Oracle Private Cloud Appliance X9-2 Workload Import

How to migrate virtual machine workloads to PCA X9-2

November 9, 2022 | Version 1.06

PURPOSE STATEMENT

This document provides a guide to importing virtual machine workloads to Oracle Private Cloud Appliance X9-2 (PCA) with software version release 3.0.1. It is intended to help you plan migration to PCA 3.0.1 and later from other cloud and virtualization platforms.

DISCLAIMER

TABLE OF CONTENTS

INTRODUCTION

This Technical Brief provides best practices and guidance for importing virtual machines from legacy hypervisor platforms to Oracle Private Cloud Appliance X9-2 (abbreviated to "PCA" in this document except when distinguishing from previous PCA versions) in "lift and shift" migration.

The Oracle Private Cloud Appliance (PCA) has emerged as the premier platform for WebLogic, Fusion Middleware and other application tier software, often in conjunction with Oracle Exadata - the premier database platform. Many of those applications have been deployed on other servers and are now being deployed on PCA for increased performance, scale, and manageability.

SCOPE AND CONTENTS

This document provides a methodology and workflow that architects and system administrators can use to migrate virtual machine workloads to PCA X9-2, including:

- Planning steps applicable for any migration to PCA
- Details specific to migration from specific platforms, in particular Oracle VM, Private Cloud Appliance X8-2 and earlier, Oracle VM VirtualBox, KVM, and VMware
- Details for different operating systems, with instructions for Oracle Linux and other Linux distributions, Microsoft Windows, and Oracle Solaris
- Description of how to move workloads between Oracle Cloud Infrastructure (OCI) and PCA

This document illustrates manual methods for moving virtual machines running different operating systems on simple VM configurations. It does not cover bare-metal instance migration or complex configurations or automate changes to virtual machine operating system kernels or configurations.

For full migration scenarios, we recommend planning with your Oracle team, and evaluation of enterprise-grade solutions from Oracle partners like Rackware, Coriolis, or Commvault.

ADVANTAGES OF PRIVATE CLOUD APPLIANCE

Oracle Private Cloud Appliance (PCA) is an Oracle Engineered System designed for implementing the application and middleware tiers. PCA is an integrated hardware and software system that reduces infrastructure complexity and deployment time for virtualized workloads in private clouds. It is a complete platform for a wide range of application types and workloads, with built-in management, compute, storage, and networking resources. PCA provides excellent performance and other system properties for hosting a broad range of applications.

Oracle Private Cloud Appliance X9-2 is the latest member of the Oracle Private Cloud Appliance product family. PCA provides cloud and administrative services for a supporting range of workloads including modernized cloud native applications. It makes use of a modern microservices architecture, Kubernetes, and related technologies, for a future-proofed software stack.

A key new feature of PCA X9-2 compared to previous PCA versions, is that it delivers private cloud infrastructure and architecture consistent with Oracle Cloud Infrastructure (OCI). PCA brings APIs and SDKs compatible with Oracle Cloud Infrastructure (OCI) to an on-premises implementation at rack scale, making workloads, user experience, tool sets and skills portable between private and public clouds. PCA can be paired with Oracle Exadata to create an ideal infrastructure for scalable, multi-tier applications. Customers preferring or requiring an on-premises solution can realize the operational benefits of public cloud deployments using Oracle Private Cloud Appliance X9-2.

DEFINITIONS

This document uses the following terms:

- The ***source*** system is the platform where the virtual machine is currently running, and the ***target*** is the PCA system it will be moved to.
- An ***instance*** is a virtual machine on the PCA. An instance has a ***lifecycle***: it can be created, started ("launched"), stopped, and terminated (removed from the PCA system). Oracle Cloud Infrastructure (OCI) uses the term instance instead of virtual machine because an instance could potentially be on bare metal. PCA conforms to the OCI definition.
- Every instance has a ***shape***, which describes its CPU, memory, network, and disk configuration. PCA has a list of **standard shapes** shown in Appendix 1. Instances can also be configured with **flexible shapes** that permit customized CPU, memory, and network resources.
- An ***image*** is the template of a virtual disk, containing the operating system and installed applications, plus descriptive metadata.
- An instance is created from an image, which contains the ***boot volume***, specification of share, network configuration and other parameters, and immediately launched. Other disks belonging to the instance are called ***block volumes*** and are created and attached to the instance after the instance is launched. Block volumes can be assigned to a single instance or can be shared by several instances.
- PCA provides ***platform images*** for Oracle Linux 7, Oracle Linux 8, and Oracle Solaris 11.4.
- ***Custom images*** can be created from an instance on PCA, making it possible to use an image as the basis for cloned instances with customized contents. Custom images can be copied to ***object storage*** and exported from PCA.
- ***Bring Your Own Image (BYOI)*** images are imported from different platforms and can be based on any of the supported operating systems.

