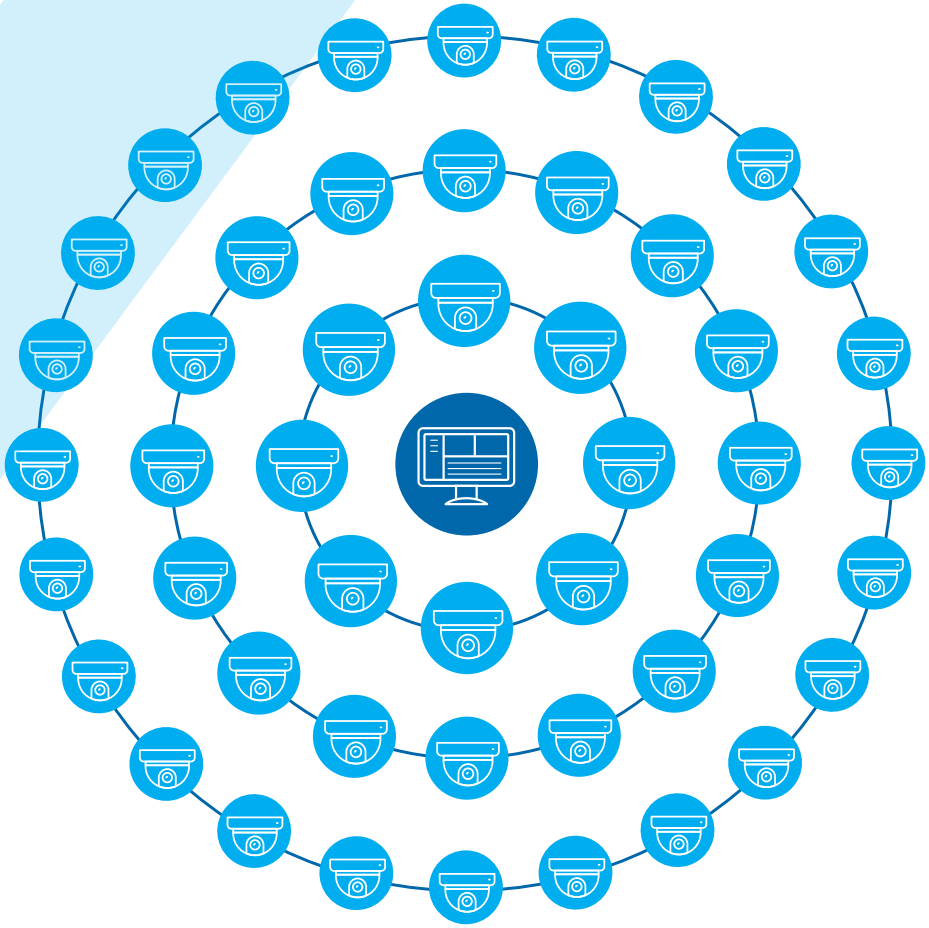


TECH TIP

Command Recording Software on Linux OS: a Tutorial

MARCH networks

Introduction

Starting from version 6.1, it's now possible to install March Networks Command Recording Software (CRS) on Linux machines. This new, highly scalable Linux version of our video management software (VMS) can support up to 3,000 cameras on a single server. This scalability is ideal for large corporate headquarters and other growing enterprises. By using this solution, customers can lower infrastructure costs with fewer servers. They can move toward centralized storage by streaming cameras to just one server in an on-site data center.

Linux OS is commonly considered as a better alternative to Windows OS in terms of cybersecurity, maintenance and longevity. At the same time, skills to be proficient with Linux procedures and methodologies are not so common in the security industry and a first approach could easily be frustrating for an inexperienced user. This document provides some guidelines on how to configure a basic system to test our CRS application. Configuring the full settings of a Linux server in a production environment is an advanced task, and should only be managed by experienced users.

This document serves as a guide to get a standard CRS for Linux installation completed, starting from OS installation and basic configuration. We'll use the virtualization approach, running the Linux OS in a virtualized environment, as this could be the very first step adopted by a user willing to test the application without requiring a dedicated machine.

