Prologin System Administration

Release 2020

Association Prologin

Aug 18, 2020

CONTENTS

This documentation hopefully explains everything there is to know about the infrastructure used at Prologin to host the finals of the contest. Our needs are surprisingly complex to meet with our low budgets and low control over the hardware and network, which explains why some things seem very complicated.

CHAPTER

ONE

INFRASTRUCTURE OVERVIEW

This section gives a brief overview of our infrastructure.

1.1 Documentation maintainers

- Alexandre Macabies (2013-2019)
- Antoine Pietri (2013-2020)
- Marin Hannache (2013, 2014)
- Nicolas Hureau (2013)
- Paul Hervot (2014, 2015)
- Pierre Bourdon (2013, 2014)
- Pierre-Marie de Rodat (2013)
- Rémi Audebert (2014-2019)
- Sylvain Laurent (2013)

1.2 Requirements

- Host 100 contest participants + 20 organizers on diskless computers connected to a strangely wired network (2 rooms with low bandwidth between the two).
- Run several internal services:
  - DHCP + DNS
  - Machine DataBase (MDB)
  - User DataBase (UDB)
  - Home File Server (HFS)
  - NTPd
- Run several external services (all of these are described later):
  - File storage
  - Homepage server
  - Wiki

  - Contest website
  - Bug tracking software (Redmine)
  - Documentation pages
  - IRC server
  - Pastebin
  - Matches cluster

1.3 Network infrastructure

We basically have a single local network, containing every user machine and all servers, on the range 192.168.0.0/23. The gateway (named `gw`) is 192.168.1.254.

This local network is further divided in three subnetworks:

- 192.168.0.0/24 is for users (staff and contestants)
- 192.168.1.0/24 is reserved for servers
- 192.168.250.0/24 is the *alien* network, reserved for machines not in the MDB that are pending registration.

1.4 Machine database

The Machine DataBase (MDB) is one of the most important parts of the architecture. Its goal is to track the state of all the machines on the network and provide information about the machines to anyone who needs it. It is running on hostname `mdb` and exports a web interface for administration (accessible to all roots).

A Python client is available for scripts that need to query it, as well as a bare-bones HTTP interface for use in PXE scripts.

It stores the following information for each machine:

- Main hostname
- Alias hostnames (for machines hosting multiple services, or for services that have several DNS aliases, eg. `docs` and `doc`)
- IP
- MAC
- Nearest root file server
- Nearest home file server
- Machine type (user, orga, cluster, service)
- Room id (pasteur, alt, cluster, other)

It is the main data source for DHCP, DNS, monitoring and other stuff.

When a machine boots, an IPXE script will lookup the machine info from the MDB to get the hostname and the nearest NFS root. If it is not present, it will ask for information interactively and register the machine in the MDB.

1.5 User database

The User DataBase (UDB) stores the user information. As with MDB, it provides a simple Python client library as well as a web interface (accessible to all organizers, not only roots). It is running on hostname `udb`.

It stores the following information for every user:

- Login
- First name
- Last name
- Current machine name
- Password (unencrypted so organizers can give it back to people who lose it)
- Type (contestant, organizer, root)
- SSH key (mostly useful for roots)

As with the MDB, the UDB is used as the main data source for several services: every service accepting logins from users synchronizes the user data from the UDB (contest website, bug tracker, …). A PAM script is also used to handle login on user machines.

1.6 File storage

3 classes of file storage, all using NFS over TCP (to handle network congestion gracefully):

- Root filesystem for the user machines: 99% reads, writes only done by roots.
- Home directories filesystem: 50% reads, 50% writes, needs low latency
- Shared directory for users junk: best effort, does not need to be fast, if people complain, tell them off.

Root filesystem is manually replicated to several file servers after any change by a sysadmin. Each machine using the root filesystem will interrogate the MDB *at boot time* to know what file server to connect to. These file servers are named `rfs-1`, `rfs-2`, etc. One of these file servers (usually `rfs-1`) is aliased to `rfs`. It is the one roots should connect to in order to write to the exported filesystem. The other rfs servers have the exported filesystem mounted as read-only, except when syncing.

Home directories are sharded to several file servers, typically two per physical room. These file servers are named `hfs-1`, `hfs-2`, etc. When a PAM session is opened on a machine, a script contacts the HFS to request that this user's home direct be ready for serving over the network. This can involve a migration, but eventually the script mounts the home directory and the user is logged-in.

The user shared directory is just one shared NFS mountpoint for everyone. It does not have any hard performance requirement. If it really is too slow, it can be sharded as well (users will see two shared mount points and will have to choose which one to use). This file server is called `shfs`.

1.7 Other small services

Here is a list of all the other small services we provide that don't really warrant a long explanation:

- Homepage: runs on `homepage`, provides the default web page displayed to contestants in their browser
- Wiki: runs on `wiki`, UDB aware wiki for contestants
- Contest website: runs on `contest`, contestants upload their code and launch matches there
- Bug tracker: `bugs`, UDB aware Redmine
- Documentations: `docs`, language and libraries docs, also rules, API and Stechec docs.
- IRC server: `irc`, small UnrealIRCd without services, not UDB aware
- Paste: `paste`, random pastebin service

CHAPTER

TWO

SETUP INSTRUCTIONS

If you are like the typical Prologin organizer, you're probably reading this documentation one day before the start of the event, worried about your ability to make everything work before the contest starts. Fear not! This section of the documentation explains everything you need to do to set up the infrastructure for the finals, assuming all the machines are already physically present. Just follow the guide!

Note: This section is for the actual hardware setup of the final. To setup a development environment in containers, see *Container setup for SADM*.

2.1 Step 0: Hardware and network setup

Before installing servers, we need to make sure all the machines are connected to the network properly. Here are the major points you need to be careful about:

- Make sure to balance the number of machines connected per switch: the least machines connected to a switch, the better performance you'll get.
- Inter-switch connections is not very important: we tried to make most things local to a switch (RFS + HFS should each be local, the rest is mainly HTTP connections to services).
- Have a very limited trust on the hardware that is given to you, and if possible reset them to a factory default.

For each pair of switches, you will need one *RHFS* server (connected to the 2 switches via 2 separate NICs, and hosting the RFS + HFS for the machines on these 2 switches). Please be careful out the disk space: assume that each RHFS has about 100GB usable for HFS storage. That means at most 50 contestants (2GB quota) or 20 organizers (5GB quota) per RHFS. With contestants that should not be a problem, but try to balance organizers machines as much as possible.

You also need one gateway/router machine, which will have 3 different IP addresses for the 3 logical subnets used during the finals:

Users and services 192.168.0.0/23

Alien (unknown) 192.168.250.0/24

Upstream Based on the IP used by the bocal internet gateway.

Contestants and organizers must be on the same subnet in order for UDP broadcasting to work between them. This is required for most video games played during the finals: server browsers work by sending UDP broadcast announcements.

Having services and users on the same logical network avoids all the traffic from users to services going through the gateway. Since this includes all RHFS traffic, we need to make sure this is local to the switch and not being routed via the gateway. However, for clarity reasons, we allocate IP addresses in the users and services subnet like this:

Users 192.168.0.0 - 192.168.0.253

Services and organizers machines 192.168.1.0 - 192.168.1.253

2.2 Step 1: Writing the Ansible inventory

The Ansible inventory is configuration that is specific to the current SADM deployment. There are two inventories in the `ansible` directory:

- an `inventory` directory, the default inventory, which is used for local testing purposes
- a `inv_final` directory, which contains the deployment for the finals, that we update each year.

TODO: document step-by-step how to write the inventory once we actually have to do it on a physical infra, which hasn't happened yet. Include secrets rotations, addition of MAC addresses, etc.

2.3 Step 2: Setting up the gateway

The very first step is to install an Arch Linux system for `gw`. We have scripts to make this task fast and easy.

2.3.1 File system setup

Note: The installation process is partially automated with scripts. You are strongly advised to read them and make sure you understand what they are doing.

Let's start with the hardware setup. You can skip this section if you are doing a containerized install or if you already have a file system ready.

For `gw` and other critical systems such as `web`, we setup a RAID1 (mirroring) over two discs. Because the RAID will be the size of the smallest disc, they have to be of the same capacity. We use regular 500 GB HDDs, which is usually more than enough. It is a good idea to choose two different disks (brand, age, batch) to reduce the chance to have them failing at the same time.

On top of the RAID1, our standard setup uses LVM to create and manage the system partition. For bootloading the system we use the good old BIOS and `syslinux`.

All this setup is automated by our bootstrap scripts, but to run them you will need a bootstrap Linux distribution. The easiest solution is to boot on the Arch Linux's install medium https://wiki.archlinux.org/index.php/Installation_guide#Boot_the_live_environment.

Once the bootstrap system is started, you can start the install using:

```
bash <(curl -L https://sadm.prolo.in)
```

This script checks out sadm, then does the RAID1 setup, installs Arch Linux and configures it for RAID1 boot. So far nothing is specific to sadm and you could almost use this script to install yourself an Arch Linux.

