# Computer Lab Infrastructure at BITL

by BITL.LV, 2019-06-05

## Overview

This document describes high-level options and supporting technical solutions to ensure access to the computer labs during the BITL (see http://www.bitl.lv/) classes and related activities such as “hakatoni” (1-2 day prototyping by a small team) and other team projects. This document includes:

* Choices of the lab setup, their comparison, advantages and related costs (**Section 1**)
* Actual requirements/scenarios how to set up the computer lab (**Section 2**)
* Server specifications and other hardware requirements (**Section 3**)
* DevOps work (initial setup), monitoring and troubleshooting guidelines (**Section 4**)

**How to use this guide:**

* Instructor - Read the requirements section of this document. Does it address everything that you will need during your class, homeworks and exams? (**Section 2**)
* BITL Program Manager - Read the alternatives, the actual requirements and related costs. Also the DevOps section and troubleshooting. (**Sections 1,2,3,4**)
* IT Administrator - Look at the DevOps and monitoring part.(**Section 4**)

## Section 1: Choices of Computer Lab Setup and Use

### Choice 1: Physical Hardware or Virtualization

IT education typically uses some combination of these 4 approaches:

|  |  |
| --- | --- |
| **Physical Computer Lab**  (1) Simple to access and to use  (2) Desktop is visible right away, no need to use VNC or RDP.  (3) No virtualization overhead in performance.  **BUT** (4) Restricts you to one operating system only.  (5) Clean starting state is difficult to ensure.  (6) Computer class resources may be underutilized.  (7) Remote access is not easy. | **Use cloud services for hosting only**  (1) Many “cloud hosting” solutions in the market:  Azure (Microsoft), AWS (Amazon), OVH (a French company) and many more.  (2) Sizing and pricing plans for all types of needs.  (3) Networking, service failures, upgrades and maintenance handled by the service provider.  **BUT**  (4) Managing hosted machines is done differently for each provider. Embotics/vCommander and other proprietary solutions try to sort this out.  (5) May need tricky permissions (to avoid “managing” something by accident.) |
| **University owned virtual infrastructure**  (1) Can run many virtual guest images (one machine or set of machines per class).  (2) Can power them on and off as needed.  (3) Easier to spin off new guest images.  (4) Can access guests remotely.  **BUT**  (5) Student still needs a “client workstation” to connect to the virtual machines.  (6) Connecting to the lab not always intuitive (run SSH tunnel, run VNC or RDP clients explicitly).  (7) Virtual servers have to be set up and administered by university people. | **Use cloud services for hosting+management**  Some providers offer hosting+managing guest images. Cloudshare, ReadyTech, etc.  (1) Very intuitive to set up and to connect (RDP clients often embedded into browsers),  (2) Trainers can monitor every student, see desktops, get useful reports.  (3) We have 24x7 customer support, if something breaks.  **BUT**  (4) Hosting lab environments in this way is expensive (20-100 USD per environment per day). |

**Things to consider:**

* Are the students using their own devices during our classes?
* Do they need to connect from home?
* Are we concerned about students who have their own laptops, but do not know how to set up their OS, install IDE and use the services needed for our classes. If student fails, is there somebody who can assist?
* Are there “spikes” in the lab utilization? (E.g. one week with very intensive use of computers)? In these situations hosting solutions on 3rd party servers could be justified.
* Do you have dedicated IT Administrator staff - people who like to manage virtual servers?

### Choice 2: Network access to the lab machines

Physical computer lab typically is a segment of a corporate LAN. It may be further restricted from other LANs (more firewalls, or vice versa - more permissive, because it is “training network”, not “production network”).

If you have virtual guest images in your lab, there are several options how to connect:

1. **Run a VPN to a server in the remote lab.** Start using IP addresses from the remote lab network on your local machine.
2. **SSH tunnel to a server in the remote lab.** Different students may use different SSH ports. Students can use PuTTY or other SSH client. After that any protocol (SSH, RDP, HTTP) could be tunneled over that connection; the IP address could still be your localhost. Not just the SSH terminal sessions, but various other protocols (including remote desktops, HTTP) can be tunneled.
3. **Use vSphere or a similar virtualization client software.** You can open consoles directly from the vSphere. This option gives you high level of control, even when guest machines have misconfigured IPs, are stopped or suspended. Not safe to hand out to students as they might interfere with other peoples’ machines. Better suited for instructors or other responsible people.
4. **Fancy remote lab management tools.** Vendors such as Embotics, Cloudshare make the connection setup more intuitive - you can open a link in a browser, log in and get browser links or tabs that open screens to various machines in your remote environment. (Under the hood they are still using browser plugins that do VPN or something similar.)
5. **Bastion/Landing machine.** If there are multiple machines in the student lab, then a “bastion” machine is often used. It is the only one reachable from an external network (any method described above can be used). After you have the landing desktop open, you can click on further RDP or SSH links to connect to other machines, servers, etc. You get “nested RDP” - i.e. an RDP session to the landing desktop lets you open a nested RDP session inside itself. Nested RDPs are sufficiently easy to use.