Prerequisites

VirtualBox project, acquired by Oracle, is a general-purpose full virtualizer for x86 hardware and is the tool we'll use to achieve our goal. VirtualBox installer can be freely downloaded from the official website at: https://www.virtualbox.org/

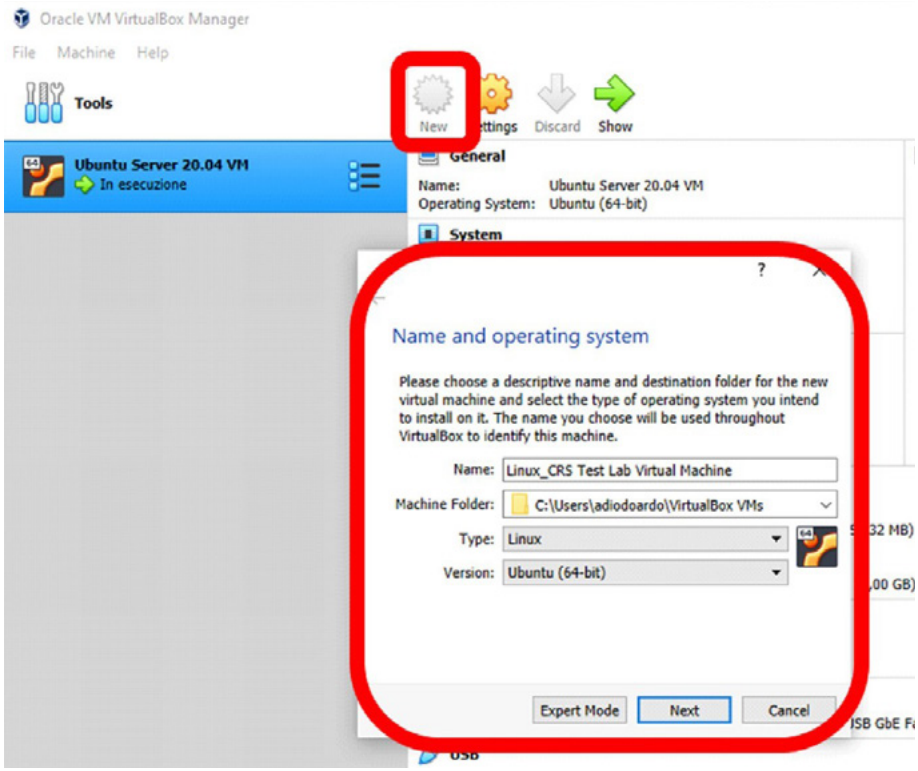
This tool is equivalent to VMWare and can create a virtual machine (VM) on a host PC. Most commonly, the host PC is running Windows OS (this is usually referred as the Host OS). In this case it will be necessary to select the Windows installer.

Once installed on a Windows PC, it's possible to launch the Virtual Box manager interface where it's possible to create and manage VMs.

At this point, to build a virtual Ubuntu machine (Ubuntu in this case is the Guest OS), it's necessary to download the ISO image of the target OS: Ubuntu Server 20.04 LTE edition represents a reasonable choice to test CRS on Linux (please note CRS on Linux only supports 64-bit OS). It's available for download from Ubuntu official website at: https://ubuntu.com

Once completed on the VBox interface, it's possible to select "New" to start the creation of a new VM. In the first popup panel, it's possible to specify the name — a destination folder to store all VM files and basic properties ("Type" and "Version" will reflect the Guest OS we want to install).

Figure 1: In Virtual Box management interface, once selected "New", a wizard is available to guide the user to create a virtual machine.