For further description of these terms, see the PCA Concepts Guide sections 7.1 (Compute Services) and 7.2 (Compute Images).

This document focuses on BYOI using a "lift and shift" approach in which an OS instance is moved to PCA with as few OS and configuration changes as possible, It focuses on steps needed to make an OS instance boot and operate on PCA X9-2 with details based on the operating system and source platform. Application migration is referred to but out of scope.

> **Note**: For purposes of this document, instructions for Linux mostly apply to Oracle Linux, Red Hat and other Red Hat based distributions. However, Ubuntu, Debian and SUSE have been successfully tested.
>
> **Note**: PCA instances are created from images already loaded into the PCA system, not directly from OS installation media (e.g.: USB or DVD ISO images). You can create a virtual machine from installation media on a separate hypervisor and follow the steps in this guide to create a PCA image. Oracle VM VirtualBox is a convenient platform for this, or you could use a different hypervisor you already have.

Another approach is to create entirely new images based on the latest version of the desired operating system and application software. This is common in "technology refresh" situations where the entire hardware and software stack is modernized. Both approaches are valid and widely used, depending on an institution's business and technical needs.

LICENSING

PCA provides platform images for Oracle Linux 7, Oracle Linux 8, and Oracle Solaris 11.4

> **Note**: You must comply with all licensing requirements when you upload and start instances based on OS images that you supply.

MIGRATION WORKFLOW

Moving to PCA X9-2, as with movement to any platform, is like a datacenter migration. It requires planning and multiple operational steps. The workflow contains these steps:

1. Preparation: perform data collection on source system.
2. Preparation: update instances on source system before they are exported.
3. Export: make the virtual machine boot disk image available outside the source hypervisor environment.
4. Transformation: convert the boot disk image for operation with PCA X9-2, if needed.
5. Import: upload the custom image into PCA.
6. Create: launch instance from the imported image.
7. Post-install: data migration and configuration.
8. Go live: Parallel test and switchover.

PREPARATION: DATA COLLECTION

The first phase of migrating workloads to PCA X9-2 is to collect information about VMs to be moved. This is done using the source platform's management tools or directly within each source virtual machine.

Collect the information described in the following sections. Some parts require logging into the VM. Others can be gathered from the hypervisor environment or management framework such as VMware vSphere or Oracle Enterprise Manager. Some information is not needed immediately but will be needed later to complete migration. Commands mentioned below are not intended to be exhaustive. A system administrator for each source environment can easily find those details

VM Shape

Collect the number of cores and memory size assigned to the source VM. This determines the PCA shape needed for the VM's requirements. You can select either a standard shape with a 1:16 ratio of cores to RAM, or a flexible shape that lets you customize the number of cores and amount of memory. See Appendix 1 for a list of PCA shapes and their resources.

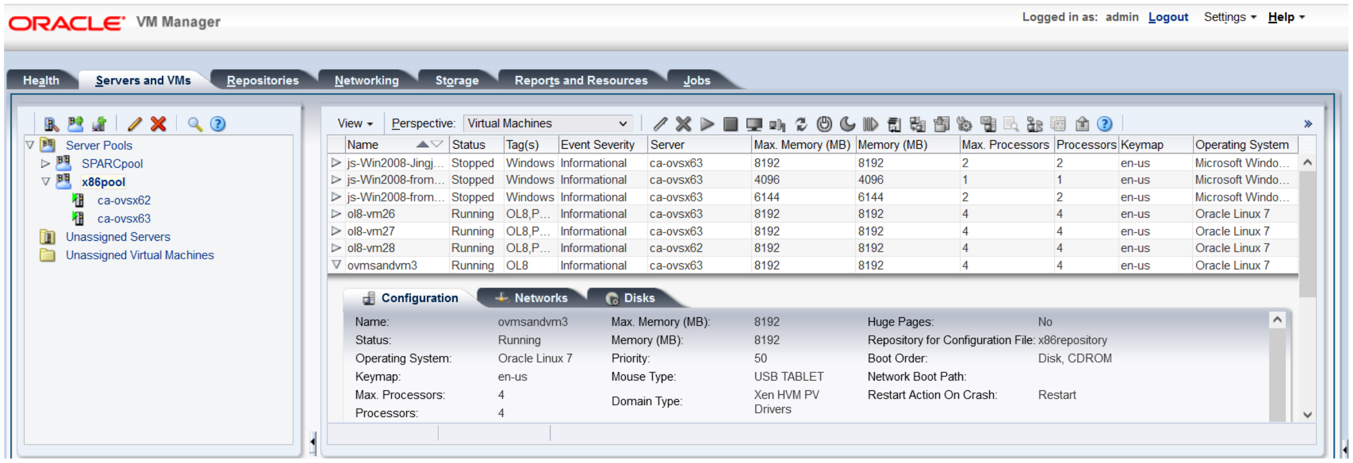
Select a shape that meets the instance's capacity needs, adjusting for CPU speed and resource utilization, and accommodates expected growth. You may be able to select a smaller shape if the source VM is over-provisioned or on a processor slower than PCA's X9-2 servers. That would help maximize the number of instances that can be hosted.

