with GitHub. You'll need to authorize the application on the first time. Then click on "New custom Image repository +":



Name it something and in the next step click on "Set keys cosign automtically":



Once it's done open the repository and clone it.

Open recipe.yml in recipes. For this demo we'll choose aurora-dx with latest version.

```yaml
# image will be published to ghcr.io/<user>/<name>
name: demo-image
# description will be included in the image's metadata
description: This is a demo image for my lightning talk.

# the base image to build on top of (FROM) and the version tag to use
base-image: ghcr.io/ublue-os/aurora-dx
image-version: latest
```

Let's install .NET 8 as a layered rpm package and Rider as a Flatpak.

```yaml
- type: rpm-ostree
  repos:
    # -
  install:
    - dotnet-sdk-8.0
  remove:
    # -

- type: default-flatpaks
  notify: true
  system:
    install:
      - com.jetbrains.Rider
    remove:
      # -
```

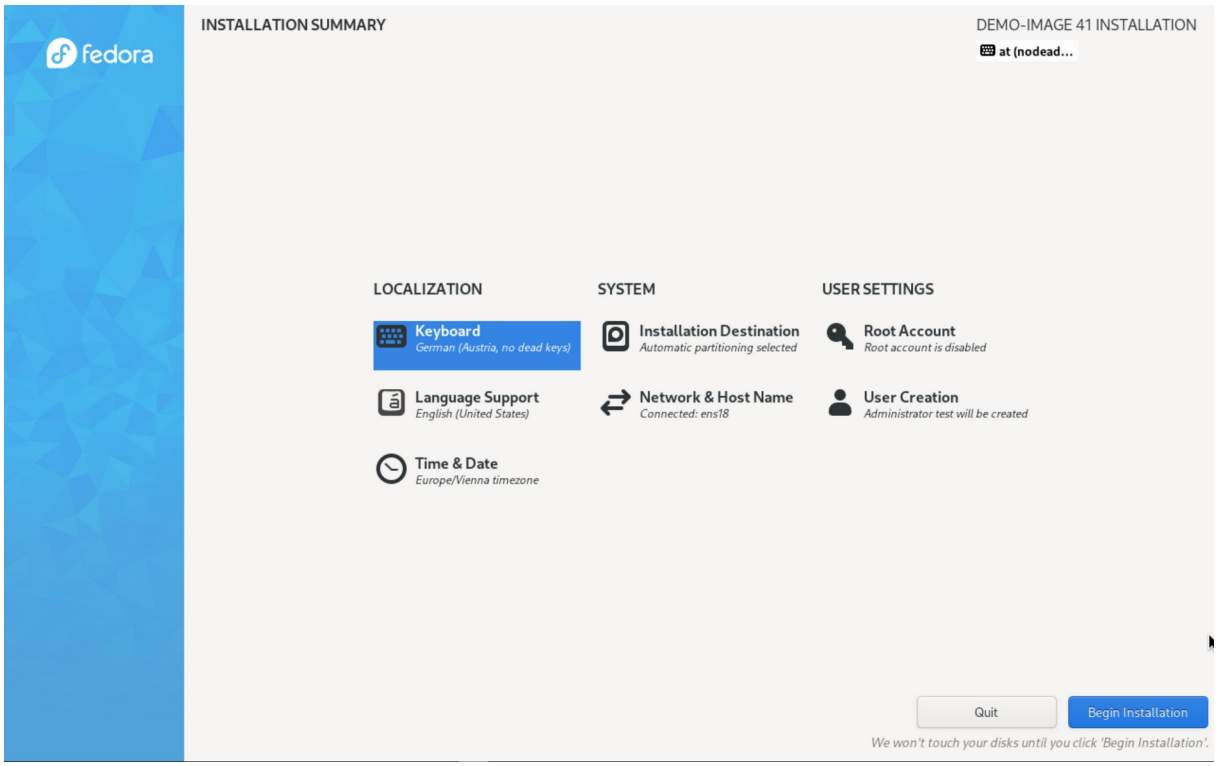
Add a ISO build job to the GitHub workflow (after bluebuild job in .github/workflows/build.yaml):

```yaml
iso:
  name: Build ISO
  needs: bluebuild
  runs-on: ubuntu-latest
  permissions:
    contents: read
    packages: write
    id-token: write
  steps:
    - name: Build
    uses: jasonn3/build-container-installer@main
    id: build
    with:
      arch: x86_64
      image_name: demo-image
      image_repo: ghcr.io/arminmiau
      image_tag: latest
      version: 41
      variant: kinoite
      iso_name: demo-image-latest_x86_64.iso
  - name: Upload as artifact
    id: upload
    uses: actions/upload-artifact@v4
    with:
      name: ${{ steps.build.outputs.iso_name }}
      path: |
        ${{ steps.build.outputs.iso_path }}
        ${{ steps.build.outputs.iso_path }}-CHECKSUM
      if-no-files-found: error
      retention-days: 0
      compression-level: 0
```