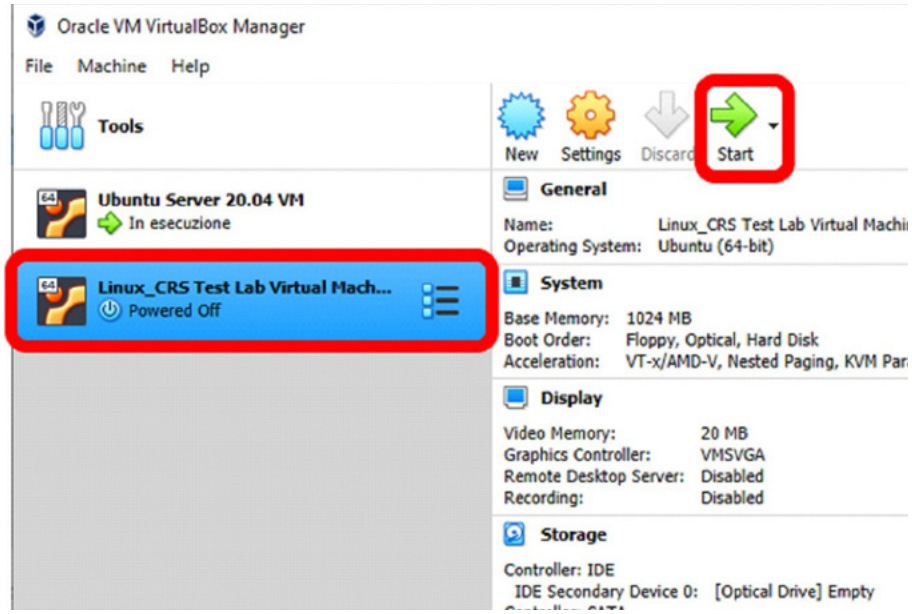


Hitting on "Next" will let us decide how much memory to assign to the VM: this should be less than 50% of the total RAM available in the host PC. Being a test environment, we could just allocate 2GB (it's possible to modify this value later if necessary). Similar considerations should be made for the number of CPU cores to assign to the VM: a minimum of 2 cores must be assigned to the new VM.

The next step is about the virtual hard disk used to store the Guest OS: we suggest you chose a minimum size of 20GB, leaving the default VDI option and default "Dynamically allocated" to reduce the initial size of the VM footprint on the host machine. Once a name is assigned to the hard disk, the system will create the virtual structure according to the selected options.

At this point, after few seconds the new VM will be listed in the left panel of the Management interface. Once selected, it's possible to start the empty VM that was just created.

Figure 2: Hitting on "Start" will launch the empty VM. Now it's time to upload the OS ISO image.



The first panel will ask for an ISO image to be used to install the Guest OS. We'll point to the downloaded Ubuntu ISO image and we will start the OS installation. From now on, the installation procedure is exactly the same as on a physical server. We'll need to select all of the standard settings for the new OS and ensure we choose the right keyboard interface (Ubuntu server is a terminal only OS and having a wrong keyboard layout can be problematic). Part of the procedure involves creating a user to access the system: just take a note of the credentials for future reference. It's also important to select a docker package during the procedure as this will be required to launch CRS (in case you miss it, it's still possible to manually install this package later).

At the end of the procedure, it's possible to wait for the system to upgrade all packages or you can skip this part and do it later. We now have a fresh VM running Ubuntu Server OS.

Configuring the Virtual Machine

Before starting the just created Ubuntu VM, we can add an additional hard disk to be used as a dedicated storage. On the management interface, opening Settings it's possible to select Storage and create a new hard disk under the SATA controller.

