Alpine Linux v3.8 ShRx
======================

This is an image of Alpine Linux v3.8, which is configured as a Shell-receive (ShRx) appliance. When this image boots, it mounts a FAT filesystem that is shared between the host's `payload/build/` and the guest's `/mnt/vdb1`, and auto-executes `/mnt/vdb1/main.sh`.

Creating this media
===================

This media was created using a manual process. Therefore, the following steps should be taken to re-create this image.

1. Boot the default ISO -- this comes directly from the upstream source. In this case, it comes from Alpine Linux. This is with the repository, so we can _always_ rebuild.
2. Install the media to a tiny boot disk. Do not set up any network interfaces.
3. Insert a cron entry to, upon boot, mount the FAT filesystem. This entry will auto-delete itself after mounting that FS, and execute `/mnt/build/main.sh`.
4. Poweroff the machine, and copy the image. This is the 'release'.

Running QEMU to create this media
=================================

Create the boot media:

```
       `make os/boot.qcow2`
```

Attaching a payload
===================

This project requires a `payload/` directory, to contain a set of files for running upon the machine's first, and only first, boot. This directory must contain another, called `build/`, which shares its files with the virtual guest's filesystem.

`payload/` should also contain a Makefile, which has `clean` and `all` targets defined. shrx-alpine makes use of these during its own build process; one may call `make all` from shrx-alpine, and `payload/Makefile` will also get called. Once all is said and done, `payload/build/main.sh` must exist. This will be the entrypoint to the virtual machine.

Running the payload installer
=============================

The first time this appliance boots, it expects there to be a /dev/vdb1 partition. This must be supplied to QEMU like a file, using a VirtIO interface. The following example assumes a `payload/build/` directory, which contains a `build/main.sh` executable file.

To build the "application-installed" image, run `make`. This will build the `release/boot.qcow2` target, which will copy `os/boot.qcow2` to `release/boot.qcow2`. Then, the VM will be started, and the cron line to run `main.sh` will go, then the cron line will be removed, then the machine will poweroff.

At this time, the files in `release/` are ready to use for production.

Building the release images
===========================

`make release`, which is also aliased by `make all`.

Running the release images
==========================

The images found in `release/` can be run with QEMU. There is a makefile target called `run` that will run them. However, it should be noted that once the `run` target activates, the images are no longer considered release-worthy, as some bits may be twiddled.

Production images should be exported as soon as a `make release` target is run.

Running `make run-release` will take it one step further, and run the image locally in the terminal. Running `make irun-release` will do the same thing, but with QEMU's '-serial mon:stdio' set, which redirects C-c to the guest.

`main.sh` notes

This file, to be in the `payload/build/` directory, should perform the following tasks to configure an application server.

x. Configure network, DNS, logging destination
x. Add application service to /etc/init.d and start at boot
x. Produce boot config file (for virtio, qemu, or proxmox)

This script will only run once, on the first boot. It does so via the cron mechanism, which self-deletes after running successfully.

This should return a non-zero exit code if anything went wrong.

Why a manual process?

A manual process is used to create the release image, because it's very hard to script commands inside the OS. There are many ways to skin a cat, and this provides a very quick turnaround (debugging `expect` scripts is time-consuming) and a minimal, solid platform for letting our application code configure the guest.

It would be great if we had an automated procedure for making this kind of image. However, once we have a master boot image, it doesn't matter any more how we got it.

It could be envisioned a similar process taking place for other guest OSes. It may not be possible to script all of them.

References

1. https://en.wikibooks.org/wiki/QEMU/Devices/Storage