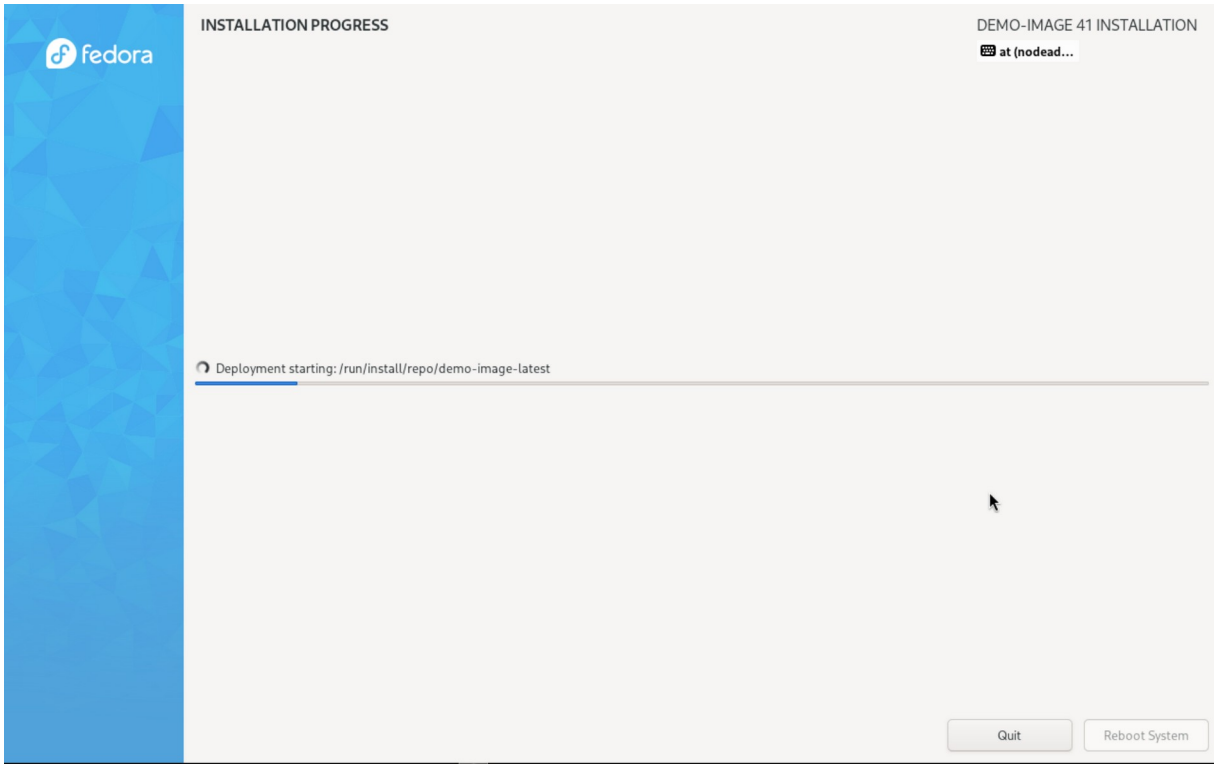
Commit and push the changes. The workflow will automatically build the new image and also an installation ISO. Once the ISO has been built you can test it out in a virtual machine.