> **Note:** The shape also determines the maximum number of virtual NICs the instance can have. You may need a larger shape than the CPU and memory requirements dictate. See Appendix 1 for details.

> **Note:** If you are counting CPU threads, also called virtual CPUs or vCPUs in virtual machines, on a hyper-threaded server, divide the number of threads by two and round up to the number of CPU cores (also called OCPUs).

Obtaining shape information from the source platform

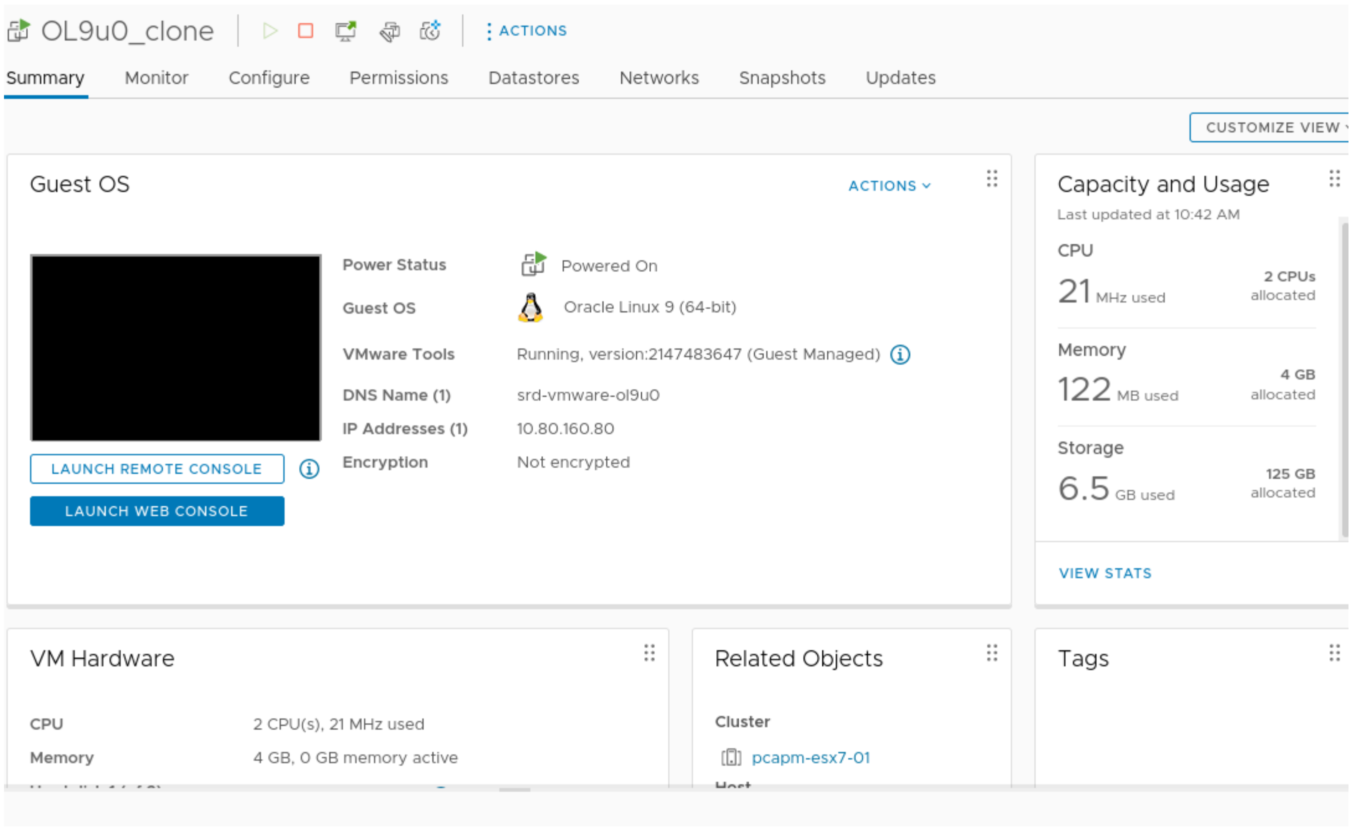
Use tools in the source platform and guest operating system to collect this information. For example, with PCA 2 and Oracle VM, the VM's shape is visible from the Oracle VM Manager browser interface, or by issuing the Oracle VM CLI command "`show vm name=VMNAME`". Oracle VM VirtualBox provides this information with the graphical interface or by using the command "`vboxmanage showvminfo VMNAME`" . VMware vCenter provides this information in its graphical console. On KVM use `virt-manager` or the command "`virsh dominfo VMNAME`" .



