BVS1 · Winter Semester 2026/2027 · TH Köln · ICCT

Lab 0 — Common Development Environment

Except where otherwise noted, © 2026 Stavros Kousidis · CC BY-SA 4.0

Planned time: 1–2 hours
Revision: 2026-10-30

Purpose

Lab 0 establishes the common technical baseline for the practical work.

It is **not yet an operating-systems implementation lab**. Its purpose is simple: before the first exercise that uses the environment, everyone must have the same supported development environment and one successful end-to-end run of `Hello BVS1`.

The reference setup is:

```
physical host
  Linux / macOS / Windows
        ↓
     Multipass
        ↓
Ubuntu 26.04 development VM: bvs1
  Git / Make / RV64 cross-toolchain / QEMU
        ↓
emulated RV64 target machine
        ↓
Hello BVS1
```

The **Ubuntu VM is the development host**. The **emulated RV64 machine is the target**.

Before the First Session

Bring your own laptop with:

- administrator rights,
- working internet access,
- enough free resources for a VM with approximately 2 CPU cores, 4 GB RAM, and 16 GB disk space.

If possible, install **Canonical Multipass** before the first practical session.

You do **not** need:

- a GitHub account,
- a RISC-V compiler or QEMU installed on the physical host,
- a pre-cloned course repository,
- prior operating-systems development experience.

We will create and verify the course environment together during Lab 0.

1. Install and Verify Multipass

Install Canonical Multipass on the **physical host**.

- **Ubuntu:** open a terminal and install Multipass with:

  ```
  sudo snap install multipass
  ```

- **Other Linux distributions:** Multipass is distributed as a Snap. If `snapd` is not already available, install and enable it using the instructions for your distribution, then install Multipass with:

  ```
  sudo snap install multipass
  ```

  For distribution-specific `snapd` setup, see the Snapcraft installation guide.

- **macOS:** Multipass requires macOS 13.3 or later. If your Mac runs an older version, update macOS if your hardware supports it. If that is not possible, use another computer with a supported operating system. After installing Multipass, open the macOS **Terminal** application.

- **Windows:** install Multipass, then open **Windows Terminal** or **PowerShell**. If you encounter problems with PowerShell, try the classic **Command Prompt** (`cmd.exe`).

  If Multipass cannot create or start a VM because of Windows virtualization or driver problems, try the **Multipass 1.17 release candidate** from the official GitHub release page. This release candidate introduces a new Windows virtualization backend based on the Windows Host Compute System (HCS). Use this only as a troubleshooting alternative if the regular Multipass installation does not work.

The commands in this section are entered in that normal host terminal. Multipass does not require a separate course terminal application.

Verify:

```
multipass version
multipass find
```

The image list must contain Ubuntu `26.04`.

Linux Alternative: Distrobox

On a **Linux** physical host you may use **Distrobox** instead of Multipass. Distrobox runs Ubuntu 26.04 as a container using Podman or Docker. Advantages:

- **Lightweight:** there is no separate VM. The container shares the host's Linux kernel, needs less memory and disk space, and starts within seconds.
- **No Snap required:** Podman or Docker are packaged by most distributions.

- **Shared home directory:** your home directory is shared with the host. You can edit course files with your usual host editor or IDE, and the files remain on the host even if the container is removed.
- **Host integration:** your normal terminal, clipboard, and graphical applications keep working.

Keep the difference in mind: a container is **not a VM**. It shares the host kernel, and commands inside it act on your real home directory. The reference environment remains the Multipass VM. If a problem cannot be resolved in Distrobox, you may be asked to reproduce it in Multipass.

If you choose Distrobox, replace Sections 2–4 with:

```
distrobox create --name bvs1 \
  --image docker.io/library/ubuntu:26.04
distrobox enter bvs1
```

Inside the container, install the course packages listed in `bvs1-cloud-init.yaml`:

```
sudo apt update
sudo apt install -y git make gcc \
  gcc-riscv64-unknown-elf binutils-riscv64-unknown-elf \
  qemu-system-riscv gdb-multiarch file tree nano vim less
```

Verify with `cat /etc/os-release | grep PRETTY_NAME` that you are in Ubuntu 26.04 LTS, then continue with Section 5. The `cloud-init` and `hostname` checks of the Multipass path do not apply; `make check` in Section 6 remains the environment gate.