When the script finishes the system is configured and bootable, you can restart the machine:

```
reboot
```

The machine should reboot and display the login tty. To test this step:

- The system must boot.
- systemd should start without any `[FAILED]` logs.
- Log into the machine as `root` with the password you configured.
- Check that the hostname is `gw.prolo` by invoking `hostnamectl`:

```
 Static hostname: gw.prolo
       Icon name: computer-container
         Chassis: container
      Machine ID: 603218907b0f49a696e6363323cb1833
         Boot ID: 65c57ca80edc464bb83295ccc4014ef6
  Virtualization: systemd-nspawn
Operating System: Arch Linux
          Kernel: Linux 4.6.2-1-ARCH
    Architecture: x86-64
```

- Check that the timezone is `Europe/Paris` and NTP is enabled using `timedatectl`:

```
      Local time: Fri 2016-06-24 08:53:03 CEST
  Universal time: Fri 2016-06-24 06:53:03 UTC
        RTC time: n/a
       Time zone: Europe/Paris (CEST, +0200)
 Network time on: yes
NTP synchronized: yes
 RTC in local TZ: no
```

- Check the NTP server used:

```
systemctl status systemd-timesyncd
Sep 25 13:49:28 halfr-thinkpad-e545 systemd-timesyncd[13554]: Synchronized to␣
↪time server 212.47.239.163:123 (0.arch.pool.ntp.org).
```

- Check that the locale is `en_US.UTF8` with the `UTF8` charset using `localectl`:

```
System Locale: LANG=en_US.UTF-8
    VC Keymap: n/a
   X11 Layout: n/a
```

- You should get an IP from DHCP if you are on a network that has such a setup, else you can add a static IP using a `systemd-network .network` configuration file.
- Check there are no failed systemd services; if there are, troubleshoot them:

```
systemctl status
```

2.3.2 SADM deployment

Now, we can install the Prologin-specific services on `gw` using Ansible. Either from a machine that is on the same network as `gw` or on `gw` itself, retrieve the SADM repository and deploy the gateway playbook:

```
cd ansible
export ANSIBLE_INVENTORY=inv_final
source ./activate_mitogen.sh  # Tool that speeds-up Ansible
ansible-playbook playbook-gw.yml
```

2.3.3 Gateway network configuration

`gw` has multiple static IPs used in our local network:

- 192.168.1.254/23 used to communicate with both the services and the users
- 192.168.250.254/24 used to communicate with aliens (aka. machines not in mdb)

It also has IP to communicate with the outside world:

- 10.a.b.c/8 static IP given by the bocal to communicate with the bocal gateway
- 163.5.x.y/16 WAN IP given by the CRI

The network interface(s) are configured using `systemd-networkd`, in files under `/etc/systemd/network/`.

For this step, we use the following systemd services:

- From systemd: `systemd-networkd.service`: does the network configuration, interface renaming, IP setting, DHCP getting, gateway configuring, you get the idea. This service is enabled by the Arch Linux bootstrap script.
- From sadm: `nic-configuration@.service`: network interface configuration, this service should be enabled for each of the interface on the system.

For more information, see the systemd-networkd documentation.

At this point you should reboot and test your network configuration:

- Your network interfaces should be up (`ip link` should show `state UP` for all interfaces but `lo`).
- The IP addresses (`ip addr`) are correctly set to their respective interfaces.
- Default route (`ip route`) should be the CRI's gateway.

Then, you can also check that the *Core services* are individually running properly, as they will be required for the rest of the setup.

2.4 Step 3: File storage

rhfs naming scheme

A rhfs has two NICs and is connected to two switches, there is therefore two `hfs-server` running on one rhfs machine, each with a different id. The hostname of the rhfs that hosts hfs `0` and hfs `1` will be: `rhfs01`.

A RHFS, for "root/home file server", has the following specifications:

- It is connected to two switches, handling two separates L2 segments. As such, the machine on a L2 segment is only 1 switch away from it RHFS. This is a good thing as it reduces the network latency, reduces the risk if one the switches in the room fails and simplyfies debugging network issues. It also mean that a RHFS will be physically near the machines it handles, pretty useful for debugging, although you will mostly work using SSH.
- Two NICs configured using DHCP, each of them connected to a different switch.
- Two disks in RAID1 setup, same as `gw`.

To bootstrap a rhfs, `rhfs01` for example, follow this procedure:

1. Boot the machine using PXE and register it into MDB as `rhfs01`.
2. Reboot the machine and boot an Arch Linux install medium.

3. Follow the same first setup step as for `gw`: see *File system setup*.

Do that for all the RHFS, then go to the next step.

2.4.1 RHFS Inventory setup

TODO: explain how to write the RHFS parts of the inventory

2.4.2 Installing the RHFS

Like before, once the base system is set up and the inventory has been written, we can install it very simply:

```
ansible-playbook playbook-rhfs.yml
```

After setting up the RHFS systems, we also need to setup the actual root FS that's mounted as `/` by the candidates' machines. Deploy ansible in the container which mounts `/export/nfsroot`:

```
ansible-playbook playbook-rfs-container.yml
```

2.4.3 Registering the switches

To be able to register the machines easily, we can register all the switches in MDB. By using the LLDP protocol, when registering the machines, they will be able to see which switch they are linked to and automatically guess the matching RHFS server.

On each rhfs, run the following command:

```
networkctl lldp
```

You should see an LLDP table like this:

```
LINK    CHASSIS ID          SYSTEM NAME   CAPS        PORT ID           PORT␣
↪DESCRIPTION
rhfs0   68:b5:99:9f:45:40   sw-kb-past-2  ..b........ 12                12
rhfs1   c0:91:34:c3:02:00   sw-kb-pas-3   ..b........ 22                22
```

This means the "rhfs0" interface of rhfs01 is linked to a switch named sw-kb-past-2 with a Chassis ID of 68:b5:99:9f:45:40.

After running this on all the rhfs, you should be able to establish a mapping like this:

```
rhfs0 -> sw-kb-past-2 (68:b5:99:9f:45:40)
rhfs1 -> sw-kb-pas-3 (c0:91:34:c3:02:00)
rhfs2 -> sw-kb-pas-4 (00:16:b9:c5:25:60)
rhfs3 -> sw-pas-5 (00:16:b9:c5:84:e0)
rhfs4 -> sw-kb-pas-6 (00:14:38:67:f7:e0)
rhfs5 -> sw-kb-pas-7 (00:1b:3f:5b:8c:a0)
```

You can register all those switches in MDB. Click on "add switch", with the name of the switch like `sw-kb-past-2`, the chassis ID like `68:b5:99:9f:45:40`, and put the number of the interface in the RFS and HFS field (i.e. if it's on the interface `rhfs0`, put 0 in both fields).

2.5 Step 4: Booting the user machines

All the components required to boot user machines should now properly be installed. Execute this on all the machines:

1. Boot a user machine
2. Choose “Register with LLDP”
3. Follow the instructions to register the machine and reboot.

Here is a test procedure to ensure everything is working well:

1. Boot a user machine
2. Login manager should appear
3. Log-in using a test account (create one if needed), a home should be created with the skeleton in it.
4. The desktop launches, the user can edit files and start programs
5. Log-out ie. close the session
6. Boot a user machine that’s bound to another HFS
7. Log-in using the same test account, the home should be migrated.
8. The same desktop launches, with state preserved from the previous session.

2.6 Step 5: Installing the service machines

We now need to set up all the service machines (monitoring, web, misc). For each of these servers:

1. Boot the machine using PXE and register it into `mdb` as a service with the correct hostname.
2. Reboot the machine and boot an Arch Linux install medium.
3. Follow the same first setup step as for `gw`: see *File system setup*.

Once all of them are running on the network and pinging properly, they can be deployed:

```
ansible-playbook playbook-monitoring.yml
ansible-playbook playbook-web.yml
ansible-playbook playbook-misc.yml
```

2.7 Note: Testing on qemu/libvirt

Here are some notes:

- Do not use the spice graphical console for setting up servers, use the serial line. For syslinux it is `serial 0` at the top of `syslinux.cfg` and for Linux `console=ttyS0` on the cmd line of the kernel in `syslinux.cfg`.
- For best performance use the VirtIO devices (disk, NIC), this should already be configured if you used `virt-install` to create the machine.
- For user machines, use the QXL driver for best performance with SPICE, and use VirtIO for video.

