Jitsi on OpenBSD

Puffy presents video conferencing

Philipp Buehler
sysfive.com GmbH
Hamburg, Germany
pb-openbsd@sysfive.com

***Abstract*—This paper will cover all bits and bolts to fully understand the components at play, their intercommunications and how this knowledge can be used to create a Jitsi-on-OpenBSD setup that features a restricted (compartmentalized) setup using dedicated machines or -as shown- VMM based VMs, where each VM runs only one of the components.**

It'll be documented what's necessary to create a sensible pf.conf on each VM and how to add reverse proxy (relayd, haproxy) for distribution of workload.

Also covering pitfalls/hints along underlying components and what to lookout for on the client/browser side for interopability.

***Index Terms*—Jitsi, OpenBSD, VMM**

I. Introduction

Jitsi and OpenBSD are both not covered much as a documented setup. Installation documents for Jitsi are almost always about Linux OS (and there mostly Debian) and do not cover some internals. The reference documentation on the other hand can be very overwhelming.

There is some FreeBSD "all in one" port (package) with no explanation and it cannot be used to install core components (only) on different nodes.

This documentation is to show a distributed install on OpenBSD using pre-packages and need-to-function (minimum) firewall settings (‘pf.conf‘).

The described configurations will enable a fully integrated video conferencing to be available at ‘https://jts.fips.de‘ - adapt the setting accordingly for your domain.

II. Riddles

Both major players show obstacles that have to be overcome to gain a functioning installation.

A. Jitsi

A "full blown" Jitsi installation can consist over over a dozen components and all the necessary networking/firewalling configuration can be exhaustive. Any possible discovery magic is not documented.

Some configuration snippets are undocumented and tend to make the understanding poorer not better. Worst example are necessary DNS settings and ‘nginx.conf.‘

The typical answer to be found on asking question is to use the official ‘all-in-one‘ Debian VM.

B. OpenBSD

This also leds to the question if it's possible to run a (core) Jitsi installation on OpenBSD only or if there's need be for Linux (VM).

Also it's a bit difficult to find example-based documentation on VMM, e.g. for using VMM as the core router, too. (Combination ‘vm.conf‘+‘pf.conf‘s).

Can we scale the installation horizontally and how to use Java based applications with ‘rcctl‘.

III. Components

A. OpenBSD

In this example setup I make heavy use of ‘VMM‘ eco system on OpenBSD which consists of:

- ‘vmm(4) - virtual machine monitor:‘ kernel driver isolating/providing the required resources for the VMs (hypervisor)
- ‘vmd(8):‘ userland daemon to interact with ‘vmm‘
- ‘vmctl(8)‘: administrative tool to create, start/stop, .. VMs
- ‘vm.conf(5)‘: persist VMs resource configuration

B. Jitsi

A ‘core‘ (basic video conferencing) setup comprised by:

- ‘nginx(8)‘ web: serving web assets and reverse proxy BOSH or websockets
- ‘prosody(8)‘ xmpp: conference chat + internal components communication (esp.PubSub for health/discovery)
- ‘jicofo‘ JItsi COnference FOcus: room+session handling in conferences (whos talking to whom and where)
- ‘jvb‘ videobridge: mediastream (WebRTC) handlings between participants (SFU)
- ‘jibri‘ JItsi BRoadcasting Infrastructure (optional): recording + streaming conferences

IV. Architecture

To host the Jitsi components in a VM each, this uses the following architecture (see Figure 1).

V. Communications

Communications between the components and the logical ‘publication‘ + ‘subscription‘ in Jitsi is as follows (needed in ‘pf.conf‘ later on) (see Figure 2+3).