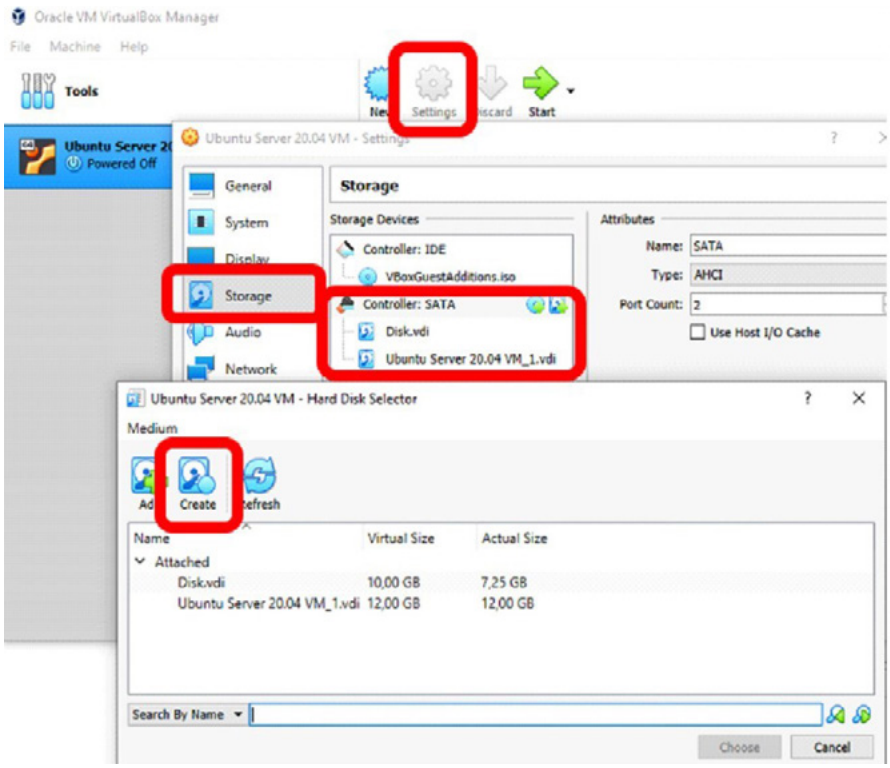


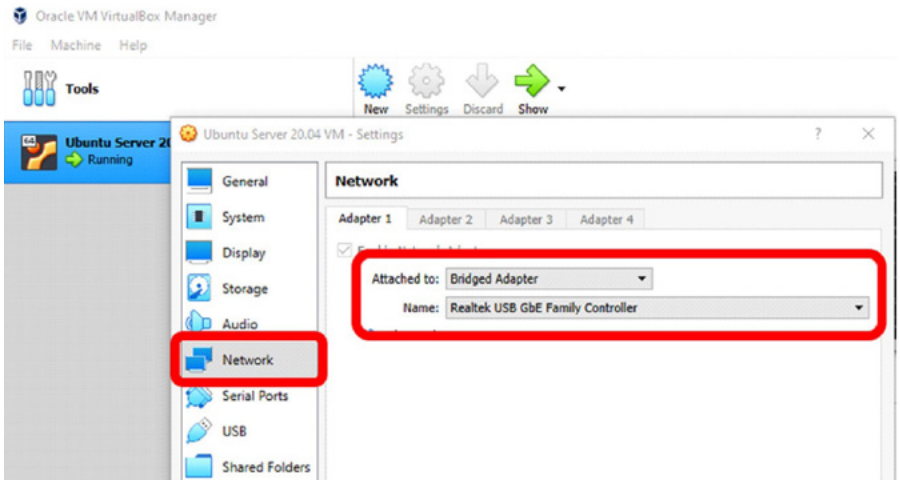
Figure 3: Adding a hard disk to the VM requires you to open the settings panel and navigate to the "Storage" section.

We'll leave the default VDI format for the new hard disk and we'll select "Fixed Size", because this offers superior performance. Once initialized by CRS, we'll use the full size (choose a value compatible with the storage available on the host PC).

The last step before approaching the Linux terminal console is to ensure we'll be able to assign a valid IP address to the VM as we'll likely need to connect cameras and the Client PC once we've completed the configuration.

Assuming a lab network with a DHCP server is available, we'll navigate to the "Network" tab and select "Bridged Adapter".

Figure 4: "Bridged Adapter" will ensure the same DHCP service used by the host PC will also assign a valid IP address to the VM.



Of course, it's necessary to select the controller connected to the local network hosting the DHCP service.

This is the last preliminary step. It's now time to start Ubuntu by hitting on the Start button.

Configuring the Operating System

Once you've logged in, the first step (assuming this was skipped during OS installation) is to upgrade all packages using the following command:

```
sudo apt-get upgrade
```

Once completed, it's recommended to create a CRS folder where we'll store CRS installation files and a CRS data folder where the installer will put all program data (configuration, logs, etc.). We'll create such folders in the home section using the command:

```
sudo mkdir CRS
sudo mkdir CRS_data
```

Once this is done, we can check with:

```
ls -la
```

to confirm the presence of the recently created folders.