2. Download the Provisioning File

Download the course-provided file from ILU:

```
bvs1-cloud-init.yaml
```

ILU appends `.sec` to the downloaded file name, so the file arrives as `bvs1-cloud-init.yaml.sec`. Rename it back to `bvs1-cloud-init.yaml` before using it:

- **Linux / macOS:** `mv bvs1-cloud-init.yaml.sec bvs1-cloud-init.yaml`
- **Windows PowerShell:** `Rename-Item bvs1-cloud-init.yaml.sec bvs1-cloud-init.yaml`

Windows Explorer hides known file extensions by default, so the file may look correctly named although it is not. Check the name in the terminal.

On the **physical host**, change to the directory containing the file and verify it:

```
ls -l bvs1-cloud-init.yaml
```

Do not continue until the file is shown. The `--cloud-init` argument is a file path; running the next command from the wrong directory will fail.

3. Create the Course VM

Still on the **physical host**:

```
multipass launch 26.04 --name bvs1 --cpus 2 \
  --memory 4G --disk 16G \
  --cloud-init bvs1-cloud-init.yaml
```

A successful launch ends with:

```
Launched: bvs1
```

Package installation may continue briefly after the VM has been created.

4. Enter and Verify the Development VM

On the **physical host**:

```
multipass shell bvs1
```

If this fails, **stop here**. Do not continue with the following commands on the physical host.

Inside the VM, wait for provisioning:

```
cloud-init status --wait
```

The result must be:

```
status: done
```

Now verify where you are:

```
hostname
cat /etc/os-release | grep PRETTY_NAME
uname -m
```

The hostname must be `bvs1`, and the operating system must be Ubuntu 26.04 LTS.

The value reported by `uname -m` describes the **development VM**. It may differ between computers. Our teaching target is still RV64.

5. Configure Git and Clone the Course Code

Still **inside the `bvs1` VM**, configure the identity used for your local commits:

```
git config --global user.name "Your Name"
git config --global user.email "your-email@example.org"
```

This does not create or connect a GitHub account.

Clone the public course-code repository:

```
git clone \
  https://github.com/stavroskousidis/teaching-bvs1-code.git
```

```
cd teaching-bvs1-code
```

This clone is your local course repository for the semester.

Keep it. Do not delete and re-clone it between lab sessions. You do not need a fork or a personal remote repository, and you do not push your work to GitHub. Do not pull new course states unless explicitly instructed.

6. Pass the Environment Gate

Still **inside the `bvs1` VM** and inside `~/teaching-bvs1-code`, run:

```
make check
```

The required result is:

```
Checking BVS1 environment...
Environment OK.
```

This verifies the expected RV64 toolchain, binary utilities, QEMU, and the QEMU `virt` machine. It does not install or modify packages.

If the check fails, stop and use the troubleshooting section below.

7. Build and Run Hello BVS1

Build the starter program:

```
make
```

Then run it on the emulated RV64 target:

```
make run
```

The important guest output is:

```
Hello BVS1
```

No guest prompt follows. This is expected: the target contains **no guest operating system and no shell**. QEMU is still running in the foreground of the development VM.

Terminate QEMU with:

```
Ctrl-a
x
```

Press **Ctrl-a**, release the keys, then press **x**.

You should return to a prompt similar to:

```
ubuntu@bvs1:~/teaching-bvs1-code$
```

The absence of a guest shell is not a defect. It is our starting point.

Lab 0 Complete

Lab 0 is complete when all of the following are true:

- `cloud-init status --wait` reports `status: done`;
- `hostname` reports `bvs1`;
- `make check` reports `Environment OK.`;
- `make` builds the RV64 program;
- `make run` prints `Hello BVS1`;
- **Ctrl-a**, then **x** returns you to the development-VM shell;
- `git status` works inside `~/teaching-bvs1-code`.

At this point the environment is established. Lecture 2 and Exercise 1 investigate what the development host, target ISA, cross-toolchain, and QEMU actually do.

After Lab 0

Lab 0 is complete once the checklist above succeeds.

Keep the VM Updated

Update the packages in your development VM regularly, for example once a week. Inside the `bvs1` VM, run:

```
sudo apt update
sudo apt upgrade
```

If the upgrade reports that a restart is required, leave the VM and restart it from the physical host:

```
multipass restart bvs1
```

Then run `make check` inside `~/teaching-bvs1-code` again. Do not upgrade to a newer Ubuntu release (`do-release-upgrade`); the course environment is Ubuntu 26.04 LTS. Avoid updating immediately before a checkpoint appointment. With Distrobox, run the same `apt` commands inside the container.

For everyday VM commands, backup, and troubleshooting, use the separate **Development Environment** reference handout.

Conceptual questions about the physical host, development VM, QEMU, the RV64 target, virtualization, and emulation are handled in Exercise 1.