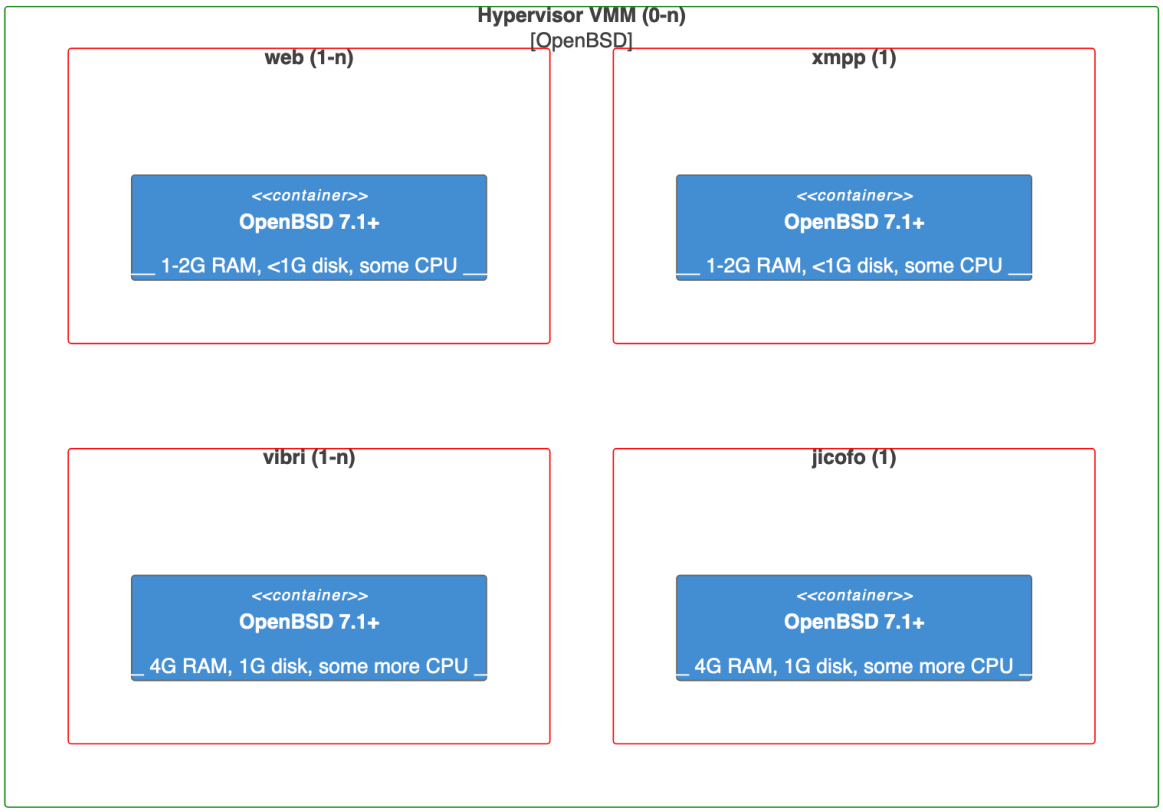

Fig. 1. OpenBSD VMM architecture

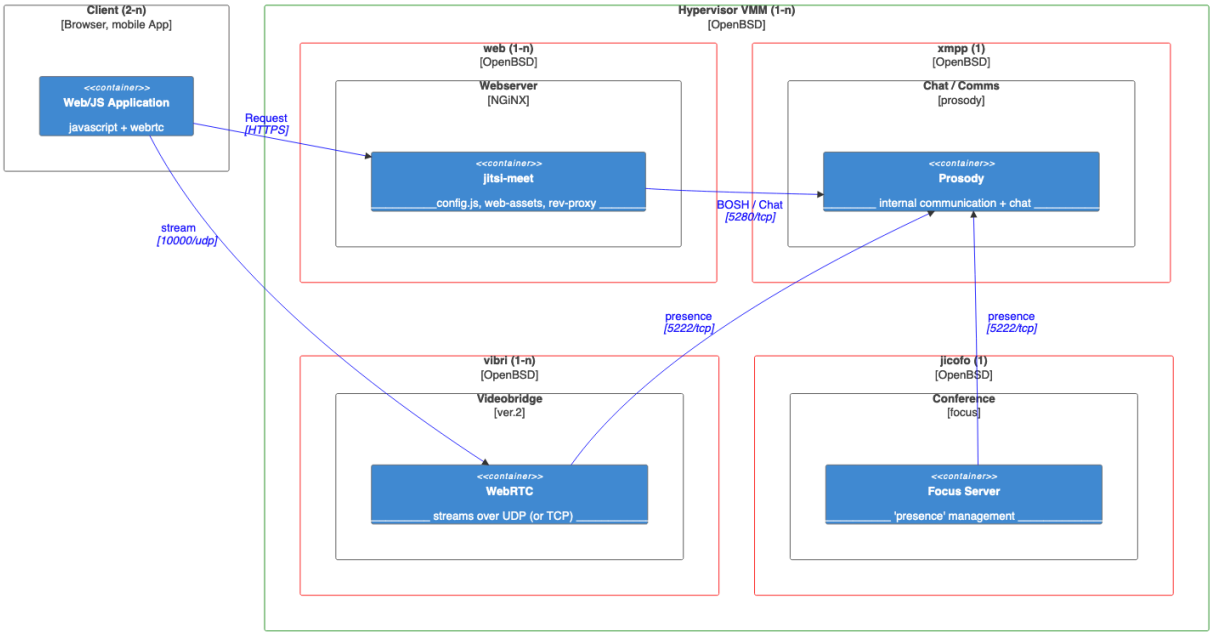

Fig. 2. OpenBSD VMM architecture