Figure 5: Directories are listed in blue. Using the "-la" option gives additional details to the basic "ls" command.

```
diodo@diodovm:~$ ls -la
total 44
drwxr-xr-x 6 diodo diodo 4096 Jan 26 13:36 .
drwxr-xr-x 3 root  root  4096 Jan 25 11:40 ..
-rw------- 1 diodo diodo 1847 Jan 26 18:09 .bash_history
-rw-r--r-- 1 diodo diodo  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 diodo diodo 3771 Feb 25  2020 .bashrc
drwx------ 2 diodo diodo 4096 Jan 25 12:50 .cache
drwxr-xr-x 2 diodo diodo 4096 Jan 26 16:16 CRS
drwxrwxr-x 3 diodo diodo 4096 Jan 26 14:32 CRS_data
-rw-r--r-- 1 diodo diodo  807 Feb 25  2020 .profile
drwxr-xr-x 3 diodo diodo 4096 Jan 25 17:06 snap
-rw-r--r-- 1 diodo diodo    0 Jan 25 17:06 .sudo_as_admin_successful
-rw------- 1 diodo diodo  847 Jan 26 13:36 .viminfo
diodo@diodovm:~$
```

We'll also create a folder where we'll mount the storage hard disk. We'll create a dedicated folder under "/media" with the following command:

```
sudo mkdir /media/CRS_Storage
```

It's now time to mount the storage hard disk on the recently created folder.

It is necessary to identify the device name assigned by Linux to the storage hard disk. We'll use the following command:

```
sudo fdisk -l |less
```

Hitting a button will scroll to the next page (without "|less" you could miss the initial output of the command - hit q at the end of the list). You should identify a disk mapped on "/dev/sdb" which needs to be initialized. This disk should have the same size you specified before.

We'll then run the command:

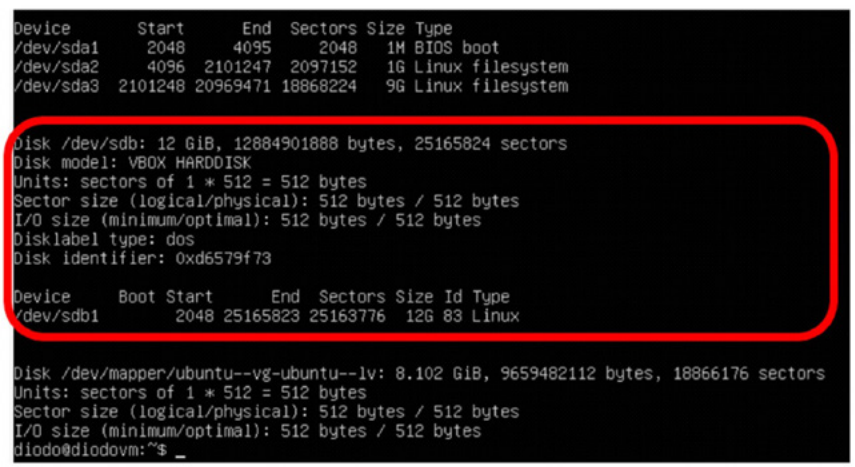
```
sudo fdisk /dev/sdb
```

This will open the procedure to create a new partition on the disk. Select "n" for new partition, "p" for primary partition and leave the other settings as default. Press "w" to save all settings on the disk. Assuming the procedure is successful, we can try again with:

```
sudo fdisk -l |less
```

This will verify that a new section is now present.

