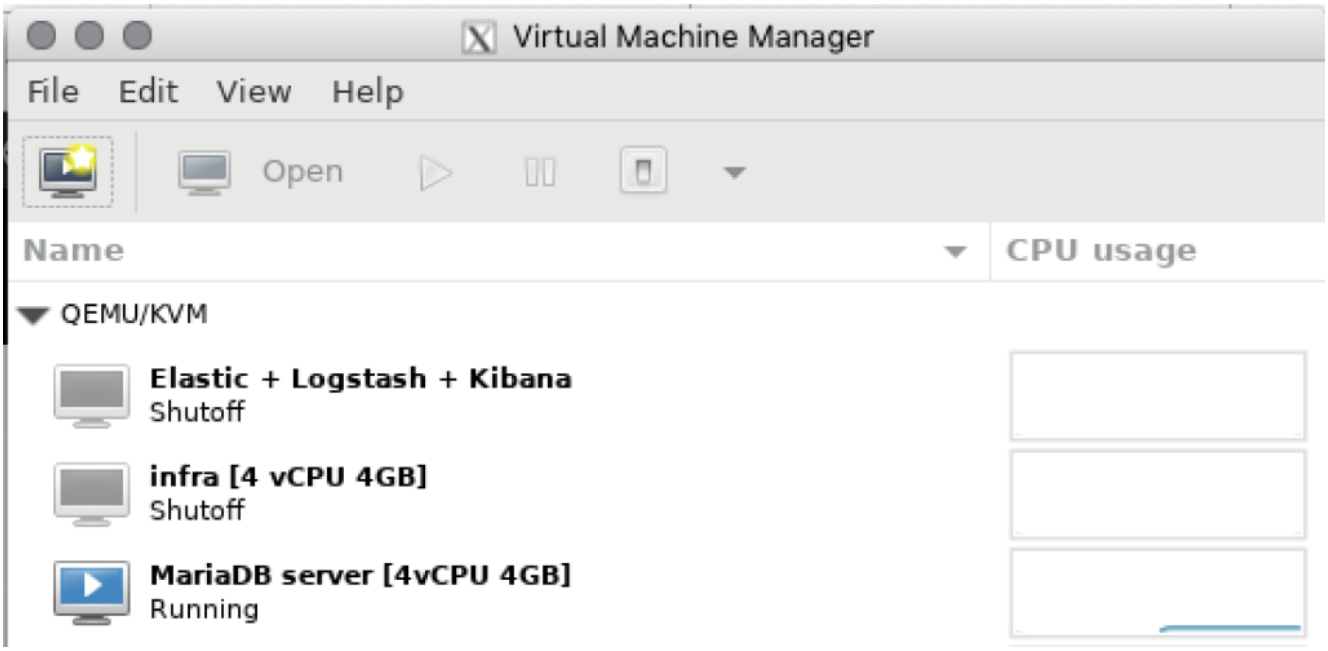
Get X11 Forwarding In macOS High Sierra



I needed to forward X11 output from one of my Linux servers recently to run **virt-manager** (manager for virtual machines in KVM), and because it's been a while I had to download and install X11 server again.

As some of you know, Xorg server is no longer shipped/installed with macOS by default. So you have to download it from XQuartz page: https://www.xquartz.org/releases/index.html. Usually you do it, install it and that's it – no additional steps are needed.

But things are slightly different for the macOS High Sierra, apparently.

The latest release hasn't been updated since 2016 which I believe is before High Sierra – which explains why things don't "just work" anymore. Fear not though – I tracked the issue down and it's explained below.

UPDATE 03/2019: MacOS Mojave works just great, you may skip Step 3 in the procedure below.

Steps to get X11 Forwarding in macOS High Sierra

1. Download and install the latest release from xquartz.org website
2. Start XQuartz
3. **IMPORTANT**: verify xauth location
   SSH configuration file /etc/ssh/ssh_config might contain path to xauth tool, which may be incorrect depending on your OSX/MacOS version. Here's how to check:

   greys@maverick:~ $ **grep xauth /etc/ssh/sshd_config**

   if this returns nothing, you can skip to Step 4 below. If this gives you an output, compare it to the path from the next command:

   greys@maverick:~ $ **which xauth**
   /opt/X11/bin/xauth

   If the locations differ, update the /etc/ssh/ssh_config file:

   greys@maverick:~ $ **sudo vi /etc/ssh/ssh_config**

4. Connect to remote server using -X option which does X11 forwarding for SSH:

   greys@maverick:~ $ **ssh -X centos.unixtutorial.or**

5. Check the DISPLAY variable, it should now be set correctly:

   greys@centos:~ $ **echo $DISPLAY**
   localhost:10.0

That's it for today!