CHAPTER

THREE

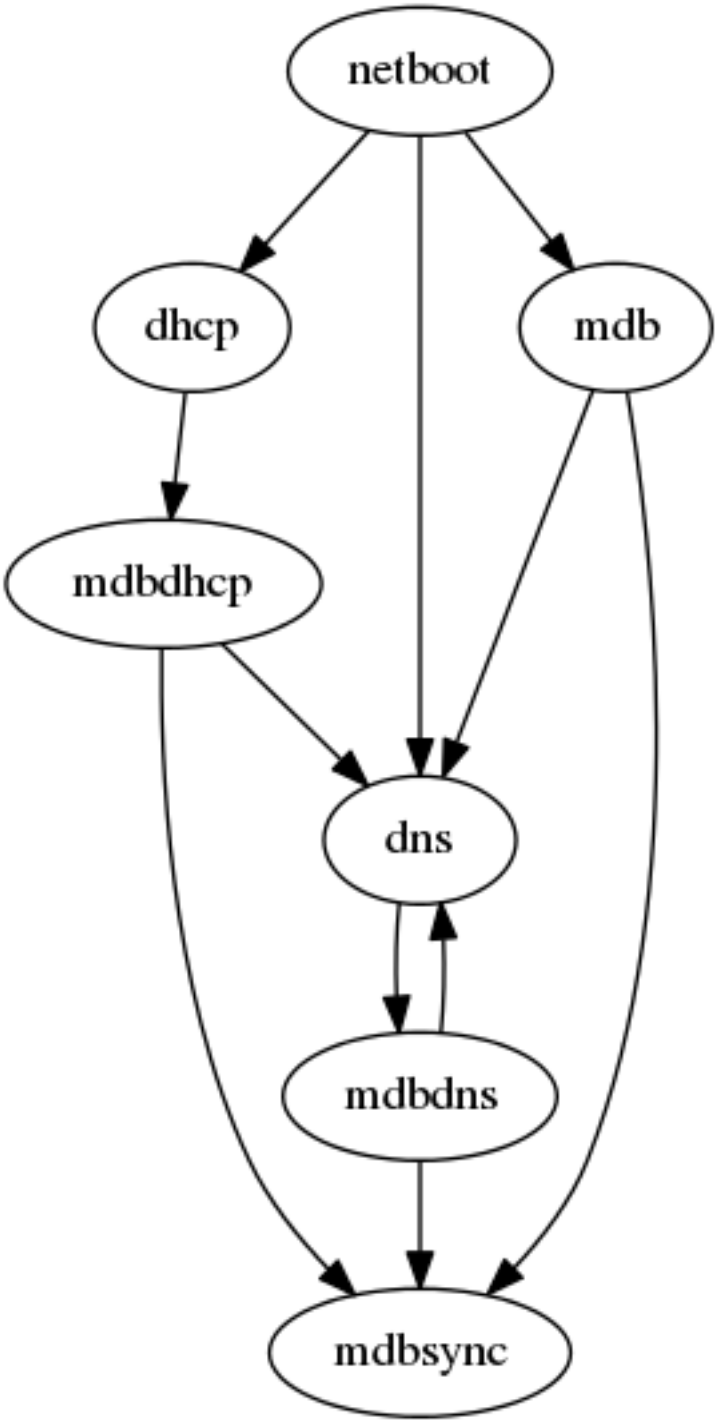
SERVICES

This section covers all the services usually installed in a Prologin SADM setup. They are divided here in the following categories:

3.1 Core services

These services are the backbone of the infrastructure and are installed very early during the setup, as most other services depend on them. They are generally installed on the `gw` machine; they could run elsewhere but we don't have a lot of free machines and the core is easier to set up at one single place.

The services installed here are a bit tricky. As this is the core of the architecture, everything kind of depends on each other:



This is worked around by bootstrapping the infra in a number of ways, most importantly by keeping an initial list of the first core services in `gw:/etc/hosts` to access them while the DNS setup is not available.

3.1.1 mdb

MDB is the machine database described in the infrastructure overview (see *Machine database*), available at `http://mdb/`. Root admins can login there to edit information about the machines.

To check that MDB is working, you can call the RPC endpoint `/call/query`:

```
curl http://mdb/call/query
# Should return a JSON dict containing the machines registered in MDB
```

3.1.2 mdbsync

`mdbsync` is a web server used for applications that need to react on `mdb` updates. The DHCP and DNS config generation scripts use it to automatically update the configuration when `mdb` changes.

To check if `mdbsync` is working, try to register for updates:

```
python -c 'import prologin.mdbsync.client; prologin.mdbsync.client.connect().poll_
↪updates(print)'
# Should print {} {} and wait for updates
```

3.1.3 mdbdns

`mdbdns` gets updates from `mdbsync` and regenerates the DNS configuration.

You can check that `mdbdns` is working by checking that the DNS configuration correctly contains the machines registered in MDB:

```
host mdb.prolo 127.0.0.1
# Should return 192.168.1.254
```

3.1.4 mdbdhcp

`mdbdhcp` works just like `mdbdns`, but for DHCP.

The DHCP server also provides an Arch Linux install medium in PXE to install all the servers. (See https://www.archlinux.org/releng/netboot/)

3.1.5 netboot

Netboot is a small HTTP service used to handle interactions with the PXE boot script: machine registration and chaining on the appropriate kernel files. It runs as `netboot.service`.

3.1.6 TFTP

The TFTP server is used by the PXE clients to fetch the first stage of the boot chain: the iPXE binary (more on that in the next section). It is a simple `tftp-hpa` daemon.

The TFTP server serves files from `/srv/tftp`.

3.1.7 iPXE bootrom

The iPXE bootrom is an integral part of the boot chain for user machines. It is loaded by the machine BIOS via PXE and is responsible for booting the Linux kernel using the nearest RFS. It also handles registering the machine in the MDB if needed.

We need a special version of iPXE supporting the LLDP protocol to speed up machine registration. We have a pre-built version of the PXE image in our Arch Linux repository. The package `ipxe-sadm-git` installs the PXE image as `/srv/tftp/prologin.kpxe`.

3.1.8 udb

UDB is the user database described in the infrastructure overview (see *User database*), available at `http://udb/`. Root admins can login there to edit information about the users.

You can then import all contestants information to `udb` using the `batchimport` command:

```
cd /opt/prologin/udb
python manage.py batchimport --file=/root/finalistes.txt
```

The password sheet data can then be generated with this command, then printed by someone else:

```
python manage.py pwdsheetdata --type=user > /root/user_pwdsheet_data
```

Then do the same for organizers:

```
python manage.py batchimport --logins --type=orga --pwdlen=10 \
    --file=/root/orgas.txt
python manage.py pwdsheetdata --type=orga > /root/orga_pwdsheet_data
```

Then for roots:

```
python manage.py batchimport --logins --type=root --pwdlen=10 \
    --file=/root/roots.txt
python manage.py pwdsheetdata --type=root > /root/root_pwdsheet_data
```

3.1.9 udbsync

`usbsync` is a server that pushes updates of the user list.

3.1.10 presencesync

`presencesync` manages the list of logged users. It authorizes user logins and maintain the list of logged users using pings from the `presenced` daemon running in the NFS exported systems.

3.1.11 presencesync_sso

This listens to both `presencesync` and `mdb` updates and maintains a double mapping `ip addr → machine hostname → logged-in username`. This provides a way of knowing which user is logged on what machine by its IP address. This is used by nginx SSO to translate request IPs to logged-in username.

`presencesync_sso` exposes an HTTP endpoint at http://sso/.

All services that support SSO already have the proper stubs in their respective nginx config. See the comments in `etc/nginx/sso/{handler,protect}` for how to use these stubs in new HTTP endpoints.

Debugging SSO

Typical symptoms of an incorrect SSO setup are:

- you're not automatically logged-in on SSO-enabled websites such as http://udb or http://concours
- nginx logs show entries mentioning `__sso_auth` or something about not being able to connect to some `sso` upstream

Your best chance at debugging this is to check the reply headers in your browser inspection tool.

- if there is not any of the headers described below, it means your service is not SSO-enabled, ie. doesn't contain the stubs mentioned above. Fix that.
- `X-SSO-Backend-Status` should be `working`, otherwise it means nginx cannot reach the SSO endpoint; in that case check that `presencesync_sso` works and http://sso is reachable.
- `X-SSO-Status` should be `authenticated` and `X-SSO-User` should be filled-in; if the website is not in a logged-in state, it means SSO is working but the website does not understand, or doesn't correctly handle the SSO headers. Maybe it is configured to get the user from a different header eg. `Remote-User`? Fix the website.
- if `X-SSO-Status` is `missing header`, it means nginx is not sending the real IP address making the request; are you missing `include sso/handler`?
- if `X-SSO-Status` is `unknown IP`, it means `presencesync_sso` couldn't resolve the machine hostname from its IP; check the IP exists in http://mdb and that `presencesync_sso` is receiving `mdb` updates.
- if `X-SSO-Status` is `logged-out machine`, it means `presencesync_sso` believes no one is logged-in the machine from which you do the requests; check that `presencesync` knows about the session (eg. using http://map/) and that `presencesync_sso` is receiving `presencesync` updates.

3.1.12 firewall

A firewall of iptables rules is automatically installed on `gw`. It handles allowing and disallowing network access, and masquerading the network traffic.

A `presencesync_firewall.service` service automatically updates this firewall to allow internet access to staff and disallow it to contestants during the contest.