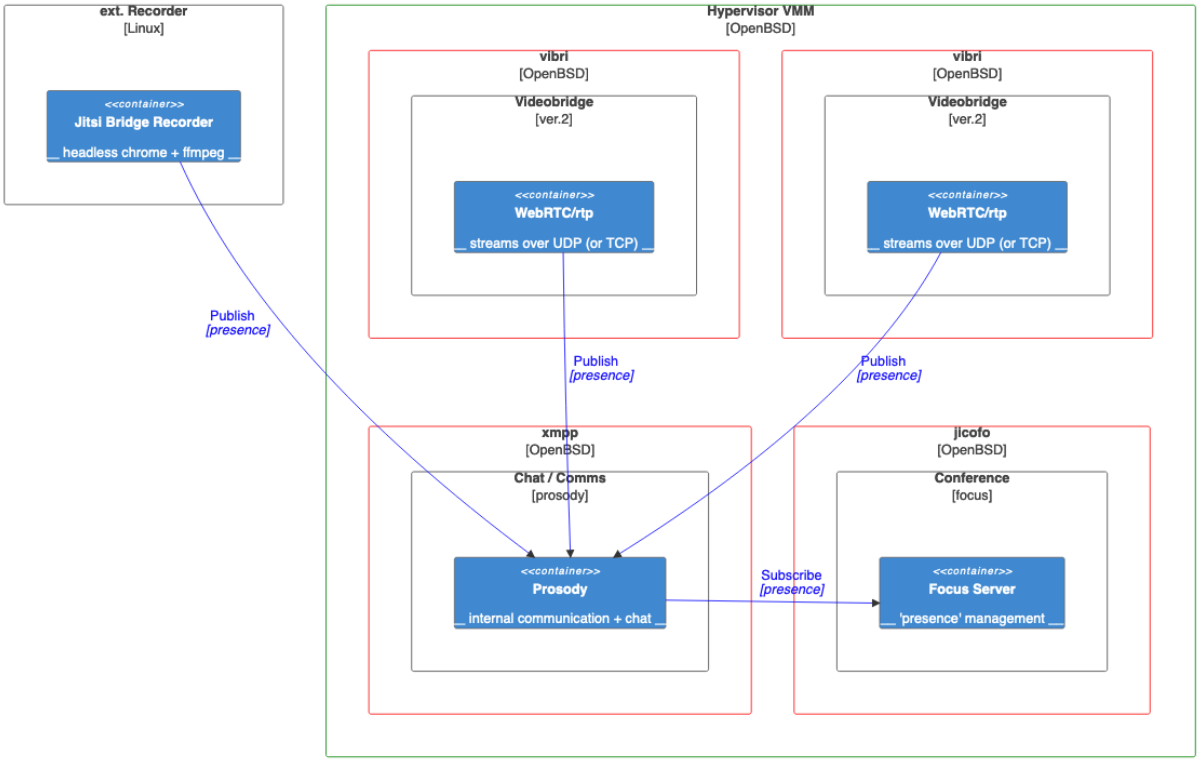

Fig. 3. OpenBSD VMM architecture

VI. Installation

The installation is structered in the following steps:

- create VM images
- construct /etc/vm.conf
- add hosts / DNS
- ‘nginx‘: install, config, certs
- ‘prosody‘: pkg install, config, certs, users
- ‘jicofo‘: pkg install, config
- ‘jvb‘: pkg install, config

A. VM setup

To create an VM image being used in the following setup:

```
rcctl enable vmd; rcctl start vmd
mkdir /home/vmm; cd /home/vmm
vmctl create -s 5G web.qcow2
ftp https://cdn.openbsd.org/pub/OpenBSD
/7.1/amd64/install71.iso
vmctl start -m 2G -L -i 1 -r install71.iso\
-d /home/vmm/web.qcow2 web
vmctl console web
## run the (I)nstaller, default options.
## only one 'a' slice on (w)hole disk
## halt -p (so new sshd_keys per VM)
vmctl stop web
for vm in xmpp jicofo jvb ; do
cp web.qcow2 \${vm}.qcow2; done
echo 'net.inet.ip.forwarding=1' >> \\
/etc/sysctl.conf
```

The VM definitions in ‘/etc/vm.conf‘ as follows. The ”instance” tells ‘vmctl‘ to use ”web”’s configuration as a template and only adapt changes like ”disk” or ”memory” to it.

```
vm "web" {
  enable
  memory 2G
  disk "/home/vmm/web.qcow2" format qcow2
  local interface { up }
}
vm "web" instance "xmpp" {
  disk "/home/vmm/xmpp.qcow2" format qcow2
}
vm "web" instance "jicofo" {
  memory 4G
  disk "/home/vmm/jicofo.qcow2" format qcow2
}
vm "web" instance "jvb" {
  memory 4G
  disk "/home/vmm/jvb.qcow2" format qcow2
}
```

B. DNS + /etc/hosts

DNS: ONE A-RR for jts.fips.de; but local hosts for jicofo (or split DNS).

The following ‘/etc/hosts‘ needs to be on each VM and on the VMM host. Use these names in ‘/etc/myname‘, too.

```
100.64.1.3     web
100.64.2.3     xmpp jts.fips.de
```

```
100.64.3.3    jicofo
100.64.4.3    jvb
```

VII. Firewalling

On each VM the following ‘pf.conf‘ is for good (admin) measure:

```
block return log
pass out quick on egress proto { tcp udp }
  to any port { 123 53 80 443 }
pass in quick on egress proto tcp
  from \$admin to port 22
```

A. VMM

Assumption that all traffic hits the VMM external IP-address (on egress) here:

```
pass in on egress proto tcp to any
  port { 80 443 } rdr-to web
pass in on egress proto udp to any
  port { 10000 } rdr-to jvb
pass in proto tcp from { jvb jicofo }
  to xmpp port 5222 # native
pass in proto tcp from web to xmpp
  port 5280 # http/BOSH
pass in on egress proto tcp to any
  port 5280 rdr-to xmpp # debug
# DNS
vms={ web xmpp jicofo jvb }
pass in proto { udp tcp } from \$vms
  to any port domain rdr-to \$resolver
```

B. web/nginx

The webserver needs the basic ports and for the proxy connection to BOSH an outgoing to 5280/tcp.

```
pass in quick on egress proto tcp
  to self port { 80 443 }
pass out quick on egress proto tcp
  to xmpp port 5280
```

C. prosody

The XMPP ”native” only used from jicofo and jvb. The BOSH as above and if need be the dedicated auth via 5347/tcp (not covered here).

```
pass in proto tcp from { jicofo jvb }
  to self port { 5222 }
pass in proto tcp from web
  to self port 5280
pass in proto tcp from { \$admin }
  to self port { 5280 5347} # debug
```