Name	Status	Tag(s)	Event Severity	Server	Max. Memory (MB)	Memory (MB)	Max. Processors	Processors	Keymap	Operating System
js-Win2008-Jingj…	Stopped	Windows	Informational	ca-ovsx63	8192	8192	2	2	en-us	Microsoft Windo…
js-Win2008-from…	Stopped	Windows	Informational	ca-ovsx63	4096	4096	1	1	en-us	Microsoft Windo…
js-Win2008-from…	Stopped	Windows	Informational	ca-ovsx63	6144	6144	2	2	en-us	Microsoft Windo…
ol8-vm26	Running	OL8,P…	Informational	ca-ovsx63	8192	8192	4	4	en-us	Oracle Linux 7
ol8-vm27	Running	OL8,P…	Informational	ca-ovsx63	8192	8192	4	4	en-us	Oracle Linux 7
ol8-vm28	Running	OL8,P…	Informational	ca-ovsx62	8192	8192	4	4	en-us	Oracle Linux 7
ovmsandvm3	Running	OL8	Informational	ca-ovsx63	8192	8192	4	4	en-us	Oracle Linux 7

Name:	ovmsandvm3	Max. Memory (MB):	8192	Huge Pages:	No
Status:	Running	Memory (MB):	8192	Repository for Configuration File:	x86repository
Operating System:	Oracle Linux 7	Priority:	50	Boot Order:	Disk, CDROM
Keymap:	en-us	Mouse Type:	USB TABLET	Network Boot Path:	
Max. Processors:	4	Domain Type:	Xen HVM PV Drivers	Restart Action On Crash:	Restart
Processors:	4				

Figure 1. Display VM shape from Oracle VM Manager (applies to PCA 2.x)



Power Status	Powered On
Guest OS	Oracle Linux 9 (64-bit)
VMware Tools	Running, version:2147483647 (Guest Managed)
DNS Name (1)	srd-vmware-ol9u0
IP Addresses (1)	10.80.160.80
Encryption	Not encrypted

Capacity and Usage
Last updated at 10:42 AM

CPU	21 MHz used	2 CPUs allocated
Memory	122 MB used	4 GB allocated
Storage	6.5 GB used	125 GB allocated