Figure 6: Disk partition has been created and it now appears as the output of the fdisk command.



```
Device       Start      End  Sectors Size Type
/dev/sda1     2048     4095     2048   1M BIOS boot
/dev/sda2     4096  2101247  2097152   1G Linux filesystem
/dev/sda3  2101248 20969471 18868224   9G Linux filesystem

Disk /dev/sdb: 12 GiB, 12884901888 bytes, 25165824 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xd6579f73

Device     Boot Start      End  Sectors Size Id Type
/dev/sdb1        2048 25165823 25163776  12G 83 Linux

Disk /dev/mapper/ubuntu--vg-ubuntu--lv: 8.102 GiB, 9659482112 bytes, 18866176 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
diodo@diodovm:~$ _
```

It's now possible to format the recently created partition. We'll use "xfs" file system:

```
sudo mkfs.xfs /dev/sdb1
```

Now we want to ensure this partition is auto-mounted at boot. We'll need to edit the "fstab" configuration file in the "etc" folder. Before applying any change to this file, we'll create a backup:

```
sudo cp /etc/fstab /etc/fstab.backup
```

Then, using vi (or any other text editor - nano for instance) we'll add two lines at the end of the "fstab" file:

```
sudo vi /etc/fstab
```

Using arrows, it's possible to position the cursor on the last line. At this point, hitting letter "o" will add a new line after the cursor position and the mode will switch to INSERT. It's now possible to type the required text (first line is just a comment, in the second line use TAB and not SPACE to separate each field):

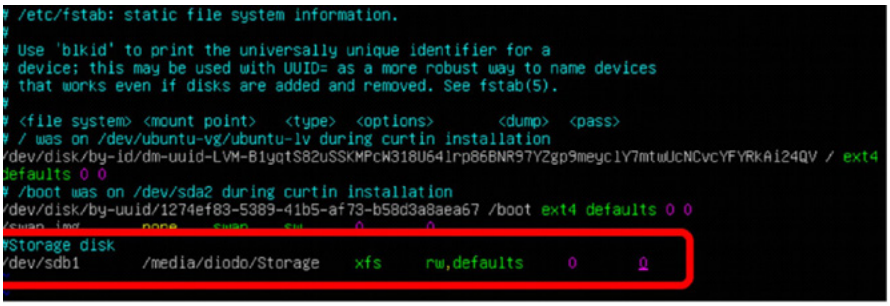
```
#CRS Storage disk
/dev/sdb1/media/CRS_Storagexfsrw,defaults00
```

To save and close the file hit ESC to exit from INSERT mode, then type:

```
:wq!
```

to exit from the editor.

Figure 7: We'll need to add two lines (first one is only a comment) to have the new partition mounted at boot.



It's now possible to finally mount the disk with the command:

```
sudo mount -a
```

To verify the procedure, use the command:

```
sudo cat /etc/mtab | less
```

Then search for a line listing /dev/sdb1.

We're quickly arriving at the end of the procedure, but before being able to execute the CRS installation, it's a good idea to reboot the machine to update all the settings using the command:

```
sudo shutdown -P now
```

Once the machine has rebooted, we need to copy the installation files in the Ubuntu PC. There are several ways to transfer files between a Windows PC and a Linux one. We'll use a third-party tool called WinSCP, freely downloadable from the Internet, to quickly move files between the host and the guest systems.

For this to work, we need to discover the IP address of the VM machine. We'll use the command:

```
ip a
```

Figure 8: Discovering the IP address is necessary to connect to the system and transfer files.