3.2 Deployment

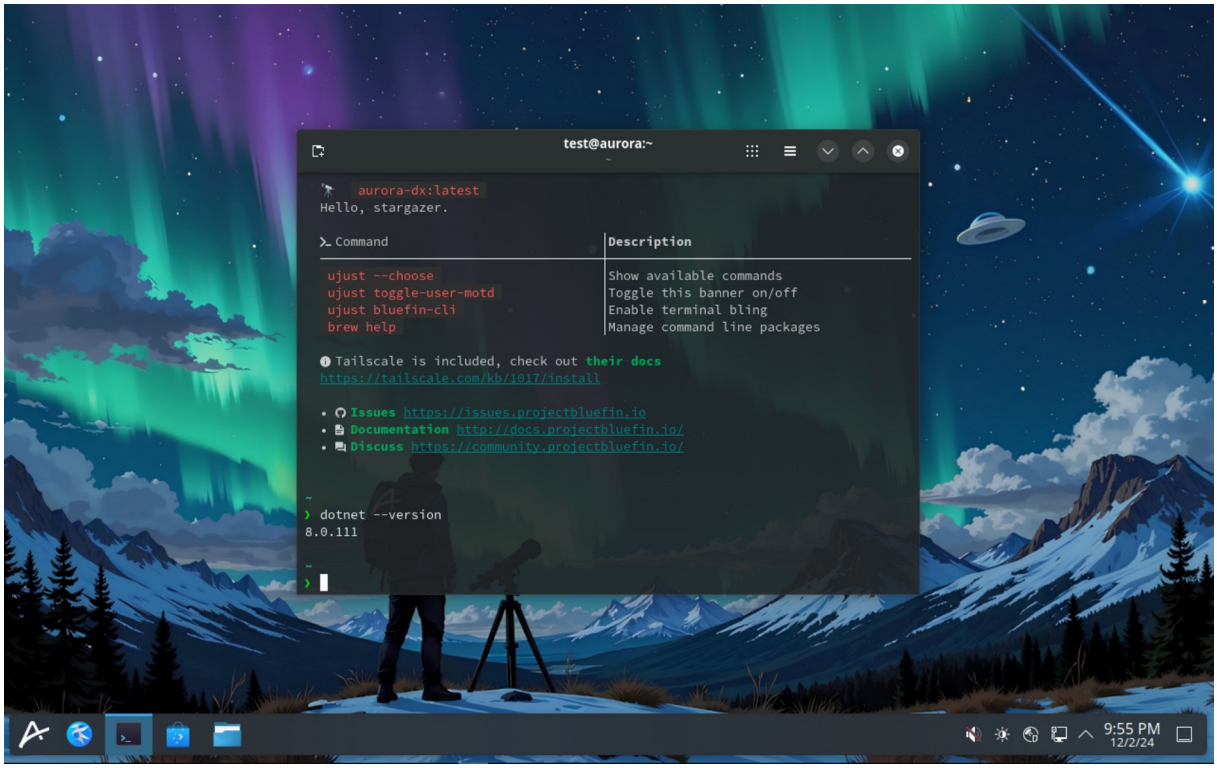
Due to the build and deploying process taking too long for the Lightning Talk, I have my virtual machine already prepared and this was the installation summary:



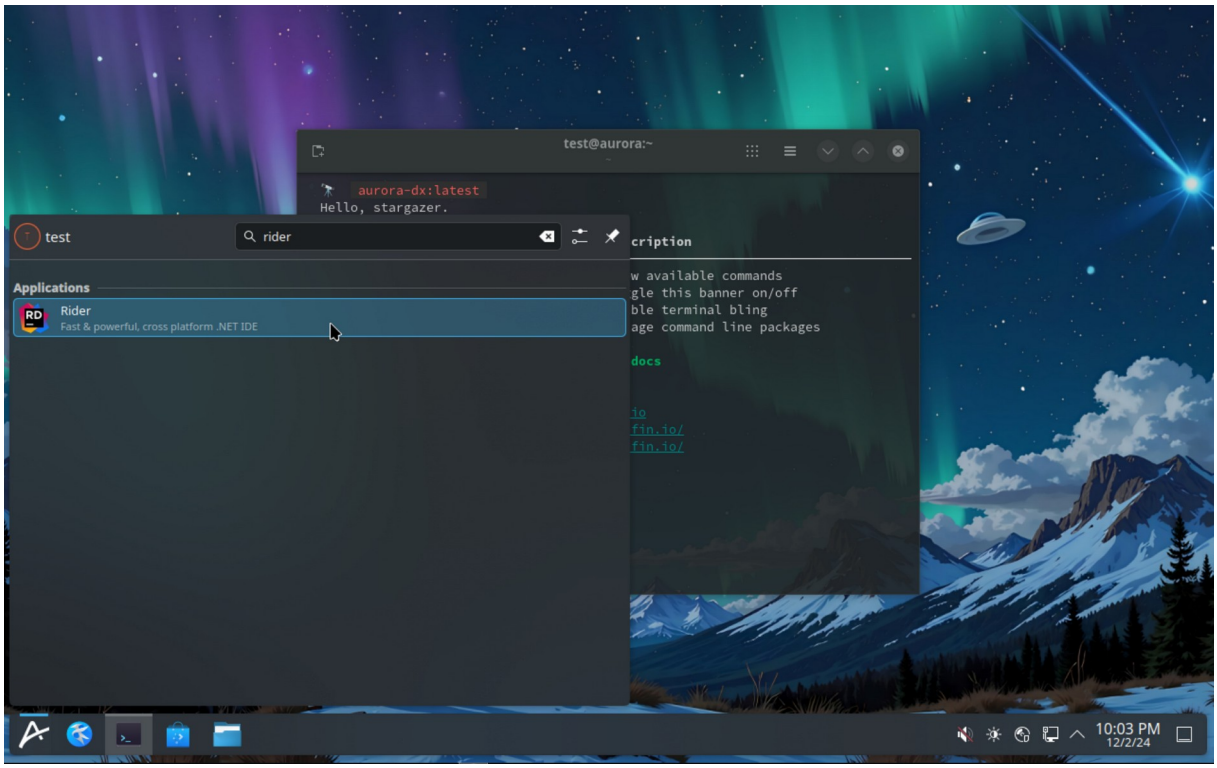
While installing you can see that it starts the deployment of the image:



After logging in we check the .NET version, which should be 8:

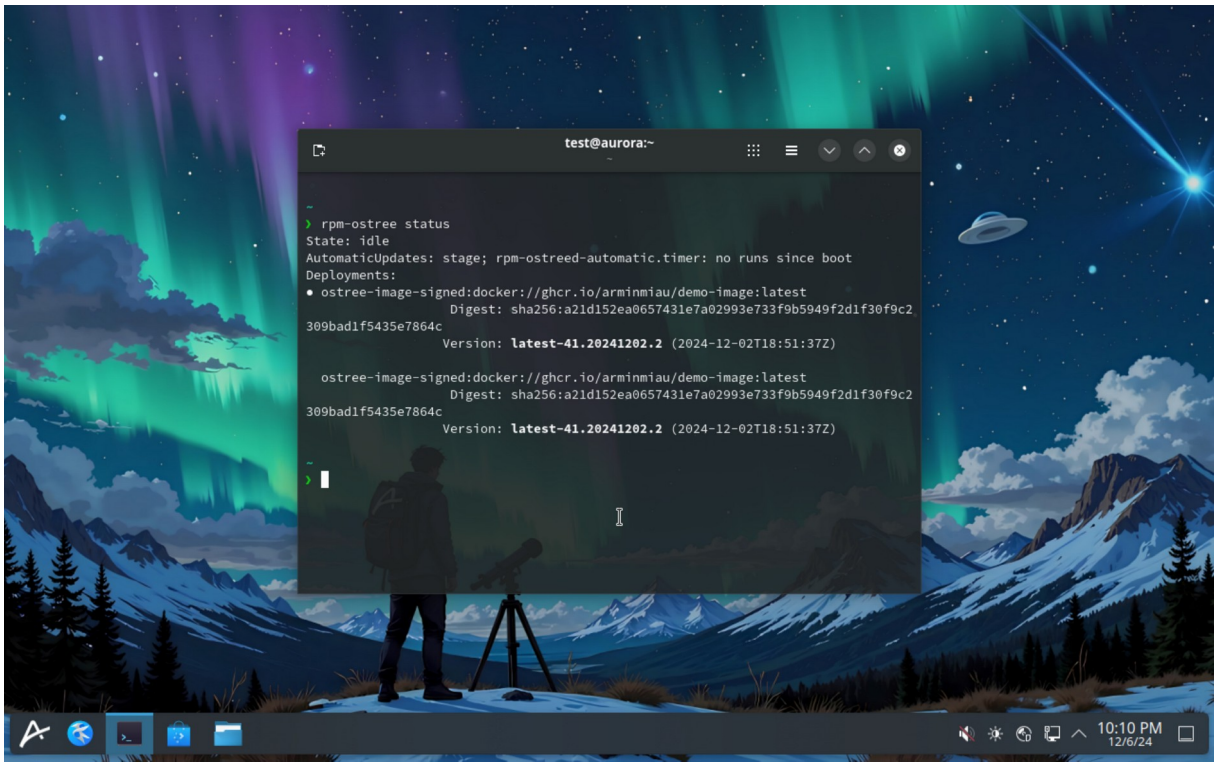


Then we can look for Rider, but at first login it will take a short time to appear as Flatpaks only get installed after first login, because they are sandboxed applications.