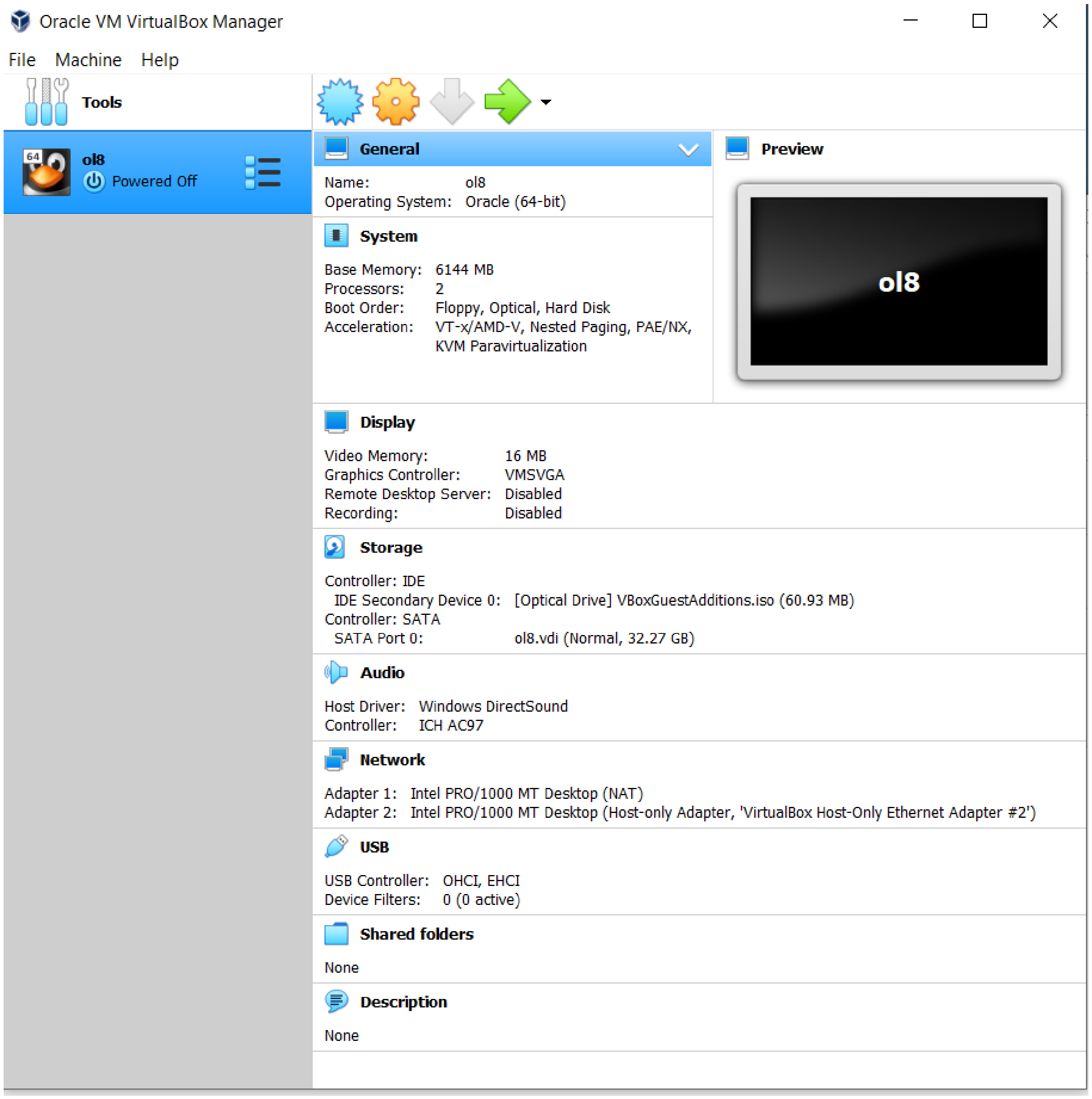
VM Hardware

CPU	2 CPU(s), 21 MHz used
Memory	4 GB, 0 GB memory active

Related Objects

Cluster
pcapm-esx7-01

Figure 2. Display VM shape information from VMware vCenter



General

Name: ol8
Operating System: Oracle (64-bit)

System

Base Memory: 6144 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Device 0: [Optical Drive] VBoxGuestAdditions.iso (60.93 MB)
Controller: SATA
SATA Port 0: ol8.vdi (Normal, 32.27 GB)

Audio

Host Driver: Windows DirectSound
Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)
Adapter 2: Intel PRO/1000 MT Desktop (Host-only Adapter, 'VirtualBox Host-Only Ethernet Adapter #2')

USB

USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)