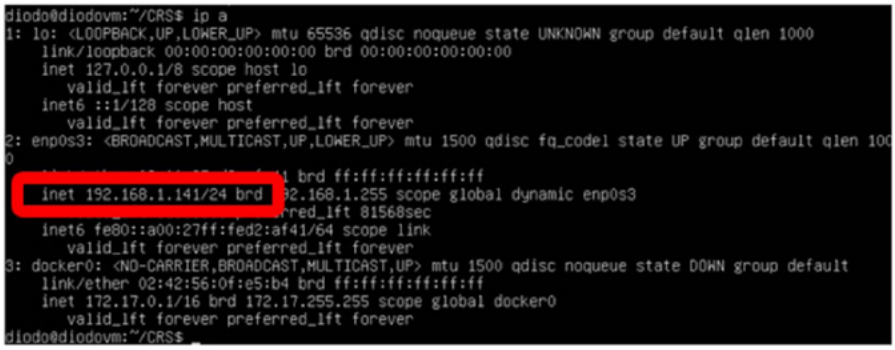
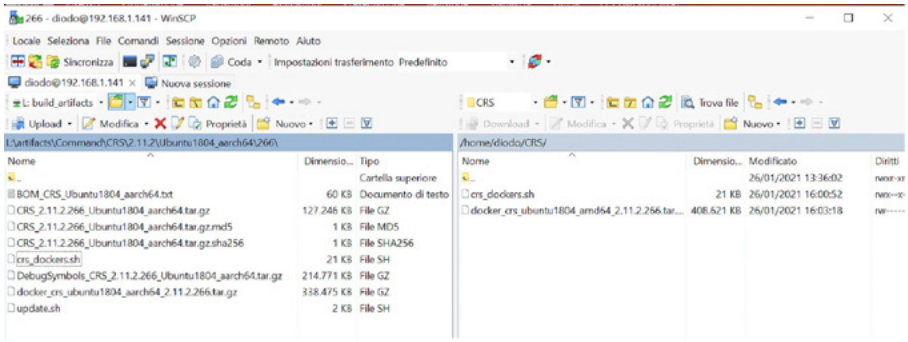


Figure 9: We'll need to add two lines (first one is only a comment) to have the new partition mounted at boot.

Once a new connection is established (using SCP protocol and the credentials of the Linux user) we'll transfer the required files.



In the Linux console, it's necessary to make the script executable with the command:

```
sudo chmod a+x crs_dockers.sh
```

Using again the command "`ls-la`" it's possible to verify the script will be listed in green and the abilities will include the x (execute).

Figure 10: It's now possible to execute the script once it has been made executable with the "chmod" command.

```
diodo@diodovm:~$ cd CRS
diodo@diodovm:~/CRS$ ls -la
total 408660
drwxr-xr-x 2 diodo diodo      4096 Jan 26 16:16 .
drwxr-xr-x 6 diodo diodo      4096 Jan 26 13:36 ..
-rwx--x--x 1 diodo diodo     21356 Jan 26 16:00 crs_dockers.sh
-rw------- 1 diodo diodo 418427770 Jan 26 16:03 docker_crs_ubuntu1804_amd64_2.11.2.266.tar.gz
diodo@diodovm:~/CRS$
```

We're now ready to launch the script to create the first docker instance of CRS. We recommend testing a single instance the first time, just to ensure the system is properly configured (most likely it would be necessary to increase the memory and the CPU cores in case of multiple instances running in parallel).

Running our First CRS Instance

Depending on the script version, the syntax could be slightly different. It's a good idea to launch the inline help to see the list of available options. As a reference, considering version 2.11.2, the command line to create a single docker instance (in the Italian time zone using the storage and data folders previously created) we'd execute the following:

```
sudo ./crs_dockers.sh -C -r 1 -i docker_crs_ubuntu1804_
amd64_2.11.2.266.tar.gz -f /home/ diodo/CRS_data -t
Europe/Rome -s /media/CRS_Storage
```

Using docker commands, it's possible to check crs1 instance has been properly created and it's running fine.

Figure 11: sudo docker ps will list the active instances currently running on the machine. Pay particular attention to the STATUS column.

```
diodo@diodovm:~/CRS$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
    PORTS               NAMES
e4b9ae927971        crs:current         "/bin/sh -c ./run.sh"    3 hours ago         Up 53 minutes
                        crs1
```

Figure 12: sudo docker logs will list all the logs generated by each instance. This is where to start in case of a system failure.

