InfraMedia Manual

Operating system for media server appliances

Flussonic

Table of contents

1. Products

2. InfraMedia

2.0.1 InfraMedia

Flussonic InfraMedia (hereinafter InfraMedia) is the operating system of media server appliances. It is a Linux with an immutable root, shipped as a single image together with the application, plus a management daemon that configures the machine over HTTP.

Its purpose is to turn a server into a finished appliance that runs without administration. The machine is configured not by a person at a console but by the application the appliance exists for, through the configurator API.

How the system is built is described on the technical architecture page, installation on the installation page, updates and rollback on the firmware update page.

Why an immutable system

The system and executable files of InfraMedia cannot be changed while the machine is in service: the root is assembled from prepared squashfs images and mounted read-only. Everything else follows from that.

- **The machine in the rack is exactly what was shipped.** A system file cannot be modified or swapped, so the edits accumulated over years that nobody remembers cannot happen either. Two machines of the same version are the same machine.
- **The operating system and the application arrive as one delivery.** They are built and tested together, and the seam between them belongs to the vendor, not to operations.
- **Their versions are still separate.** The application lives in a layer of its own, so updating it without touching the system is routine, and the other way round as well.
- **An update lands whole or not at all.** The new version is placed beside the running one and switched on by a single line in the loader. There is no half-updated state, and a rollback is that same line put back.
- **No external repositories are needed.** In service the machine installs no packages: everything it has arrived in the image.

What the system is made of

Part	Purpose
Base layer	Linux with systemd, the management daemon, the web interface, kernel modules
Kernel and initrd	a kernel built for the target machine and the early boot script
Configurator	the management daemon: an HTTP API that configures the machine through systemd
Web interface	machine overview, interfaces, system, updates, power
Application layer	the media server (Mcaster, for example) as a layer of its own above the base
`postgres` layer	a shared component for the products that need a database

Two versions

The operating system carries its own version number and the server firmware carries another. The firmware is InfraMedia of one version plus the application of another; both are recorded inside the boot file and shown in the machine's interface. The operating system number does not move because the application has been updated.

Documentation map

This documentation will help you:

- Understand how the system is built
- Install the firmware on a server from a USB stick
- Update the firmware and roll back to the previous version

2.1 Technical architecture

InfraMedia is assembled as an image and is not modified on the machine. The root filesystem is composed of layers, each layer a squashfs image packed at build time. Everything the machine writes lives on separate partitions.

2.1.1 Firmware layers

Layer	Contents
Base	Linux with systemd, the management daemon, the web interface, kernel modules
`postgres`	the database server for the products that need one
Application	the media server: its executables and whatever they need beyond the base layer

The application layer is an increment over the base one: only the difference from the base layer goes into it. Libraries already present below are not carried up again.

2.1.2 Disk partitions

The installer lays the disk out in three partitions.

- **FIRMWARE** — the EFI boot partition: the boot files of the installed versions, the layer images, the loader menu.
- **SETTINGS** — the machine's settings: the configurator's store and the `/etc` overlay.
- **VAR** — mutable data: logs, the database directory, the application's working files.

The root stays read-only throughout. The home directory of `root` is a tmpfs and lives until the next reboot; whatever has to survive one belongs on the mutable data partition.

2.1.3 Boot

The machine boots through systemd-boot. The kernel, the initrd, the command line and the **manifest of the firmware contents** are glued into a single boot file. This is done for one property: choosing a boot entry chooses the kernel, the initrd and the list of layers together, and they cannot fall out of sync. Two firmware versions on the disk are two menu entries, and a rollback returns the whole set.

The boot goes like this.

1. UEFI hands control to the loader, which starts the boot file of the version selected in the menu.
2. The initrd reads the manifest, finds the layer images on the boot partition and **verifies their sha256** against the manifest.
3. The layers are attached and assembled into the root; the `/etc` overlay from the settings partition is attached on top.
4. Control passes to systemd, which brings up the management daemon and the application.

The initrd is a busybox script: it does not depend on the application and is built once per architecture.

2.1.4 The configurator

The configurator is the machine's management daemon. It listens on HTTP port 8090 and exposes the API: machine overview, network interfaces, machine name and time servers, firmware, reboot and power off. It also serves the web interface — there is no separate port for it.