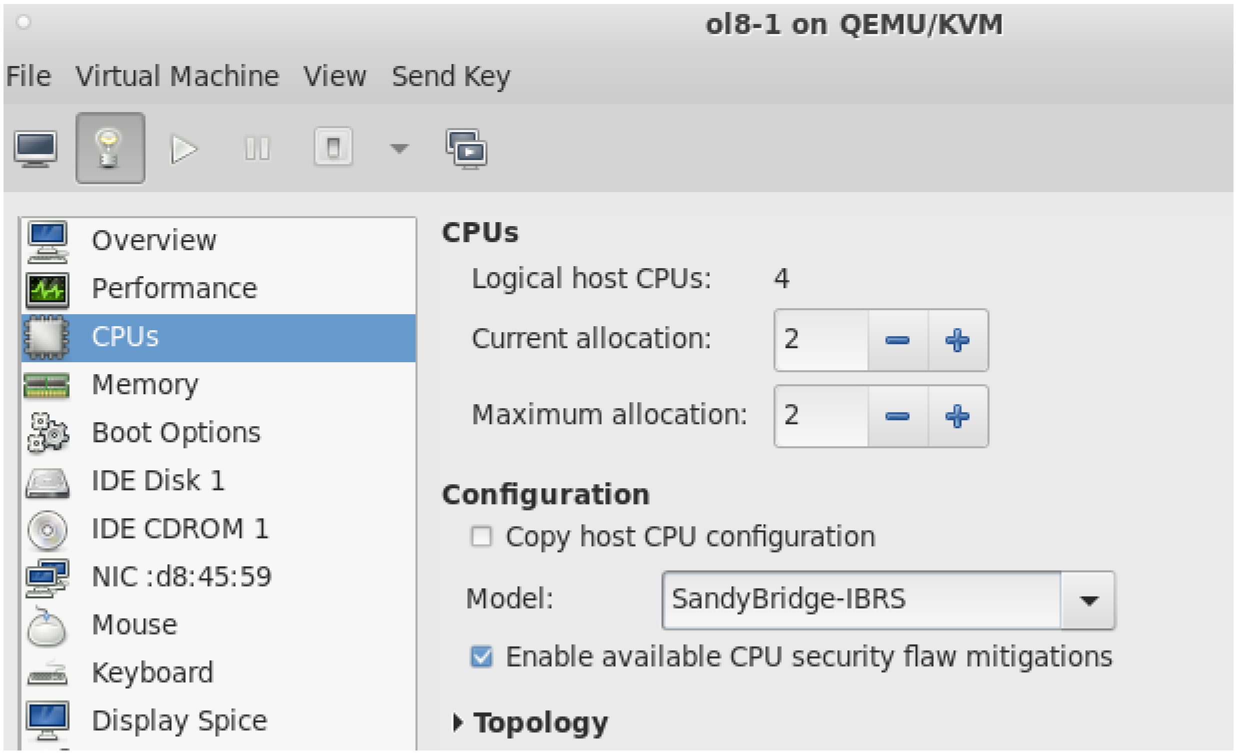
Shared folders

None

Description

None

Figure 3 – Display VM shape information from Oracle VirtualBox


ol8-1 on QEMU/KVM

File Virtual Machine View Send Key

CPUs

Logical host CPUs: 4

Current allocation: 2

Maximum allocation: 2

Configuration

☐ Copy host CPU configuration

Model: SandyBridge-IBRS

☑ Enable available CPU security flaw mitigations

▸ **Topology**

Figure 4. Display VM shape information from KVM virt-manager

Obtaining shape information from the guest operating system

You can also obtain shape information within the guest operating system:

- On Linux, issue commands: `awk '($1 == "processor")' /proc/cpuinfo | wc -l ; mpstat -P ALL` for CPU count, and "`free -h`" or "`cat /proc/meminfo |grep MemTotal`" for memory size.
- On Oracle Solaris issue "`psrinfo -vp`" for CPU count and "`prtconf|grep Mem`" for memory size.
- On Windows, depending on the version, right click on This PC or select Settings->System→About or use Control Panel, or enter Task Manager to get the number of CPUs and memory size.

Storage Configuration

Collect the number and sizes of disks owned by the VM.

> **Note**: This document describes moving the boot disk, referred to as the boot volume on PCA. Disks containing data, swap, `/tmp`, application binaries are not transferred as part of this process. Their sizes will be needed after the boot volume is transferred and the instance is created. Data disks are referred to as *block volumes* and are defined and populated after the instance is created.

As with CPU and memory size this can be determined from the source platform hypervisor. In PCA 2 and Oracle VM the details are visible from Oracle VM Manager browser interface and CLI as described above. The graphical tools for VMware, Oracle VirtualBox, KVM and other hypervisors provide similar access.

Alternatively, from within the VM:

- On Linux issue "`df -h; lsblk; mount`".
- On Solaris issue "`df -h; zpool list; zpool status`".
- On Windows, use the same method as above for CPU and memory.

Filesystem mounts (NFS and SMB/CIFS) and iSCSI block storage

Though not part of the instance definition, you will need details on NFS and SMB mounts and iSCSI targets accessed by the VM. If the new platform has access to the same storage platform they may still be used. Use the same "within the VM" commands as in the Storage configuration section above.