D. jicofo

Jicofo talks to prosody as above. The webclient connects to jicofo via 10000/udp. There is an administrative connection possible to 8080/tcp to establish e.g. monitoring with prometheus. for scale out the 10000/udp can be changed and would make use of a port range e.g. 10000:10050 vertically or explicit rdr-to in VMM for horizontally.

```
pass out quick on egress proto
  tcp to xmpp port { 5222 5280 }
pass in quick on egress proto
  udp to self port 10000
pass in quick on egress proto
  tcp from \$monitor to self port 8080
```

E. videobridge

The videobridge only needs to reach out to prosody. The monitoring exporter on 8888/tcp seems to be broken (no values) as of this writing.

```
pass out quick on egress proto
  tcp to xmpp port 5222
pass in quick on egress proto
  tcp from \$monitor to self port 8888
```

VIII. Prosody

To install prosody some simple steps are enough:

```
pkg_add unzip-- prosody
prosodyctl install \
 --server=https://modules.prosody.im/rocks/ \
 mod_client_proxy
prosodyctl install \
 --server=https://modules.prosody.im/rocks/ \
mod_roster_command
```

The modules do not need further configuration in ‘prosody.cfg.lua‘. ‘client_proxy‘ gets loaded with ”Component” configuration. ‘roster_command‘ is CLI only. The configuration key bits in ‘/etc/prosody/prosody.cfg.lua‘ are:

```
http_interfaces = { "*", "::" }
VirtualHost "jts.fips.de"
    authentication = "anonymous";

    modules_enabled = { "bosh";
      "pubsub"; }
    c2s_require_encryption = false

VirtualHost "auth.jts.fips.de"
    admins = { "focus@auth.jts.fips.de",
     "jvb@auth.jts.fips.de" }

    ssl = { key =
     "/var/prosody/auth.jts.fips.de.key";
        certificate =
    "/var/prosody/auth.jts.fips.de.crt"; }

    authentication = "internal_hashed"

Component "conference.jts.fips.de" "muc"
Component "jvb.jts.fips.de"
    component_secret = "CHANGE_jvb"
Component "focus.jts.fips.de" "client_proxy"
    target_address =
      "focus@auth.jts.fips.de"
```

```
Component "internal.auth.jts.fips.de" "muc"
    muc_room_locking = false
    muc_room_default_public_jids = true
```

The additional ‘FQDN‘ are internally only and do NOT need any DNS configuration (like an HTTP ‘Host‘ header).

It’s possible that ‘jvb‘ follows the example of ‘focus‘ to change authentication from shared secret to a user (‘target_address‘).

A. Users

The connection for ‘jvb‘ uses a shared secret as shown on the previous page. For further users in use here:

```
rcctl enable prosody
rcctl start prosody
prosodyctl register focus \
  auth.jts.fips.de CHANGE_FOCUS
prosodyctl mod_roster_command \
  subscribe focus.jts.fips.de \
  focus@auth.jts.fips.de
```

B. TLS

Besides WebRTC demanding to use ‘https‘ between browser and server side, we shall encrypt all internal traffic, too. For the prosody connections:

```
prosodyctl cert generate \
  auth.jts.fips.de
# fill in `openssl req` dialog

cd /var/prosody
yes | /usr/local/jdk-11/bin/keytool
  -import -alias prosody -file \
  auth.jts.fips.de.crt -keystore \
  jicofo-key.store -storepass jitsicool

cp jicofo-key.store jvb-key.store
# copy to VM jicofo and jvb accordingly
```

‘keytool‘ comes with JDK, this task can also be done on jicofo or jvb VM - or copy over the resulting store-files to jicofo/jvb VM respectivly. Do NOT change JDK’s ‘lib/security/cacerts‘ - any later upgrade from jdk-11 would ”forget” the changed keystore.

This paper, presentation slides and other directly related resources can be found on github: https://github.com/double-p/presentations/AsiaBSDCon/2022/

REFERENCES

[1] OpenBSD project https://www.openbsd.org/
[2] Jitsi https://github.com/jitsi/