It acts on the system by calling the standard programs and writing configuration files — `ip`, `networkctl`, `resolvectl`, `hostnamectl`, `timedatectl`, `systemctl` and the files under `/etc/systemd/`.

The settings store

The single source of truth about how the machine is configured is the configurator's settings store. The systemd files are rendered from it and are **never read back**.

Two practical rules follow.

- Editing the generated files by hand is pointless: the next render restores the stored value.
- The stored value and the actual state are different things. The stored value comes from the store, the actual state the configurator takes from the running system through the machine-readable output of the system utilities, and the two can differ.

The change session

Network changes are applied in a confirm-or-revert mode. A change opens a session, and while it is open the store is frozen as a whole. If the confirmation does not arrive in time, the store returns to the saved copy and the settings to the previous ones. There is never more than one open session.

The point of the mechanism is practical: a mistake in an interface setting no longer means losing access to the machine.

Limited mode

If the store cannot be read — the file is damaged or its version is newer than the daemon itself — the configurator falls into limited mode: read operations work, changes are rejected, nothing is rendered. The machine keeps running on what has already been rendered.

2.1.5 The hardware profile

The hardware profile describes the machines the firmware of this architecture runs on: the kernel configuration with their drivers and the rules that bring physical interfaces to canonical names by device path. The drivers of every supported machine are built into one kernel, and the same image boots on any of them.

Canonical names are assigned by the role of the interface — `manage` for management, `streaming` for streaming, with an ordinal number. Only an interface that has a canonical name can be configured through the API; a physical interface without one is visible but not configurable.

2.1.6 The web interface

The web interface ships in the base layer and is served by the daemon itself — it has to work on a machine where the application is absent or has not come up. It has five sections: **Overview**, **Interfaces**, **System**, **Updates**, **Power**.

The screens are built apart from the shell: they carry no theme, no router and no authentication, and they do not know where the API lives. That gives them two entrances — the application on the configurator's port, and a section inside the console of the application, which loads the screens from the machine itself. The operator configures the chassis where they already work with the media server.

2.1.7 Two versions

The operating system and the server firmware carry different numbers. The firmware is InfraMedia of one version plus the application of another; both are recorded in the manifest of the boot file. The application is built on the published parts of the operating system rather than rebuilding it.

3. Administrator

3.1 Starting

3.1.1 Installation

The firmware is installed on a server from a USB installer. The installer asks no questions and waits for no keypress: everything it needs it takes from a text file on the stick itself. A person writes the image, fills in the file, plugs the stick into the server and switches the power on.

The USB installer

The image carries the same firmware that will land on the machine's disk, and two partitions.

- **INSTALLER** — the boot partition with the firmware.
- **CONFIG** — an ordinary data partition holding the `install.txt` file. macOS and Windows show it by themselves; the boot partition they hide.

The image depends on no secrets: it is built with a template `install.txt`, and the file is filled in on the stick itself, in any editor. The line endings and the byte order mark that Windows and macOS add are discarded by the installer.

Write the image to the stick and replug it — the `CONFIG` volume appears.

The install.txt file

Open `install.txt` on the `CONFIG` volume and replace the placeholders in angle brackets.

Field	Meaning
`SALT`	the fleet salt: the secret the machine's `root` password is derived from. It stays on the stick and never reaches the disk
`EDIT_AUTH_LOGIN`	the account name for the application console and the chassis settings
`EDIT_AUTH_PASSWORD`	its password
`LICENSE_KEY`	the licence key of the application
`HOSTNAME`	the machine name
`INSTALL_DISK`	the disk to install on. Left out, the largest disk other than the stick itself is taken

A placeholder left in place is an empty value. Without the salt or without the password the installer refuses **before it touches the disk**.

Installing

Put the stick into the server and boot from it — in the boot menu it is the `UEFI: <stick>` entry.

From there the installer works on its own:

1. lays the machine's disk out in the FIRMWARE, SETTINGS and VAR partitions — **erasing it completely**;
2. copies the firmware onto it;
3. leaves the initial values file on the disk: the console account, the licence key and the hash of the `root` password;
4. prints the chosen disk and the machine's hardware address on the console;
5. powers the machine off.

The stick may stay in place: on the next boot the early script sees the same version on the disk and boots the disk. A stick with a different version installs that one afresh — a reinstall is done the same way.

The root password

The `root` password of each machine is derived from the fleet salt and the hardware address the installer printed. The salt lives only on the installation media; the machine gets a hash, and the password cannot be recovered from it.