**Things to consider:**

* Are students intimidated by opening SSH tunnels, VPN connections or something similar - before they can use the computer lab? Do they have the necessary tools installed and do they have networking experience?
* Are any of the protocols (SSH tunnels or VPNs, strange port numbers) restricted in the university network? Or maybe the university (default/WiFi) network is too slow to run multiple RDP sessions simultaneously?
* Are there risks of lab machine misuse, for example, to send unwanted or infected emails, stage DoS attacks or to misrepresent one’s identity in the Internet? If so, some monitoring of egress traffic could be needed or even ACLs that allow to access only course-related sites.

### Choice 3: Remote desktop access

1. **Physical machine:** In a physical computer lab each computer is connected to a monitor directly. One may need additional solution (hardware switch?) to enable quick connection of a computer to a multimedia projector.
2. **RDP session:** If student uses Windows workstation, s/he can run mstsc.exe and use a Windows Remote desktop protocol. It is also possible to connect to remote networks in this way (e.g. **Start > Administrative Tools > Remote Desktop Services > Remote Desktop Session Host Configuration**).
3. **VNC client:** This approach is similar to RDP, it may be slightly trickier to configure, but works on both Windows and Linux machines.
4. **RDP embedded in a browser:** This is intuitive and convenient for the user - each machine can open a desktop in a separate browser tab. Hosting services like Cloudshare, ReadyTech, vCommander provide various flavors of this. Free solutions supporting this may exist, but have not been tested so far.

**Things to consider:**

* Which OS is installed on the local workstations used to access the remote lab? For Windows workstations, an RDP client is typically used (together with PuTTY sessions to connect to the remote Linux hosts). If the workstation itself uses Linux, VNC seems to be the only option.
* Typically computer programming does not require high screen resolution or image accuracy. Still, the remote desktop, if not configured properly, may have noticeably worse user experience (compared to large monitors connected to a physical machine). Screen resolutions and refresh rates should be tested before they are deployed.
* Some programming courses require extensive copy-pasting from a file that is on student’s workstation into the virtual lab (chunks of computer code, chunks of configuration files, etc.). RDP typically supports copy-pasting things between the local and the remote screens. It may also support intuitive “drag-and-drop” transfer of files. Some other solutions are not as flexible - in this case all the information student needs should be pre-loaded on the remote machine (or accessible from a Web page to be open remotely).

### Choice 4: Configuring the lab starting state

Every instructor has certain preferences - what OS, what tools and what configuration they will use.

There are various ways to ensure this:

1. **Install everything manually.** Configure permissions that prevent unauthorized users to change the state of crucial components too much. This is often the “lazy” choice, but it may be either too permissive (when lab machines get infected or receive unwanted software installations) or too restrictive (when people cannot complete certain tasks because they lack permissions).
2. **Clone physical lab machines regularly.** You would need consistent hardware (e.g. the same disk and external drivers for each lab machine), then you can have a “clean factory image”. In this case students get used to backup all their files from the lab machines (for example, once every month), and the lab machine returns to its default starting state. If there is a large number of machines to re-image, this step is tedious.
3. **Use virtual OVA images (e.g. on a VMWare ESXi server).** Instructor may create his/her own OVA image (on a Virtual Box or other simple virtualization host), it is then copied into the remote lab, and used to initialize virtual guests.   
   This approach is fast and flexible - it allows to use tens of initial images for various uses, you can also use free software licenses for the 30-day “trial period”. The OVA images may be slow to copy (they are typically tens of GiB), and running everything as a virtual guest needs good hardware.
4. **Use Ansible playbooks (e.g. on a Docker host).** Besides Ansible there are other configuration automation tools (Salt, Puppet, etc.). This approach is also flexible, it is often scheduled or run from a continuous integration tool like Jenkins. Unlike OVA it is not tied to any specific virtualization or hosting technology, any typical Linux setup operation can be automated in this way. One downside to Ansible, compared to OVA is that environment building takes time (a typical Linux server is created in some 10-30 minutes rather than instantly). If Ansible playbooks (initialization scripts) are not robust, the build may also fail, if something unexpected happens.