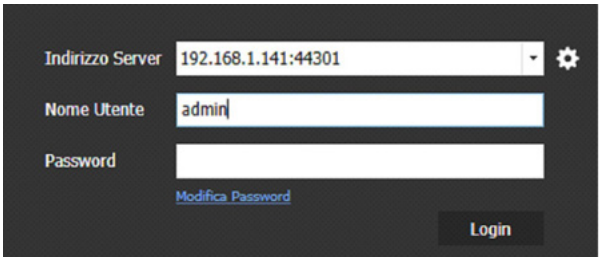
```
diodo@diodovm:~/CRS$ sudo docker logs
"docker logs" requires exactly 1 argument.
See 'docker logs --help'.

Usage:  docker logs [OPTIONS] CONTAINER

Fetch the logs of a container
diodo@diodovm:~/CRS$ sudo docker logs crs1
Starting CRS Docker container...
- server_data loading...
- reg_data loading...
- enabling core dumps generation (requires "/var/crash/core_%e_%p" set from host)
Starting CRS...
./plugins/libUbuntu1804_amd64_DVR.so: undefined symbol: _ZN14HSCCMInterface8SendTextEiPKci
./plugins/libUbuntu1804_amd64_DVR.so: undefined symbol: _ZN14HSCCMInterface8SendTextEiPKci
Starting CRS Docker container...
- server_data loading...
- reg_data loading...
- enabling core dumps generation (requires "/var/crash/core_%e_%p" set from host)
Starting CRS...
./plugins/libUbuntu1804_amd64_DVR.so: undefined symbol: _ZN14HSCCMInterface8SendTextEiPKci
./plugins/libUbuntu1804_amd64_DVR.so: undefined symbol: _ZN14HSCCMInterface8SendTextEiPKci
diodo@diodovm:~/CRS$ _
```

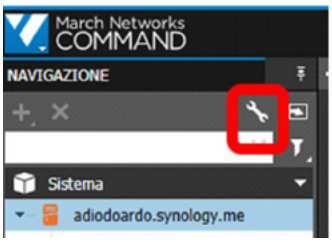
At this point it's possible to launch Command Client on the host machine (or any other workstation in the same LAN) and open the Linux IP address specifying port 44301 for the HTTPS protocol.

Figure 13: By default, admin user has no password. The https port is 44301 and can be changed during docker creation.



By selecting the server name, it's possible to launch the system configurator and, as a first step, register the recorder to CES to get licenses. By design, this is the only supported way to license CRS on Linux.

Figure 14: A quick way to launch CRS configuration interface is to click on the tool from the Client Interface.



From now on, the system will behave exactly like the more traditional Windows version and it's possible to refer to the manuals and online training material for detailed instructions on how to configure the recorder.

Conclusion

CRS is a very powerful and flexible software which can be installed on many different hardware platforms to match size, cost and expected performance. Traditionally, this has only been possible using Windows-based Operating Systems. With version 6.1, we now also support Linux OS and docker container technology. This will let users scale the solution to manage up to several thousand cameras under the same physical or virtual machine (simply creating multiple instance of the same CRS image). You could also be tempted to install CRS on your Raspberry to have a small form factor embedded system equipped with March Networks technology. Feel free to reach out to your March Networks sales representative for further details on how to start your journey with CRS on Linux.

Company Overview

March Networks® helps organizations transform video into business intelligence through the integration of surveillance video, analytics, and data from business systems and IoT devices. Companies worldwide use our software solutions to improve efficiency and compliance, reduce losses and risk, enhance customer service and compete more successfully. With deep roots in video security and networking, March Networks is also recognized as the leader in scalable, enterprise-class video management and hosted services. We are proud to work with many of the world's largest financial institutions, retail brands, cannabis operators and transit authorities, and deliver our software and systems through an extensive distribution and partner network in more than 70 countries. Founded in 2000, March Networks is headquartered in Ottawa, Ontario, Canada. For more information, please visit **www.marchnetworks.com**.

Customer Support and Assistance

North America, South America, & Asia Pacific

Direct: 1.613.591.1441

Toll Free (US & Canada): 1.800.472. 0116

E-mail: techsupport@marchnetworks.com

Europe, Middle East & Africa

Direct: +39 0362 17935 ext. 3

E-mail: supporteurope@marchnetworks.com

© 2021 March Networks. All rights reserved. Information in this document is subject to change without notice. MARCH NETWORKS, March Networks Command, March Networks Searchlight, March Networks RideSafe, and the MARCH NETWORKS logo are trademarks of March Networks Corporation. All other trademarks are the property of their respective owners. 060-3462-00-A