3.1.13 conntrack

`conntrack.service` does the necessary logging to comply with the fact that we are responsible for what the users are doing when using our gateway to the internet.

3.1.14 hfsdb

TODO

3.1.15 udbsync_rootssh

TODO

3.1.16 udbsync_django

TODO

3.2 Monitoring services

Monitoring is the art of knowning when something fails, and getting as much information as possible to solve the issue.

We use prometheus as our metrics monitoring backend and grafana for the dashboards. We use elasticsearch to store logs and kibana to search through them.

We use a separate machine for monitoring (called, surprisingly, `monitoring`), as we want to isolate it from the core services, because we don't want the monitoring workload to impact other services, and vice versa.

3.2.1 Prometheus

Prometheus is a monitoring system that stores a backlog of metrics as time series in its database. Prometheus runs on the `monitoring` machine as `prometheus.service`.

All the machines in the infrastructure have a `prometheus-node-exporter.service` service running on them, which periodically exports system information to Prometheus.

Most SADM services come with built-in monitoring and will be monitored as soon as prometheus is started.

The following endpoints are available for Prometheus to fetch metrics:

- `http://udb/metrics`
- `http://mdb/metrics`
- `http://concours/metrics`
- `http://masternode:9021`
- `http://presencesync:9030`
- hfs: each hfs exports its metrics on `http://hfsx:9030`
- workernode: each workernode exports its metrics on `http://MACHINE:9020`.

TODO: add more information on how to use alerting.

3.2.2 Grafana

Grafana is a web service available at `http://grafana/` that allows you to visualize the information stored in Prometheus.

To access it, the username is `admin` and the password is the value of `grafana_admin_password` in your Ansible inventory. It is also possible to allow guest user access, so that contestants will also be able to see the state of the infrastructure.

Some built-in dashboards are automatically added to Grafana during the installation, and show the current state of the machines and main services.

Monitoring screen how-to

We like to have a giant monitoring screen showing sexy graphs to the roots because it's cool.

Start multiple `chromium --app http://grafana/` to open a monitoring web view.

We look at both the `System` and `Masternode` dashboards from grafana.

Disable the screen saver and DPMS using on the monitoring display using:

```
$ xset -dpms
$ xset s off
```

3.2.3 Icinga

Icinga aggregates the logs of all the machines in the infrastructure, and stores it in an ElasticSearch database. It allows you to quickly search for failures in the logs of the entire setup, including the diskless machines.

These logs are exported through the `journalbeat.service` service that is installed on all the machines and extracts the logs from the systemd journal.

TODO: add more information on how to use Icinga

Icinga runs on `monitoring` as `icinga.service`.

3.3 RHFS services

3.3.1 RFS

TODO

Forwarding of authorized_keys

On a rhfs, the service `udbsync_rootssh` (aka. `udbsync_clients.rootssh`) writes the ssh public keys of roots to `/root/.ssh/authorized_keys`. The unit `rootssh.path` watches this file, and on change starts the service `rootssh-copy` that updates the `authorized_keys` in the `/exports/nfsroot_ro`.

3.3.2 HFS

TODO

3.4 Contest services

These contest services are the infrastructure for visualizing, scheduling and running matches and tournaments between the champions. They are one of the most important parts of the final infrastructure and should be prioritized more than non-core services.

3.4.1 Concours

Concours is the main contest website available at `http://concours/`. It allows people to upload champions and maps, schedule champion compilations, and schedule maps. Staff users can also use it to schedule tournaments.

Concours is usually installed on the `web` machine, running as `concours.service`. The on-disk champion and match data is stored in `/var/prologin/concours_shared`. Users are automatically synchronized from UDB with `udbsync_django@concours`.

Note: Concours is a *contest* service. It won't be available to users until the contest gets enabled in the Ansible inventory. See *Switching in and out of contest mode*.

API

Concours has a REST API available at `http://concours/api`. It can be used by contestants to script their interaction with the website.

3.4.2 Masternode

Masternode is the scheduler node of a distributed task queue that runs on all the user machines. Its goal is to retrieve pending tasks to compile champions and run matches from the Concours website, find an available node to run the task and send the required task data to the "Workernodes". Once the task is completed, the Workernode sends back the result to Masternode, which saves it in the database.

Masternode has to be setup on the same machine as Concours, as it requires access to `/var/prologin/concours_shared` to retrieve and save the champion data. It runs as `masternode.service` on `web`.

You can check that Masternode is properly working by going to the master status page on http://concours/status. It will tell you whether Masternode is accessible, and list the Workernodes that are currently attached to it.

3.4.3 Workernode

Workernode is the service that runs on all the user machines. It uses isolate and *camisole <https://camisole.prologin.org/>* as isolation backends, to make sure that the contestants' code is run safely without annoying the machine users.

Workernode runs as `workernode.service`. Workers that cannot be reached from the Masternode are automatically dropped from the pool. It is possible to configure the number of "slots" allocated to each worker to throttle parallelism in the ansible configuration, as well as the time, memory and other resource constraints.

3.4.4 Year-specific customizations

Each year we create a new multiplayer game to be used in the final. We usually need to add customizations for the game of the year, either aesthetic (custom CSS for the theme of the game) or practical (map validation scripts, game configuration, …).

Configuration

TODO: Document how to set the number of players, whether to use maps, etc.

Theme

TODO: Document how to deploy and set the static path of the game theme.

Replay

TODO: Document how to deploy the Javascript replayer script.

Map preview

TODO: Document how to deploy the Javascript map previewing script.

Map validity checks

TODO: Document how to deploy the map checking script to ensure uploaded maps are valid. This might become obsolete if this issue is solved: https://github.com/prologin/stechec2/issues/136

3.4.5 Running tournaments

TODO

3.5 Web services

These web services are usually installed on the `web` machine. They are important services like documentations, wikis, bug tracking systems etc. that make the contest experience better.

3.5.1 Homepage

Homepage is a landing page for all contestants, available at `http://homepage`. It is a simple page that shows a list of links to the available services.

Homepage is a Django application that allows the staff to easily add new links by going to the admin panel here: http://homepage/admin

This service runs as `homepage.service`. Users are automatically synchronized from UDB with `udb-sync_django@homepage`.

3.5.2 Docs

Documentation of the finals (Stechec2 usage, game API, FAQ, …) can be hosted on `http://docs/`. After the initial setup, this page is just an empty page containing instructions on how to upload content here.

For more flexibility, so that the staff working on the game rules can easily make changes to the documentation, the deployment of this documentation is manual. Building and deploying the documentation should look like this:

```
cd prologin20XX/docs
make html
rsync --info=progress2 -r _build/html/* root@web:/var/prologin/docs
```

It is recommanded to add this line as a `make deploy` rule in the documentation Makefile.

3.5.3 DevDocs

The language documentations are available at `http://devdocs/`. It is a self-hosted instance of DevDocs that allows people to easily browse the documentation of the supported languages without having access to the internet.

It is possible for each user to enable or disable the documentation of their languages of choice, so that when they search for a keyword they only see the relevant parts of the documentation.

This service runs as `devdocs.service`.

3.5.4 Wiki

A Wiki service is available at `http://wiki`. It is a self-hosted instance of django-wiki that everyone can edit with any information they want.

TODO: add documentation on how to create the initial pages as staff.

This service runs as `wiki.service`. Users are automatically synchronized from UDB with `udb-sync_django@wiki`.

3.5.5 Paste

A pastebin service is available at `http://paste`. It is a self-hosted instance of dpaste, where everyone can paste anything with syntax highlighting.

This service runs as `paste.service`. Users are automatically synchronized from UDB with `udb-sync_django@paste`.

3.5.6 Bug tracker

TODO: We ditched Redmine and we are working on a good replacement based on Django. This section should be updated once it's ready.

3.5.7 Map

A map of all the users is available at `http://map`. It uses a SVG template which maps the usual machine room setup. It uses machine hostnames to map positions in the room with machines registered in MDB.

The `presencesync_usermap.service` service watches for changes in UDB, MDB and Presencesync, and re-generates a map based on the map template after each change. It also regularly pings the machines to check for network issues and display them.

3.6 Misc services

These services are usually installed on the `misc` machine. They are generally relatively unimportant services that do not prevent the contestants from participating in the finals when not present (game servers, music voting system, …). Performance and reliability are generally not a priority for them, so we put them in this machine.

3.6.1 `/sgoinfre`

`/sgoinfre` is a shared read/write NFS mount between all the contestants. The NFS server is on misc, and all the managed user machines mount it automatically when available through the `sgoinfre.mount` unit.

3.6.2 Radio

Sometimes, when the DJ is sleeping, we allow the contestants to vote for the music that gets played in the music room. We use djraio, which is able to pull music from YouTube and Spotify. It acts as a proxy so that contestants can search for their music without internet access. This service is available at `http://radio/`.