### Choice 5: Submitting work from the lab

1. **Uploading individual files or archives to a “grading server”.** This is used by many universities - it is a low tech approach, which can still have some integrity checks. It does not have the downsides of Email communication (email misclassified as junk, not delivered to the instructor), it can provide a feedback to the student (MD5 hash, time-stamp - to ensure that the exercise was submitted on time).
2. **Public Git service (GitHub.com).** All the configuration files created by students are checked into a version control system, if necessary, the version to be graded can be tagged. This approach gives good insight into the “open source community”, and students can build their code portfolio - which may help to find jobs in IT. Public Git may be problematic, if we want to avoid academic dishonesty - because everyone sees files created by everyone else.
3. **University owned Git service.** This allows more fine-grained access control. For example, only students in the same team (plus their instructor) can see the work. On the other hand, this solution has to be maintained by the university - costs may be higher.
4. **The configuration is graded directly.** This approach can be used, if students create all the relevant files on the virtual lab machines. When they are done, a grading script can be triggered from a build automation tool (Ant, Maven, some Python script, Jenkins). This approach is more flexible than Git (or any other file version control), because in this case we can look not just at the files, but we can test the solution locally - exactly as experienced by the student. We can check what changes s/he has done to the overall configuration of the machine(s) involved. This is sometimes used in certification exams.

## Section 2: Requirements for the BITL Program

This part needs more input from the BITL instructors.

### Scenario 1: (Keeping things simple) Physical workstations run everything

* Each workstation holds a (clone-able) image of MS Windows or Linux OS.
* Pre-installed browsers, PDF and media file viewers, plaintext editors and file managers, Integrated Development Environments (IDEs), database clients and some other tools. (Plus - there are instructions how students can install similar software on their own machines.)
* Workstations can have their own development stack (local database server,

This approach is typically used in computer labs in Latvian universities. It is easy to set up. It may have more expensive maintenance (anything you do on a physical workstation is hard to automate), it may give somewhat inconsistent user experience (some lab machines may start to behave strangely, re-imaging them every time is a slow a tedious activity). It also may require that students do all the assignments from the lab itself. (Or, perhaps, they can try to emulate the development environment on their own laptops, and deal with inevitable inconsistencies when they try to integrate with other students’ work.)

### Scenario 2: Use some client tools locally, combine with remote/virtual servers

This approach emulates how most software development teams are working.

In this case we keep to a reasonable minimum the client tools installed on a user workstation:

* Browsers
* PuTTY and SCP to log into various remote servers
* Plaintext editors and IDEs
* Git or other version control clients

All the development stack is located on external servers (typically virtual guests) accessible either locally or remotely. This approach encourages remote working - one can do the assignment in every place with an Internet connection.

### Scenario 3: Run everything remotely

In this case even the student workstations are virtual guests. Only the browsers and remote desktop clients are needed locally; all the development tools are remote.

This approach provides high degree of flexibility and control. For example, instructor has a choice to have several “starting states” for the student workstation - s/he knows exactly what is installed there, can monitor everything.

This approach requires additional resources (and remote desktops do not always have crisp desktop picture).

## Section 3: Technical Specification

TBD.   
**Part 1: RBS lab topology picture before the BITL labs were added**

The following infrastructure would be displayed:

(1) Router(s) and firewall(s) connecting the LAN to the Internet Service Provider.

(2) VPN services and protocols currently in use. What is the current picture of LANs and VLANs.

(3) RBS email server and Intranet web server, if they are leveraged by the BITL classes

(4) Any Web proxies, Email gateways, Antiviruses and other security solutions that we need to integrate with.

(5) User Directory (LDAP or Microsoft Directory Service) with up-to-date user data. In many cases we want to avoid creating unnecessary new accounts (and forcing students to memorize additional passwords).

**Part 2: RBS lab topology after the BITL labs were added**

(1) Additional router(s) specifically for the BITL lab, routing policies, WiFi settings, physical location.

(2) Workstations in the lab (unless we opt for BYOD - each student brings his/her own laptop).

(3) Server(s) with VMWare ESXi or Docker host. These machines can hold our development stack (things like database servers, Web servers, some monitoring tools such as Grafite/Grafana, continuos build tools like Jenkins, version control like Git, etc.).

## Section 4: Initial Setup, Monitoring and Troubleshooting

TBD.   
In this section we describe, how to handle some use-cases:

(1) **One-time steps to set up the lab infrastructure.** These steps are not done frequently, some documentation helps, if the initial DevOps engineer has forgotten the steps or has left the company.

(2) **New server or student workstation image.** Instructor needs to set up a particular type of server (or even student workstation) for some activity. If we use VMWare/OVA or Docker/Ansible approaches, we need to find hardware requirements (CPU cores, RAM, virtual HD), networking requirements (NICs, their IP settings), access rights, initial configuration.

(3) **What should we do, if a lab machine is broken.** In some cases it could be quick HW replacement (bring in a new box). But we also need to re-image things, return them to the default state.

(4) **What happens, if lab servers exceed their capacity.** RAM and CPU activity is usually monitored to see, if the servers are healthy, if we need to switch something off.

(5) **What happens, if lab connectivity fails.** If people work from their homes (including on weekends), there may be a need to restore lab resources to their original state, to reboot misbehaving machines.