See Also

- Enable Auto-Start in KVM
- hw virtualization
- Fix fonts in X11 forwarding

SSH Port Forwarding

SSH port forwarding is a feature of SSH protocol that allows client and server to forward additional network connections using base SSH session as a secure, encrypted and compressed (for improved performance) tunnel.

Naturally, SSH port forwarding is just a specific SSH-based implementation of a bigger concept: **port forwarding** in general helps you get around rigid network and firewall structures by allowing bi-directional specific network connectivity via certain network ports. The reason SSH port forwarding is so popular is because SSH client-server session provides most of the requirements for network traffic, so port forwarding is a logical extension of the SSH functionality.

How SSH port forwarding works

When you establish a new SSH connection, your SSH client connects to remote server on SSH port, usually **22/tcp**. When you configure port forwarding, you gain ability to map certain remote server's TCP ports locally onto your SSH client device: to your smartphone or PC these connections would be presented as if their services listen on local ports. Connecting to such forwarded ports would send all the network

packets inside your score SSH tunnel.

TODO: Diagram 1 (will update this page later).

Types of SSH port forwarding

There are three SSH port forwarding techniques:

1. **Local port forwarding** – making remote connections available by mapping them as local TCP ports on your client device. Although the easiest scenario is to use local forwarding to access services on the remote SSH server, it is also possible and very useful to use one SSH server for forwarding network services provided by other servers that are accessible for it.
2. **Remote port forwarding** – the opposite of local forwarding: mapping remote ports (activated on your remote SSH server) to local services accessible from your SSH client or other servers on its local network.
3. **Dynamic forwarding** – this forwarding will configure SOCKS proxy on your local system, meaning other applications like web browsers or file transfer clients can be configured to use Dynamic Forwarding for complete network access tunnelled through SSH session. Compared to Local and Remote forwarding, this approach is way more flexible because you do not have to configure (or even know) specific TCP ports of network connections that need forwarding.

Additional SSH forwarding features

X11 forwarding

X11 forwarding is a Unix native solution to forward graphical applications from a remote server: you start an application on the remote SSH server and get GUI interface for the application shown on your Linux or Unix client.

Agent forwarding

This functionality allows you to easily and transparently access multiple SSH servers behind initial server A connection, because your identity (public SSH key) and private SSH key exchange are forwarded securely back to your SSH client. The main benefit is that you're not asked for your SSH password for every SSH server you're connecting from your initial server A session, and you don't have to put your private SSH key onto server A for SSH key-based access to local servers behind it.

What You Can Do with Port Forwarding

This will be a growing collection of SSH port forwarding scenarios that we find to be the most useful. We will continue expanding the list and documenting new ways of using SSH port forwarding.

Definitions

- **Server A** – SSH server that you can connect to: it has public IP address and you have a working username/password or SSH key to access the server. It will usually have a number of local IP addresses that you can't connect to directly, but can use for accessing sever A's local area network (LAN).
- **Server B** – any other server on the same LAN as server A. You can't access B directly because such servers only have local IP addresses and are protected by firewalls. But server A access to server B via local IP addresses, and as such can be used to forward your client's connections to services on server B.
- **Client C** – your SSH client: a smartphone or a PC (usually your desktop) where you're running SSH client (command line, standalone SSH software or even a mobile

app).

Local Port Forwarding

- access another server’s SSH via existing SSH connection (connect to SSH server B via SSH server A)
- access network services on the remote SSH server
- connect to MySQL/Postgres server via SSH server A
- access intranet web server/wiki via SSH server A
- Remote Desktop access – connecting to server A or B’s RDP (3389/tcp) or VNC (5000/tcp and higher) session
- secure and encrypt your access to remote protocols that are otherwise insecure
    - FTP
    - telnet/rsh
    - IMAP/POP3 (if IMAPs and POP3s are unavailable)
    - Web access on HTTP (if HTTPS is unavailable)

Remote Port Forwarding

- allow remote SSH servers on the Internet to access (and present to the Internet) your network services from SSH client or other local systems
- serve local web server traffic
- provide remote SSH access to local servers
- allow RDP/VNC access to your home system
- provide online DEV environment (with public IP addresses) access to your development MySQL/Postgres hosted on laptop

Dynamic Forwarding

- configure one of your browsers to use SSH based SOCKS proxy for accessing remote (intranet) resources behind SSH server
- allow your SSH client and other systems on your LAN to

access Internet using your remote SSH server's public IP address (to access third parties that have access list and disallow dynamic IP addresses)
- Configure individual SSH client systems to use different public IP addresses (by making each of them for dynamic forwarding via different remote SSH servers)

Port Forwarding: Summary

That's it for the moment! Hope you learned something new and found answers to some of your questions. Please get in touch if you would like me to explain something about port forwarding using your specific example!

See Also

- SSH port
- SSH command
- Enable SSH server in Ubuntu
- Change passphrase to your private SSH key