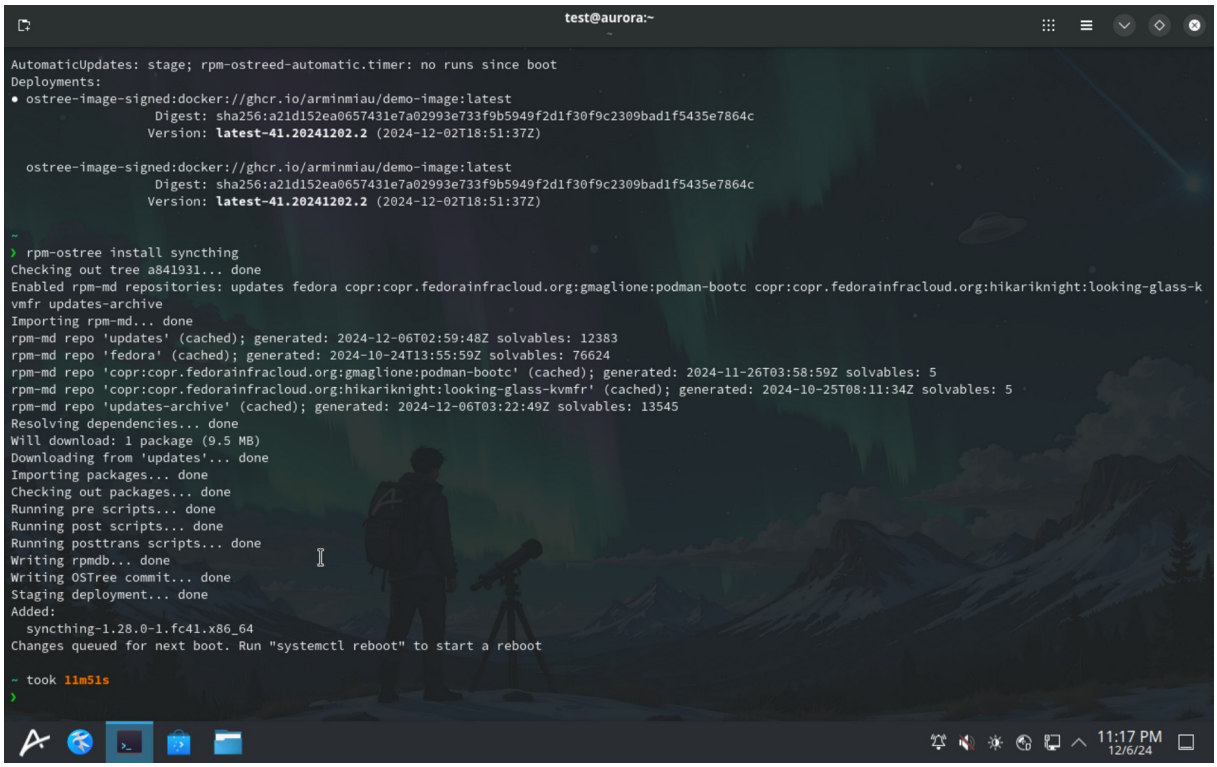


3.3 Layering and live-apply example

With rpm-ostree status we get a brief overview of the deployment status.