TODO: the setup of this service isn't automated yet. To install it, follow README instructions here: https://bitbucket/Zopieux/djraio

3.6.3 IRC

An IRC server is available at `irc://prologin:6667/`. It's an UnrealIRCd server running as `unrealircd.service`. When connecting to the server, contestants are automatically added to three channels:

- **#prologin**: A free-for-all discussion channel.
- **#announce**: A channel where only the staff can talk, to send announcements to contestants.
- **#issues**: A channel streaming status updates on issues created in the bug tracker.

Everyone can create new channels, there are no restrictions.

Oper privileges

Staff users can get oper privileges on the channels by typing the command:

```
/oper ircgod <operpassword>
```

where `<operpassword>` is the `irc_oper_password` defined in the Ansible inventory.

This will make you automatically join the fourth default chan:

- **#staff**: A channel that only the staff can join, to coordinate together.

IRC issues bot

TODO: A bot is supposed to stream the status updates of issues in the **#issues** channel, but we don't have that anymore since we migrated away from redmine. This section requires updating once we have that back.

Motus bot

An Eggdrop IRC bot that automatically joins the **#motus** channel to play an IRC variant of the game of Motus. To start a game, simply say in the channel:

```
!motus
```

For the full documentation of the bot commands, check out the Motus bot homepage.

3.6.4 Teeworlds Server

A Teeworlds game server named "Prologin" runs as `teeworlds-server.service`. It automatically enables everyone on the LAN to play.

3.6.5 World of Warcraft Server

It is possible to setup a World of Warcraft Server, however the setup is a bit complex and not really automated. Instructions for the setup are available here: https://github.com/seirl/prologin-sadm-wow

This setup is based on the open-source CMaNGOS server implementation.

The repository also contains:

- a `udbsync_mangos` service that synchronises the World of Warcraft accounts from UDB (and automatically adds the staff as Game Masters).
- a `wow` wrapper script that symlinks the WoW data files in the user homes to allow them to keep their own configuration between runs without storing the entire game in their home.

CHAPTER

FOUR

COOKBOOK

All the things you might need to do as an organizer or a root are documented here.

4.1 Install a package for the contestants

The recommended way to add a package for the contestants is to add it in the list of packages, so that it will be also installed in future editions. Simply edit `ansible/roles/rfs_packages/tasks/main.yml` and add the wanted packages where appropriate, then redeploy:

```
ansible-playbook playbook-rfs-container.yml
```

However, sometimes you might not want to add a package in a permanent fashion, and just want to install something in a quick and dirty way on all the RFS exports. In that case, use an Ansible ad-hoc command like this:

```
ansible rfs_container -m shell -a "pacman -S mypackage"
```

4.2 Switching in and out of contest mode

Contest mode is the set of switches to block internet access to the users and give them access to the contest ressources.

Switching to contest mode is completely automated in Ansible. Initially, the inventory should contain this default configuration, which disables contest mode:

```
contest_enabled: false
contest_started: false
```

Switching to contest mode can be done in two steps:

1. Edit your inventory and enable these two values:

   ```
   contest_enabled: true
   contest_started: true
   ```

2. Run the following ansible command:

   ```
   ansible-playbook playbook-all-sadm.yml --tags contest_mode
   ```

Once the contest is over, you will want to go back to a configuration where everyone can access the internet, but you do not want to remove the game packages, because you will need them to run tournament matches. You can go to a configuration that is out of contest mode, but where the contest resources are no longer considered "secret" and are installed on the machines:

1. Change your inventory values to this:

```
contest_enabled: false
contest_started: true
```

2. Run the ansible command again:

```
ansible-playbook playbook-all-sadm.yml --tags contest_mode
```

4.2.1 Customize the wallpaper

To customize the desktop wallpaper, edit your inventory to add a custom wallpaper URL like this:

```
wallpaper_override_url: "https://prologin.org/static/archives/2009/finale/xdm/
↪prologin2k9_1280x1024.jpg"
```

Then, run the following command:

```
ansible-playbook playbook-rfs-container.yml --tags wallpaper
```

This will install the new wallpaper at `/opt/prologin/wallpaper.png`.

The following DE are setup to use this file:

- i3
- awesome
- Plasma (aka. KDE)
- XFCE

Gnome-shell is still to be done.

4.2.2 Customize the lightdm theme

TODO

4.3 Sending announcements

To send pop-up announcements on all the user's machines, use an Ansible ad-hoc command like this:

```
ansible user -m shell -a "/usr/bin/xrun /usr/bin/zenity --info --text='Allez manger !
↪' --icon-name face-cool"
```

`xrun` is a binary we install on all the user machines that knows how to find the currently running X server and sets all the required environment variables to run a GUI program on the remote machine.

4.4 User related operations

Most of the operations are made very simple with the use of `udb`. If you are an organizer, you can access `udb` in read only mode. If you are a root, you obviously have write access too.

`udb` displays the information (including passwords) of every contestant to organizers. Organizers can't see the information of other organizers or roots.

All services should be using `udb` for authentication. Synchronization might take up to 5 minutes (usually only one minute) if anything is changed.

Giving back his password to a contestant First of all, make sure to ask the contestant for his badge, which he should always have on him. Use the name from the badge to look up the user in the `udb`. The password should be visible there.

Adding an organizer **Root only**. Go to `udb` and add a user with type `orga`.

4.5 Machine registration

`mdb` contains the information of all machines on the contest LANs. If a machine is not in `mdb`, it is considered an alien and won't be able to access the network.

All of these operations are **root only**. Organizers can't access the `mdb` administration interface.

Adding a user machine to the network In the `mdb` configuration, authorize self registration by adding a VolatileSetting `allow_self_registration` to true. Netboot the user machine - it should ask for registration details. After the details have been entered, the machine should reboot to the user environment. Disable `allow_self_registration` when you're done.

Adding a machine we don't manage to the user network Some organizers may want to use their laptop. Ask them for their MAC address and the hostname they want. Finally, insert a `mdb` machine record with machine type `orga` using the IP address you manually allocated (if you set the last allocation to 100, you should assign the IP .100). Wait a minute for the DHCP configuration to be synced, and connect the laptop to the network.

4.6 Network FS related operations

Two kind of network file systems are used during the finals, the first one is the Root File System: RFS, the second is the Home File System: HFS. The current setup is that a server is both a RFS and a HFS node.

The RFS is a read-only NFS mounted as a rootnfs in Linux. It is replicated over multiple servers to ensure minimum latency over the network.

The HFS is a read-write, exclusive, user-specific export of their home. In other words, each user has it's own personal space that can only be mounted once at a time. The HFS exports are sharded over multiple servers.

4.6.1 Resetting the hfs

If you need to delete every `/home` created by the hfs, simply delete all nbd files in `/export/hfs/` on all HFS servers and delete entries in the `user_location` table of the hfs' database:

```
# For each hfs instance
rm /export/hfs/*.nbd

echo 'delete from user_location;' | su - postgres -c 'psql hfs'
```

4.6.2 Remove a RAID 1

The first step is to deactivate and remove the volume group:

```
vgchange -a n data
vgremove data
```

Then you have to actually deconstruct the RAID array and zero the superblock of each device:

```
mdadm --stop /dev/md0
mdadm --remove /dev/md0
mdadm --zero-superblock /dev/sda2
mdadm --zero-superblock /dev/sdb2
```

If you want to erase the remaining ext4 filesystem on thoses devices, you can use fdisk.

CHAPTER

FIVE

DISASTER RECOVERY

What to do when something bad and unexpected happen.

Here are the rules:

1. Diagnose root cause, don't fix the consequences of a bigger failure.
2. Balance the "quick" and the "dirty" of your fixes.
3. Always document clearly and precisely what was wrong and what you did.
4. Don't panic!

5.1 Disk failure

5.1.1 Hard fail

The `md` array will go into degraded mode. See `/proc/mdstat`.

If the disk breaks when the system is powered off, the `md` array will start in an inactive state and your will be dropped in the emergency shell. You will have to re-activate the array to continue booting:

```
$ mdadm --stop /dev/md127
$ mdadm --assemble
$ mount /dev/disk/by-label/YOUR_DISK_ROOT_LABEL new_root/
$ exit  # exit the emergency shell and continue the booting sequence
```

CHAPTER

SIX

PAST FINALS

We try to write each year a return of experience on what happened and what changed, how we were able to deal with issues, etc. You can see them here:

TODO: dig up old google docs and complete the matrix!

6.1 Prologin's 2014 setup

Authors:

- Marin Hannache
- Paul Hervot
- Pierre Bourdon
- Rémi Audebert

6.1.1 Overview

We had 2 rooms:

- Pasteur, ~ 96 machines, 95% of them working
- Masters, ~ 42 machines, 90% of them working

We did not setup an alternate cluster because we had enough computing power with all the i7.

6.1.2 Electrical setup

We asked an electrician to setup Masters for ~40 machines.

6.1.3 Hardware setup

All machines were of i7 generation.

We used the machines already in Pasteur.

We moved machines from SM14 into Masters, used screens, keyboards and mice from the storage room of the Bocal. Some SM14 machines were used a server and stored in Pasteur. Each machine we took from SM14 had to put back exactly at the same location se labeled accordingly.