> **Note**: Changing the virtual machine will change its iSCSI IQN, and continued access to the same iSCSI block storage will require adjustment within the VM instance and at the iSCSI SAN storage layer.

Network Configuration

For each virtual NIC in the VM, collect its IP address, netmask, MTU, routing, DNS and NTP settings. The list below is not intended to be exhaustive, as system administrators have multiple tools to obtain this information.

- On Linux `use ifconfig -a; ip a; netstat` commands and `'cat /etc/resolv.conf'`. You can also use graphical tools: on GNOME you can use the desktop navigation System→Preferences→Internet and Network→Network Connections and then display IP settings. On Oracle Linux 8 or 9 you can use the `nmtui` or `nmcli` commands to view and administer network settings.
- On Solaris 11.4 use `dladm show-link; ipadm show-if; ipadm show-addr`, and `netstat` comments.
- On Windows, use the `ipconfig` command or display via Control Panel/Settings

PREPARATION: PRE-EXPORT UPDATES

It is advisable and sometimes necessary to prepare a VM for PCA by making changes to the VM on the original source system.

Operate on a clone of the production image on the source system. This protects the production environment and provides a known fallback if the process needs to be repeated. Use a method provided by your source system for cloning a VM. For example, in Oracle VM and PCA 2.4, select the VM in the Servers and VMs tab, right click and select Clone. The source VM should be stopped to ensure that the clone has consistent disk contents.

Perform the following steps on the clone, not the original VM. **Use a clone!**

Patch to current level of the OS

Update the clone VM to the current patch level of the same OS. This ensures that the most recent device drivers are available and prevents risk from recently closed bugs and security exposures.

The procedure depends on the guest operating system. For example, 'yum update' to the latest patch level for Oracle Linux 7 or 8, or 'pkg update' on Solaris 11.4, or Windows Update on Microsoft Windows. Reboot after patching to ensure proper operation post-patching.

Change virtual network devices to DHCP

Several changes may be needed to the guest VM's virtual network for it to work after being installed on PCA.

PCA X9-2 uses DHCP to provide a network address to the new instances. Since multiple instances can be cloned from the same image, it makes sense to assign new MAC and IP addresses when each instance is created.

> **Note**: The IP address acquired by DHCP when the instance is created is stable.
> **Note:** the IP address visible to the instance is a private IP address on a virtual cloud network (VCN). The instance is accessed from outside the PCA via a Network Address Translation (NAT) layer. See the Network tab in the Compute Enclave's instance description to see both private and public IP addresses.

On Oracle Linux 7 and 8 and equivalent, edit the file `/etc/sysconfig/network-scripts/ifcfg-NICNAME` and remove `HWADDR` or `MACADDR` lines and set `BOOTPROTO="dhcp"`. On Oracle Linux 8 or 9 you can use the `nmtui` or `nmcli` commands to administer network settings. Alternatively, you can use the graphical tool provided with your distribution. Additionally, remove the files `/etc/udev/rules.d/70-persistent-ipoib.rules` and `/etc/udev/rules.d/70-persistent-net.rules` if present. Those files contain MAC addresses that won't match the new instance, and are recreated automatically if needed.

On Solaris use the `ipadm` command, with a sequence of commands like

```
# ipadm delete-addr net0/v4
# ipadm create-addr -T dhcp net0/v4
```

On Windows, use the Control Panel or Settings.

Remove VMware, Oracle VM, or Oracle VM VirtualBox tools/drivers

If you are migrating from VMware or Oracle VM or VirtualBox, remove associated paravirtualization drivers and tools from the clone. For example, you can remove the Oracle VM Paravirtualization (PV) drivers for Microsoft Windows by locating the drivers in Control Panel, then right-clicking and selecting Uninstall. See https://docs.oracle.com/en/virtualization/oracle-vm-pv-drivers/3.4.5/winpv/vmwpv-uninstall.html for further details.

The image will work with the tools and drivers installed, though boot may take longer, and you may see error messages at boot time or when logging in, so this is optional.

Install kernel and driver features for PCA

The operating system may need to have VirtIO device drivers installed and included at boot time. The method for this depends on the operating system.