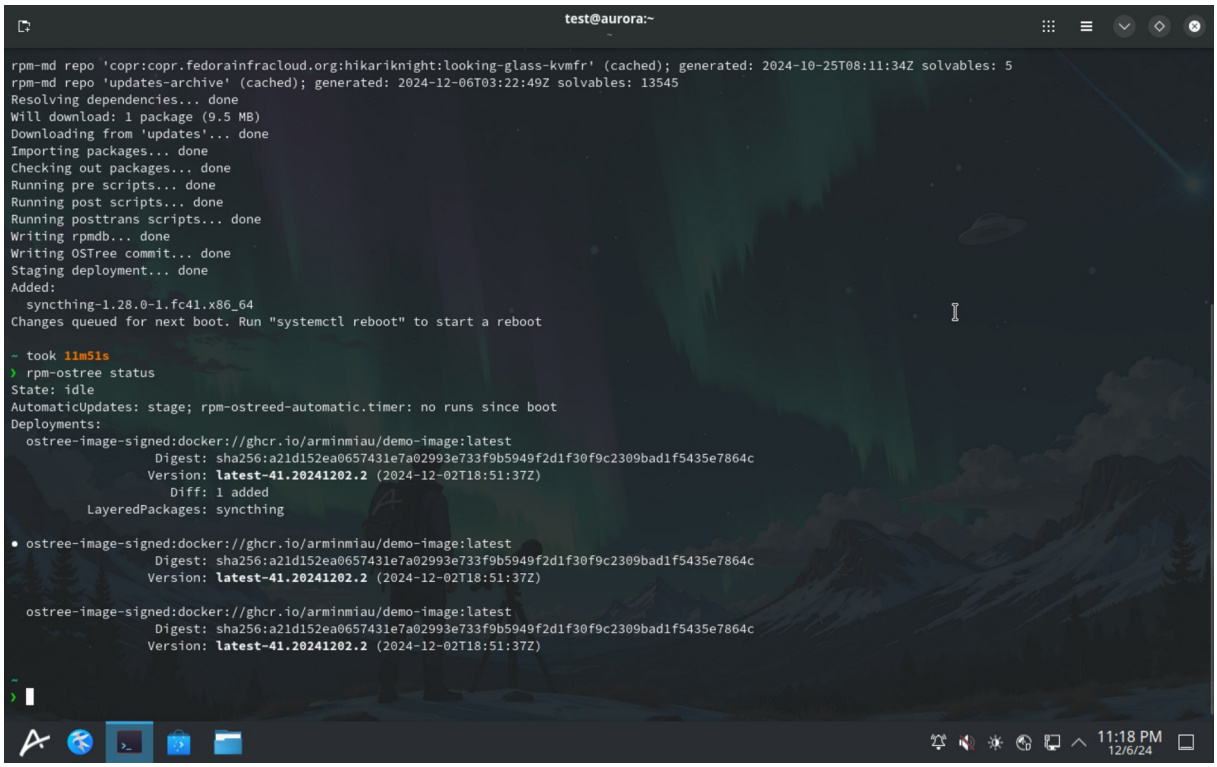


To layer a package we can use rpm-ostree install <package>. Let's install syncthing:

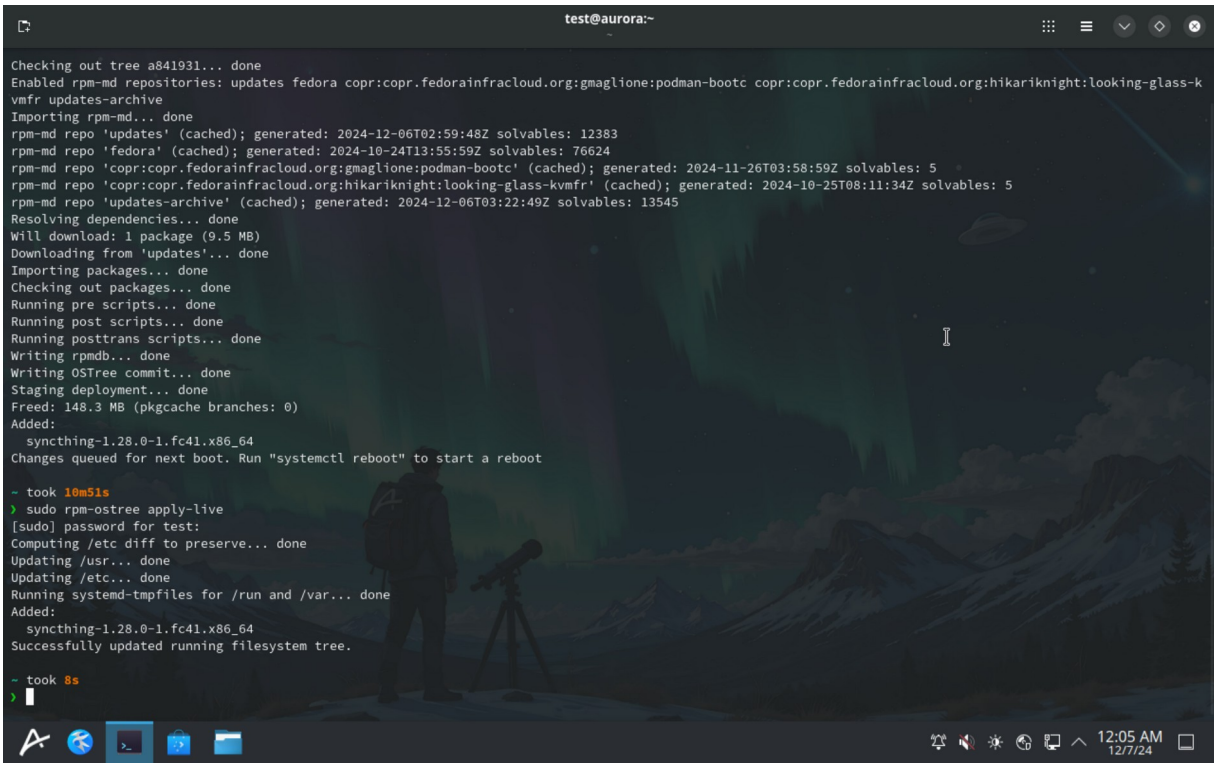


At the end the command will say: “Changes queued for next boot”.

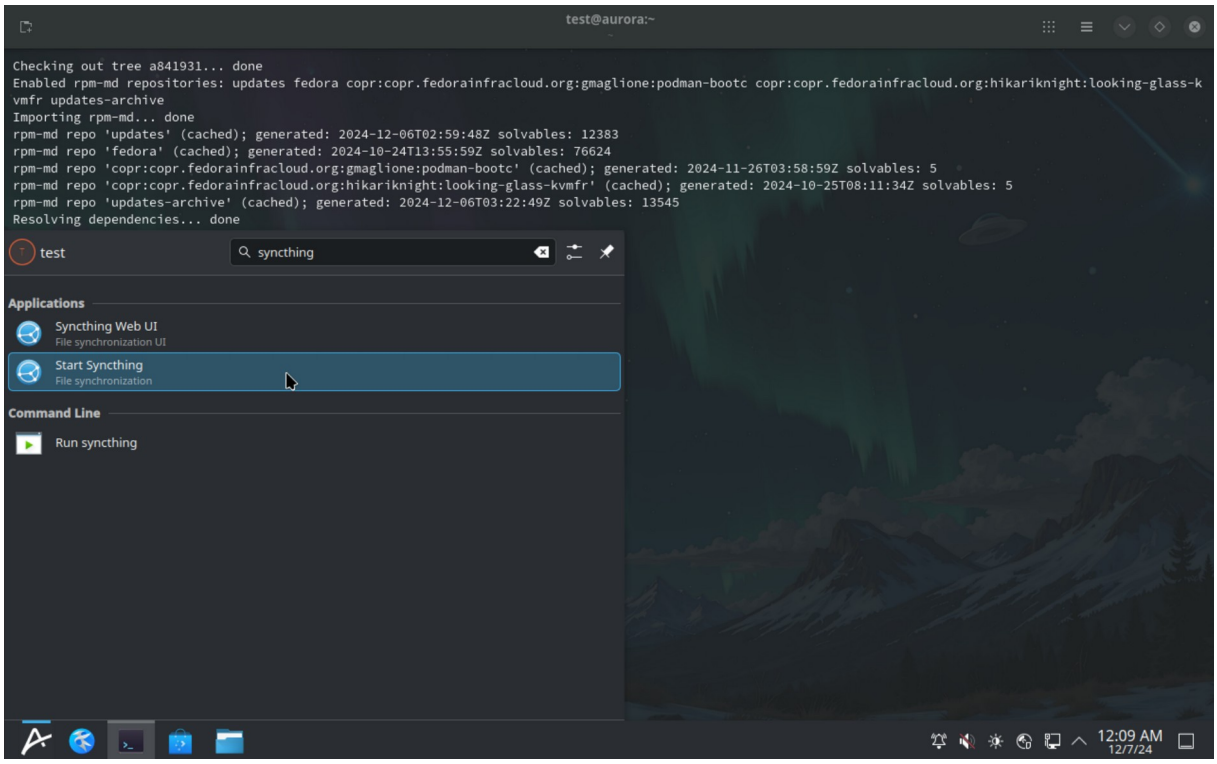
If we look at rpm-ostree status again, we see that the new change hasn’t been deployed yet.