We bought 10 500Go disks, 2 for each RHFS in RAID 1. The EPITA ADM lent us 4 1To racks that we used for the other servers: gw, web, misc1 and misc2.

Note: We bought the same brand for all the disks, it is *not* a good idea to do that. If a disk from the batch is faulty, then it is pretty certain that the other are too. We should have bought disks from different manufacturers.

6.1.4 Network setup

Pasteur

There were 7 switches in Pasteur and 4 boxes to hold them. Each switch's uplink was wired directly back to The Bocal but they could not setup a proper VLAN so they brought a 24 port gibabit switch, removed uplinks from every switch and wired them to this one.

Masters

We borrowed a 48 port Gigabit switch (HP Procurve 2848) from the LSE and satreix lent us his little 16 port Gbit switch. 3/4 of the room was on the 48 port switch and 1/4 was on the other one.

The link between Pasteur and Masters was done by a custom cable setup by the Bocal.

Wifi for organizers

We used a TP-LINK 703n and bridged the WLAN and LAN.

MAC addresses for the organizers' machines were added to mdb with an IP on the services range.

6.1.5 Services organization

GW:

- bind
- dhcpd
- firewall
- mdb
- netboot
- udb
- postgresql database for hfs

Web:

- concours
- postgresql database for concours
- redmine
- map

misc1:

- minecraft
- collectd
- graphite
- dj_ango

RHFS:

- rhfs01 (pasteur)
- rhfs23 (pasteur)
- rhfs45 (pasteur)
- rhfs67 (masters)

6.1.6 The gate lock

There should be another article on the subject.

6.1.7 Issues encountered during the event

Bad network setup

We asked for the network to be setup such as all links were on the same VLAN and no dhcp server. Our gateway were to route the packets to the Bocal's gateway.

Instead, no VLAN was setup, all uplinks were disconnected and all the switches were connected to another Gigabit switch. Because we wanted to have an uplink, we had to add another nic to our gateway and connect it to another network, then route the packets from one interface to another.

Some of the iptables rules we used are in the cookbook.

Switch failure

4~6 hours after the beginning of the event a switch suddenly stopped forwarding packets. After quick checks we diagnosed a hardware problem, and asked the contestants to go to another spot in the machine room.

We rebooted the switch and disconnected every cable from it and started looking for the one that was giving us trouble. At some point it started to work again, and did not fail thereafter. The only cables we did not connect were the uplink, the IP phone and a strange PoE camera.

Services misconfigurations

- mdbDNS misconfiguration: a machine was inserted with a bad hostname (it contained a '_'), causing bind to fail reading the configuration file.
- mdb and DHCP misconfiguration: the MAC address of a machine is used as the primary key, modifying it is like creating another entry in the table. For mdb is added another machine with the same hostname but with another MAC address.

Fix: Remove the offending entry from the database.

Ethernet flow control

One RHFS was flooding the neighboors with pause packets, causing the NBD/NFS to be really slow and make the machines freeze.

Fix: `ethtool --pause autoneg off rx off rx off`

References:

- Beware Ethernet flow control
- Wikipedia: Ethernet flow control

Bad NTP server

We did not edit ntp configuration on the rfs root so it was trying to contact `0.pool.ntp.org` instead of `gw.prolo`.

Fix: pssh on all machines "ntpdate gw && find /home -print0 | xargs -0 touch"

6.1.8 Cookbook

Here are the tools, techniques, and knowledge we used to setup and run everything.

LLDP

The switches broadcasted LLDP packets to every machines connected to them. It contains, among other things, the name of the switch and the port to wich the link is connected. We used those packets to know where each machine was connected, and select the closest RHFS.

Note: Not all the switches sent those packets.

Reloading LVM/RAID devices

```
# If using LVM, remove logical volumes
$ dmsetup remove /dev/mapper/<NAME>
# Deactivate MD device
$ mdadm --stop /dev/mdXXX
# Scan for hotplugged/swapped disks
$ for d in /sys/class/scsi_host/host*/scan; do echo '- - -' > $d; done
# Rescan for RAID devices
$ mdadm --assemble --scan
```

iptables and ipset

We used ipset to implement ip-based filtering.

Sample usage:

```
$ ipset -! create allowed-internet-access bitmap:ip range 192.168.0.0/23
$ ipset add allowed-internet-access 192.168.0.42
$ ipset flush allowed-internet-access
# Allow packets having src in the set
$ iptables -A FORWARD -m set --match-set allowed-internet-access src -j ACCEPT
```

Sample rules:

```
# Rewrite packets going out of interface lan
$ iptables -t nat -A POSTROUTING -o lan -j MASQUERADE
# Allow packets coming from 192.168.1.0/24 to go out
$ iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT
# Black list a set of IP to access port 80
$ iptables -A INPUT -i lan -p tcp --destination-port 80 -m set --match-set allowed-
↪internet-access src -j DROP
# Allow packets in an already established connection
$ iptables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Eggdrop's latency fixes

By default eggdrop added fakelag to the motus modules, we removed it by patching the binary at runtime.

6.2 Prologin's 2019 setup

6.2.1 Overview

We had 2 rooms:

- Paster, 96 machines
- Masters, 41 machines

6.2.2 Network setup

Pasteur

There were 5 48 ports switches in Pasteur and 4 boxes to hold them.

gw.prolo had 1 nic and was the network gateway.

Roots and organizers were on the last 2 rows.

Masters

There were 2 24 ports switches (lent by the bocal). There two RHFS (67 & 89). The room was separated into two parts, each connected to a switch and a RHFS. (67 & 89). The switches were interconnected, and one was connected to the bocal network. (to link Pasteur <-> Master).

Wifi for organizers

We used a NETGEAT AC1200 to bridge the WLAN with the LAN.

MAC addresses for the organizers' machines were added to mdb with an IP on the services range.

6.2.3 Services organization

gw.prolo:

- bind
- dhcpd
- firewall
- mdb
- netboot
- postgresql database for hfs
- udb

web.prolo:

- concours
- masternode
- postgresql database for concours
- map

misc.prolo:

- redmine
- djraio
- irc_gatessh
- sddm-remote
- spotify
- wow (World of Warcraft)

monitoring.prolo:

- elasticsearch
- grafana
- kibana
- prometheus

RHFS:

- rhfs01 (pasteur)
- rhfs23 (pasteur)
- rhfs45 (pasteur)
- rhfs67 (masters)
- rhfs89 (masters)

6.2.4 Issues encountered during the event

No major issue this year.

RHFS sync breaks login

The `rfs/commit_staging.sh` script overwrites `/etc/passwd`, and during the time the `rsync` is running, this file does not contain the `udb` users. This prevented users from logging in and for logged in users tools that relied on it failed. The `/etc/passwd` file is updated by the `udbsync_passwd` service, which is run after the `rsync` is finished.

Impact on contestants: medium

Remediation: See https://github.com/prologin/sadm/issues/169

High network usage, freeze and OOM

After starting a match `concours`, the user landed on the match replay page and a non-identified bug in the match replay stack (Godot web) made the the contestants system freeze due to high network usage. Symptoms where full bandwidth usage of the NIC, >100MB/s and high CPU usage. We suspect that the code entered a busy loop hammering the NFS client. This prevented us from logging-in with ssh, but the prometheus-node-exporter still worked and we could gather logs. We initially had no clue what was causing the freeze, due to a lack of per-process monitoring, but inspection of the machines when they were frozen shown consistent correlation with opening a replay page on concours. Also, users that did not open such page did not experience the freeze.

Unfreezing the machine required either to a) reboot the machine, with risk of lost data and FS corruption, or b) unplug the network cable for some seconds and re-plug it, after that waiting ~30 seconds and the OOM killer would kill the browser. Multiple contestants did reboot their machines when they froze, without data loss.

Impact on contestants: high, ~10 freeze per hour

Detection: created a dashboard in grafana to identify systems with abnormaly high network bandwidth usage

Remediation: fix web replay, limit network bandwidth to allow ssh

Packet drop on uplink

Organizers using `gw.prolo` as uplink saw packet drop that mainly impacted DNS queries. Other part of the network stack were also unstable, but DNS failures had the most impact, mainly on the radio service that was querying external APIs.

Impact on contestants: no impact

Detection: general packet loss, “Server IP address could not be found” error in browsers

Remediation: added retry of network requests

Next year: prepare a secondary uplink in case the main one fails

Heat

We did not have enough fans and the temperature in the rooms was very high.

Next year: ensure each row has a large fan, put drink cart in front of the room.

CHAPTER

SEVEN

ARCH LINUX REPOSITORY

Prologin has setup an Arch Linux package repository to ease of use of custom packages and AUR content.

7.1 Usage

Add the following section to the `/etc/pacman.conf` file:

```
[prologin]
Server = https://repo.prologin.org/
```

Then, trust the repository signing keys:

```
$ wget https://repo.prologin.org/prologin.pub
$ pacman-key --add prologin.pub
$ pacman-key --lsign-key prologin
```

Finally, test the repository:

```
$ pacman -Sy
```

You should see "prologin" in the list of synchronized package databases.

7.2 SADM related packages

Some packages are key parts of the SADM architecture. They should always be the lastest revision possible. The packages we maintain are in the `pkg` folder.

7.3 Uploading packages

Only the owner of the repository's private key and ssh access to repo@prologin.org can upload packages.

To import the private key to your keystore:

```
$ ssh repo@prologin.org 'gpg --export-secret-keys --armor␣
↪F4592F5F00D9EA8279AE25190312438E8809C743' | gpg --import
$ gpg --edit-key F4592F5F00D9EA8279AE25190312438E8809C743
```

Trust fully the key.

Then, build the package you want to upload locally using `makepkg`. Once the package is built, use `pkg/upload2repo.sh` to sign it, update the database and upload it.

Example usage:

```
$ cd quake3-pak0.pk3
$ makepkg
$ ~/rosa-sadm/pkg/upload2repo.sh quake3-pak0.pk3-1-1-x86_64.pkg.tar.xz
```

You can then install the package like any other:

```
# pacman -Sy quake3-pak0.pk3
$ quake3
```

Enjoy!

7.4 More information

The repository content is stored in `rosa:~repo/www`. Use your Prologin SADM credentials when asked for a password or a passphrase.

7.5 Troubleshooting

7.5.1 Invalid signature of a database or a package

This should not happen. If it does, find the broken signature and re-sign the file using `gpg --sign`. You must also investigate why an invalid signature was generated.

CHAPTER

EIGHT

CONTAINER SETUP FOR SADM

This page explains how to run and test the Prologin SADM infrastruction using containers.

Note: TL;DR run `container_setup_host.sh` then `container_setup_gw.sh` from `install_scripts/containers/`

8.1 Why containers?

Containers are lightweight isolation mechanisms. You can think of them as "starting a new userland on the same kernel", contrarily to virtual machines, where you "start a whole new system". You may know them from tools such as docker, kubernetes or rkt. In this guide we will use system-nspawn(1), which you may already have installed if you are using systemd. Its main advantages compared to other container managers are:

- Its simplicity. It does one thing well: configuring namespaces, the core of containers. No configuration file, daemon (other than systemd), managed filesystem or hidden network configuration. Everything is on the command line and all the options are in the man page.
- Its integrated with the systemd ecosystem. A container started with `systemd-nspawn` is registered and manageable with *machinectl(1) <https://www.freedesktop.org/software/systemd/man/machinectl.html>*. You can use the `-M` of many systemd utilities (e.g. *systemctl*, *journalctl*) to control it.
- Its feature set. You can configure the filesystem mapping, network interfaces, resources limits and security properties you want. Just look at the man page to see the options.

Containers compare favorably to virtual machines on the following points:

- Startup speed. The containers share the devices of the host, these are already initialised and running therefore the boot time is reduced.
- Memory and CPU resources usage. No hypervisor and extra kernel overhead.
- Storage. The content of the container is stored in your existing file system and is actually completely editable from outside of the container. It's very useful for inspecting what's going on.
- Configuration. For *system-nspawn*, the only configuration you'll have is the command line.

8.2 Overview

This guides starts by discussing the virtual network setup, then we build and start the systems.

8.3 Networkd in your base system

The container setup requires systemd-networkd to be running on your system. That said, you might not want it to be managing your main network interfaces.

If you want to tell systemd-networkd to not manage your other interfaces, you can run this command:

```
cat >/etc/systemd/network/00-ignore-all.network <<EOF
[Match]
Name=!vz*

[Link]
Unmanaged=yes
EOF
```

8.4 Automated container setup

If you want to setup SADM in containers to test something else than the install procedure, you can use the automated container install scripts. They will create and manage the containers for you and perform a full SADM install as you would do manually. They are intended for automated and end-to-end tests of SADM.

Requirements:

- The host system should be Arch Linux. Experimental support has been added for non Arch Linux hosts (CoreOS) and will be used if the script detects you are not running Arch.
- For convenience, `/var/lib/machines` should be a btrfs volume. The scripts will run without that but you will not have the ability to restore intermediate snapshots of the install. Note that if you don't want to use a btrfs volume you can use:

  ```
  echo 'USE_BTRFS=false' > install_scripts/containers/container_setup.conf
  ```

To start, run the host setup script, you are strongly advised to check its content beforehand, as it does some substantial changes to your system setup:

```
cd install_scripts/containers/
./container_setup_host.sh
```

Then, run the container install scripts:

```
./container_setup_gw.sh
./container_setup_rhfs.sh
./container_setup_web.sh
./container_setup_pas-r11p11.sh
```

That's it!

You should be able to see the containers listed by `machinectl`, and you can get a shell on the system using `machinectl shell CONTAINER_NAME`.

8.5 What do the scripts do?

They automate setups of Arch Linux and SADM components in containers. The commands in the scripts are taken from the main setup documentation. We expect the container setup to follow the manual setup as strictly as possible.

8.6 BTRFS snapshots

Each stage of the system setups we are building can take a substantial amount of time to complete. To iterate faster we user file system snapshots at each stage so that the system can be rollback the stage just before what you want to test or debug.

Each `stage_*` shell function ends by a call to `container_snapshot $FUNCNAME`.

8.7 Cleaning up

If you want to clean up what these scripts did, you must stop the currently running containers. List the containers with `machinectl list` and `machinectl kill` all of them. You can then remove the containers' data by deleting the content of `/var/lib/machines`. List bind-mounted directories: `findmnt | grep /var/lib/machines/` and `umount` them. Then delete the BTRFS snapshots. List them using `btrfs subvolume list .` and delete them using `btrfs subvolume delete`.

8.8 Containers deep dive

As mentioned above, these scripts setup containers using `machinectl`. It's not necessary to understand how the containers work to test features in prologin-sadm, but you may encounter weird bugs caused by them. The following sections discuss some internals of the containers setup.

A key design decision is that the container setup should not require special cases added to the normal setup. This is to avoid bloating the code and keep it as simple as possible. The container setup can highlight potential fixes, for example how to make the setup more generic or how to decouple the services from the underlying system or network setup.

We note that containers do require special configuration. It should be applied in the container scripts themselves.

8.9 Virtual network setup

The first step consists in creating a virtual network. Doing it with containers is not that different compared to using virtual machines. We can still use bridge type interfaces to wire all the systems together, but we also have new possibilities, as the container is running on the same kernel as the host system.

One interesting thing is that we will be able to start one system as a container, let say gw.prolo, and others as virtual machines, for example the contestant systems, to test network boot for example.

We will use a bridge interface, the next problem to solve is to give this interface an uplink: a way to forward packets to the internet, and back again. To do that, we have multiple choices, here are two:

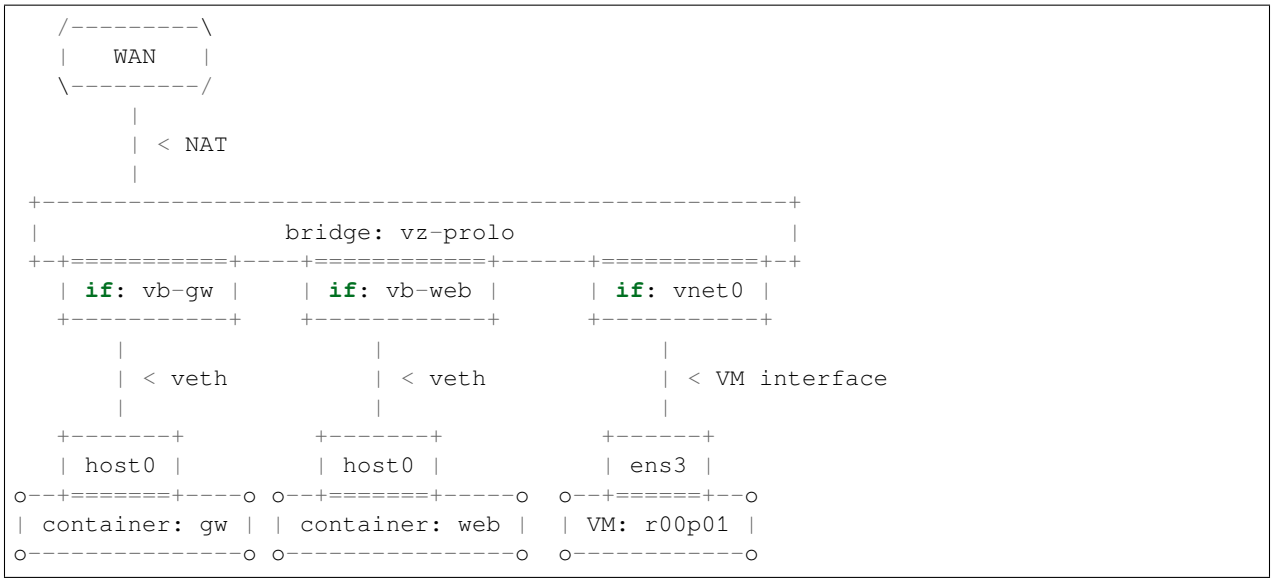
- Masquerade ("NAT") the `vz-prolo` bridge type interface behind your uplink. With this technique the packets arriving on `vz-prolo` will be rewritten, tracked and moved to your uplink to be routed as if they originated from it. The machines behind the NAT will not be accessible directly and you will have to setup port forwarding to access them from outside your system. From within your system they will be accessible directly using their local IP

address. In this guide we will use the "zone" network type of `systemd-nspawn` and `systemd-networkd` as the main system network manager. `systemd-networkd` will manage the NAT in iptables for us. Be careful, if you shadowed the `80-container-vz.network` rule with a catch-all (`Name=*`) `.network` configuration files, the NAT will not be created.

- Bridge your uplink interface with `vz-prolo`. This will have the bad effect to link your LAN, which is most likely already using your router DHCP server, to SADM network, which has its own DHCP server. Depending on various parameters your machine and those on your LAN might get IPs and DNS configuration from Prologin SADM. Be careful if you choose this option, as bridging your uplink will down the interface, `vz-prolo` will get an IP from your DHCP server if you use one and you may have to clean your routes to remove the old ones. It is still the fastest to setup, especially if you just want to give internet to a container. Note: as of 2016, some wireless drivers such as broadcom's `wl` do not support bridging 802.11 interfaces.

The NAT setup is simpler and more flexible, that's what we will use.

All the containers will be connected to their own L2 network using a bridge interface. This interface is managed by systemd, created when the first container using it is spawned. Here is a small diagram to explain how we want the network to look like:



Veth type interfaces what we will use) linked to a bridge will have the name `host0`. `systemd-networkd` provides a default configuration (`80-container-host0.network`) file that enable DHCP on them. With the NAT rule managed by `systemd-networkd` and that, the internet will be accessible out-of-the-box in the containers. The only remaining configuration to do being the DNS resolver (`/etc/resolv.conf`).

8.10 Setting up gw manually

Let's boot the first container: `gw`

Everything starts with an empty directory. This is where we will instantiate the file system used by `gw`:

```
$ mkdir gw
```

Use the Arch Linux install script from the sadm repository to populate it. Here is how to use it:

```
# ./install_scripts/bootstrap_arch_linux.sh /path/to/container machine_name ./file_
↪containing_plaintest_root_pass
```

We suggest storing the password in a text file. It's a good way to be able to to reproduce the setup quickly. If you don't want that, just create the file on the fly or delete it afterwards.

The first system we build is *gw*, so let's create the container accordingly. Run it as root:

```
# ./install_scripts/bootstrap_arch_linux.sh /path/to/gw gw ./plaintest_root_pass
```

Packages will get installed a few scripts run to configure the Arch Linux system. This is the same script we use for the bare metal or VM setup.

Then, start the container with a virtual ethernet interface connected to the `vz-prolo` network zone, a bridge interface managed by systemd, as well an ipvlan interface linked to your uplink:

```
# systemd-nspawn --boot --directory /path/to/gw --network-zone=prologin
```

Note: To exit the container, press 'ctrl+]' three time. `systemd-nspawn` told you that when it started, but there is good chance you missed it, so we are putting it here just for you :)

You should see systemd booting, all the units should be `OK` except `Create Volatile Files and Directories.` which fails because `/sys/` is mounted read-only by `systemd-nspawn`. After the startup you should get a login prompt. Login as *root* and check that you see the virtual interface named `host0` in the container using `ip link`:

```
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT␣
↪group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: host0@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode␣
↪DEFAULT group default qlen 1000
    link/ether e6:28:86:d2:de:6e brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

The host system should have two new interfaces:

- `vz-prolo`, a bridge type interface.
- `vb-gw`, a veth device whose master is `vz-prolo`, meaning it's wired in this bridge.

Both these interface have an extra `@...` suffix. It is not part of the interface name and simply indicates their peer interface.

If you are running `systemd-networkd` on your host system, with the default configuration files, the `vz-prolo` interface will get an IP from a private subnet and a `MASQUERADE` rule will be inserted into iptables. You can start `systemd-networkd` inside the container to get an IP in the `vz-prologin` network, which will be NAT'ed to your uplink.

For some reason `host0` cannot be renamed to `prologin` by a `systemd-networkd` .link file. What needs to be changed to account for that is:

- The firewall configuration

You can do the usual install, with the following changes:

- In `prologin.network`, in `[Match]`, set `Name=host0` to match the virtualized interface.

What will *not* work:

- Some services are disabled when run in a container, for example `systemd-timesyncd.service`.
- `nic-configuration@host0.service` will fail (`Cannot get device pause settings: Operation not supported`) as this is a virtual interface.

Note: When you exit the container everything you started inside it is killed. If you want a persistent container, run:

```
# systemd-run systemd-nspawn --keep-unit --boot --directory /full/path/to/gw --
↪network-zone=prologin
Running as unit run-r10cb0f7202be483b88ea75f6d3686ff6.service.
```

And then monitor it using the transient unit name:

```
# systemctl status run-r10cb0f7202be483b88ea75f6d3686ff6.service
```

8.11 Manual network configuration

This section is a do-it-yourself version of the `--network-veth --network-bridge=prologin` nspawn's arguments. The main advantage of doing so is that the interfaces are not deleted when the container is shut down. Its useful if you have iptables rules you want to keep.

First let's make sure we have ip forwarding enabled, without that the bridge will move packets around:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

We will create a bridge interface named `prologin` that will represent the isolated L2 network for SADM:

```
# ip link add prologin type bridge
```

You can now see the prologin interface using:

```
# ip link show
...
4: prologin: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN␣
↪mode DEFAULT group default qlen 1000
```

For each system we want to start, we create a veth and plug one end to the bridge. For example for the `gw`:

```
# ip link add gw.local type veth peer name gw.bridge
# ip link show label 'gw*'
```

Here we create the two virtual ethernet interfaces, `gw.local@gw.local` and `gw.bridge@@gw.bridge`. On veth pairs, a packet arriving to one these interface is dispatched to the other. When manipulating them only the part of the name before the `@` is required, the other is just a reminder of what interface is at the other end.

Let's wire `gw.bridge` to the bridge:

```
# ip link set gw.bridge master prologin
```

You can see that the interface is connected to the bridge with the `master prologin` keyword on the following command:

```
$ ip link show gw.bridge
```

The interface is not running (`state DOWN`), we have to enable it:

```
# ip link set dev prologin up
```

8.12 Going further/discussion

What could make your container usage better?

- Use the `--overlay` option from `systemd-nspawn`. Have only one base Arch Linux distro and build other systems form it. It reduces the time to install and disk usage (if that's your concern).

CHAPTER

NINE

RUNNING THE WEBSITES WITHOUT A COMPLETE SADM SETUP

When hacking on the various web services included in SADM, it is not necessary to setup a full-blown SADM infrastructure. Typically, when making the design of `concours` website for a new edition, only a small Django setup has to be completed.

1. Clone SADM and cd into it:

```
git clone https://github.com/prologin/sadm
cd sadm
```

2. Configure the website:

```
# for the 'concours' site
vim etc/prologin/concours.yml
```

   Refer below for a guide of values to adapt depending on the website being worked on.

3. Create a venv (don't activate it yet):

```
python -m venv .venv
```

4. Add the configuration path to `.venv/bin/activate` so it is automatically set up when you activate the venv:

```
echo "export CFG_DIR='$PWD/etc/prologin'" >> .venv/bin/activate
```

5. Activate the venv, install the requirements:

```
source .venv/bin/activate
pip install -r requirements.txt -e .
```

6. Apply the migrations, create a super-user and run:

```
# for the 'concours' site
cd django/concours
python manage.py migrate
python manage.py createsuperuser --username prologin --email x@prologin.org
# fill in the password;
python manage.py runserver
```

   Go to http://localhost:8000/, use `prologin` and the password you just chose to log in.

9.1 Working on concours

9.1.1 Configuration

Customize etc/prologin/concours.yml with the following:

db.default The easiest way is to use SQLite:

```
ENGINE: django.db.backends.sqlite3
NAME: concours.sqlite
```

contest.game Use the correct year, typically prologin2018 for the 2018 edition.

contest.directory Use a writable directory, eg. /tmp/prologin/concours_shared.

website.static_path Put the absolute path to prologin<year>/www/static or whatever directory is used for this edition.

Other contest entries (eg. use_maps) Adapt to the correct settings for this edition.

9.1.2 Importing a stechec2 dump for testing

When developing the Javascript replay and other features, you might need to import test dumps that can be loaded on the website.

While in the correct virtualenv:

```
cd django/concours
python manage.py import_dump /path/to/my/dump.json
```

This will create a dummy match with zero players and no map, that will successfully load on the dedicated URL. The match detail URL output by this command will only work in the default setup where manage.py runserver is used on localhost:8000. Adapt the host/port if needed.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search