Hence the property that matters: the password of a given machine is reproducible by whoever holds the media and the machine's identifier — with no database and no record of the machine itself.

First boot

After the installation switch the server on. The machine brings up the management daemon and then the application.

- The configurator and its web interface are on port **8090** of the machine.
- The application console is on its usual port.

The credentials come from the file the installer left behind. The password is turned into a hash when the settings store is first filled, and editing the file on an installed machine changes nothing.

> **Without credentials there is no way in**
>
> If the account fields in `install.txt` were left empty, the daemon rejects **any** password, not merely a wrong one. There are no credentials in the store and nowhere for them to come from: the machine has to be installed again.

3.2 Maintaining

3.2.1 Firmware update

An installed machine is updated without a USB stick. The update package is installed **beside** the running version rather than over it: settings, data and passwords survive the update, and the previous version stays on the boot partition for a rollback.

The machine downloads and installs nothing on its own. Installing an update is always an explicit request: from the interface, from the API, or by hand at the console.

The update package

The package is an archive of what goes onto the boot partition: the boot file of the new version, its layer images, the loader menu entry, the product description and a manifest of checksums. The package's own checksum is published next to it.

The update channel

The machine learns about new versions from the index of an update channel whose address is **compiled into the build**. It reads the index and lists the versions of its own product and its own architecture; the ones already on the partition are marked.

A build without a channel does not know about one at all, and can only be updated by URL or by file.

Installing

FROM THE WEB INTERFACE

The **Updates** section opens with two versions: the one running and the one that will boot next. Next to them is the reboot button — when the next version differs from the running one it is highlighted, and the screen says what exactly will change on reboot.

The channel index is read when the interface is opened: the menu item carries a mark when the channel holds a version that is not on the partition. The **Check for updates** button re-reads the index.

THROUGH THE API

The response comes back at once and the installation runs in the background.

Install a package from a file:

```
curl -u admin:password -H 'content-type: application/x-tar' \
    --data-binary @mcaster-<version>-amd64.update.tar http://<machine>:8090/system/firmware
```

Install a package from a URL:

```
curl -u admin:password -H 'content-type: application/json' \
    -d '{"url":"https://…/mcaster-<version>-amd64.update.tar"}' \
    http://<machine>:8090/system/firmware/install
```

See the progress of the operation, the installed versions and the one that boots next:

```
curl -u admin:password http://<machine>:8090/system/firmware
```

List the versions in the update channel:

```
curl -u admin:password http://<machine>:8090/system/firmware/available
```

Reboot into the new version:

```
curl -u admin:password -X POST http://<machine>:8090/system/reboot
```

What happens during an installation

1. The daemon unpacks the package onto the mutable data partition.
2. It verifies every file against the manifest of checksums, and the product, the architecture and the hardware profile against the running ones.
3. Only then it remounts the boot partition for writing and copies what is missing. Layer images shared with the versions already installed are not copied.
4. It points the loader's default entry at the new version.

After the installation the partition holds the running version and the new one. The rest are removed together with the layer images nobody needs any more.

The new version starts working only after a reboot — the installation does not reboot the machine by itself.

Rollback

A rollback is picking the previous version and rebooting:

```
curl -u admin:password -X POST http://<machine>:8090/system/firmware/<version>/select
curl -u admin:password -X POST http://<machine>:8090/system/reboot
```

Remove a version that is no longer needed:

```
curl -u admin:password -X DELETE http://<machine>:8090/system/firmware/<version>
```

The running version and the selected one cannot be removed.

When the daemon is not there

If the daemon has not come up or cannot be reached, the same path is walked by hand at the console as `root`. The `firmware-update` program lives in the base layer, depends only on the shell and the layer's utilities, and does exactly three things — download, put in place, switch — with the same checks the daemon makes. It removes nothing.

```
firmware-update install https://…/mcaster-<version>-amd64.update.tar   # or a path to a file
firmware-update list                    # versions on the partition: running and next
firmware-update select <version>        # boot the previous version next time
systemctl reboot
```

What is not there yet

There is no automatic rollback after a failed boot of a new version: if the machine does not come up, the default boot entry is put back by hand from the loader menu. Machines of the early builds carry a boot partition sized for one version, and an in-place update does not fit there — such machines are reinstalled from a USB stick once.