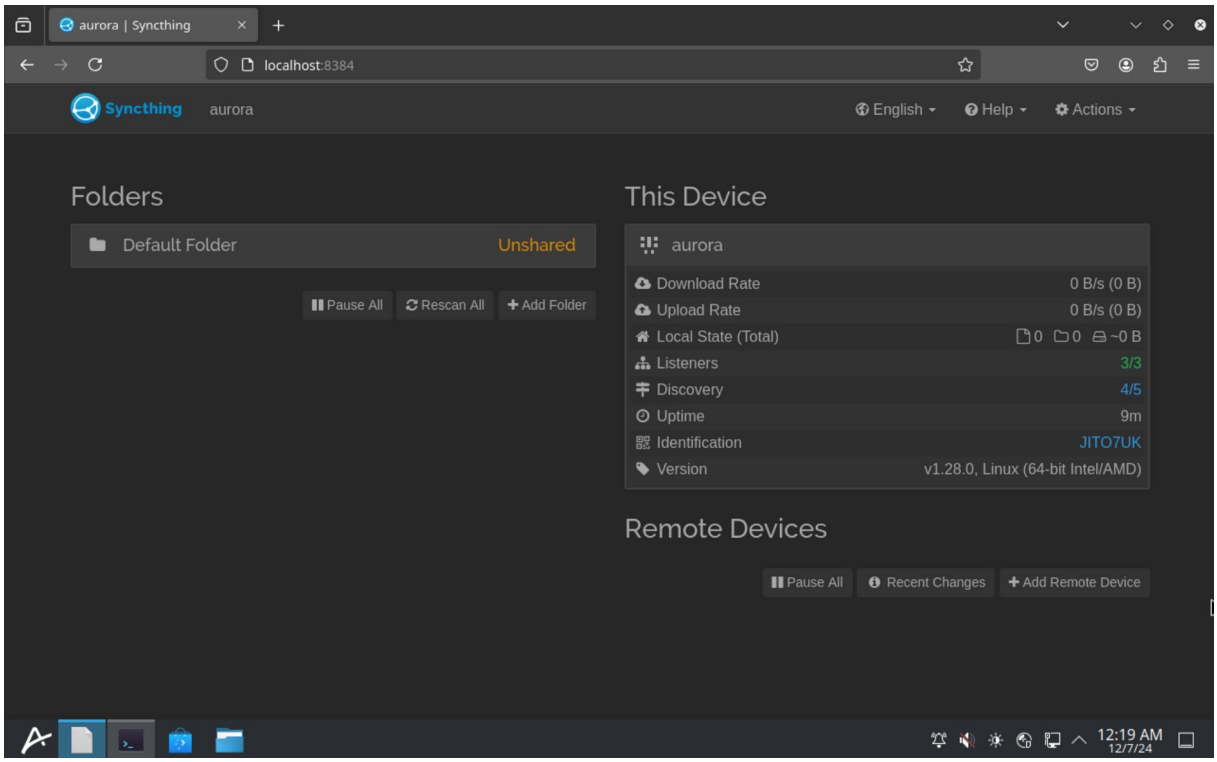
We can trigger a live deploy using sudo rpm-ostree apply-live. (Which might not always work, in my testing it only worked after layering one package and rebooting before adding syncthing and applying live.)



Once done we can start syncthing (the background process of it):



When the syncthing daemon has started we can look at the web ui.



4 Links

If you want to research more yourself here are some sources:

https://ostreedev.github.io/ostree/

https://coreos.github.io/rpm-ostree/

https://coreos.github.io/coreos-assembler/

https://fedoraproject.org/wiki/Initiatives/Fedora_bootc

https://fedoraproject.org/wiki/Changes/OstreeNativeContainer

https://fedoraproject.org/wiki/Changes/DNFAndBootcInImageModeFedora

https://fedoraproject.org/coreos/

https://universal-blue.org/