Linux

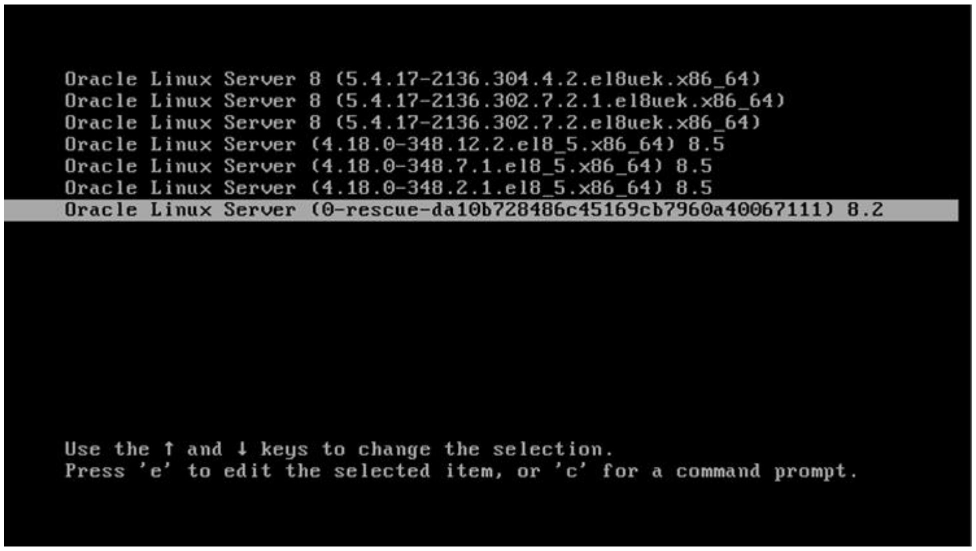
Use the `dracut` command to add device drivers to `initramfs`: "`dracut --force -N`" or "`dracut -N --regenerate-all`". It is installed by default on Oracle Linux. You may need to add it in other distributions. For example, on Ubuntu, issue the command "`sudo apt install dracut-core`". Optionally backup kernels before running `dracut`, e.g.: `mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).img.bak`

- The `-N` option disables 'hostonly' mode, which creates an `initramfs` only for the current host. This option produces a generic image.
- If you are running an old version of Linux that does not have the "-N" option, list the kernel modules needed for PCA: "`dracut --add-drivers 'sg sd_mod virtio virtio_pci virtio_ring virtio_scsi'`"
- The `--regenerate-all` option regenerates `initramfs` for all kernel versions in `/boot`, not just the one currently booted. This is useful for Oracle Linux to ensure that both UEK and RHCK kernels get the needed drivers, regardless of which is currently booted
- The option `--force` overlays the current `initramfs` file already present.

You can see the updated `initramfs` files in `/boot` They will be larger than previous versions because of the added device drivers.

Note: If the `dracut` step is omitted, it can be performed the first time the instance is launched on PCA X9-2: Log into the Compute Enclave, select the instance, connect to the console, and select the rescue line in the

Grub menu. If boot fails before you can select the rescue boot, issue a Reset action on the instance from the Compute Enclave. After the instance boots, login and issue the `dracut` commands described above.



```
Oracle Linux Server 8 (5.4.17-2136.304.4.2.el8uek.x86_64)
Oracle Linux Server 8 (5.4.17-2136.302.7.2.1.el8uek.x86_64)
Oracle Linux Server 8 (5.4.17-2136.302.7.2.el8uek.x86_64)
Oracle Linux Server (4.18.0-348.12.2.el8_5.x86_64) 8.5
Oracle Linux Server (4.18.0-348.7.1.el8_5.x86_64) 8.5
Oracle Linux Server (4.18.0-348.2.1.el8_5.x86_64) 8.5
Oracle Linux Server (0-rescue-da10b728486c45169cb7960a40067111) 8.2










Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

Figure 5. Grub menu showing available kernels, and highlighting rescue kernel

Solaris

Update Solaris 11.4 to the latest SRU by using the "`pkg update`" command and booting into the new boot environment. VirtIO drivers in Solaris 11.4 are described in the blog "Introducing VirtIO Drivers with Oracle Solaris on OCI" at https://blogs.oracle.com/solaris/post/introducing-virtio-drivers-with-oracle-solaris-on-oci

Solaris 11 users are strongly urged to upgrade to Solaris 11.4, which is an easy procedure and highly compatible. Move Solaris 10 environments by creating Solaris 10 branded non-global zones in a Solaris 11.4 instance. That lets the Solaris 10 environment benefit from the modernized kernel and device drivers. For further information see `https://blogs.oracle.com/oracle-systems/post/solaris-and-non-global-zones-on-oracle-private-cloud-appliance-x9-2`

Windows

Add the VirtIO drivers needed to operate Windows, as described in User Guide section 5.3.2. The release number shown below may change.

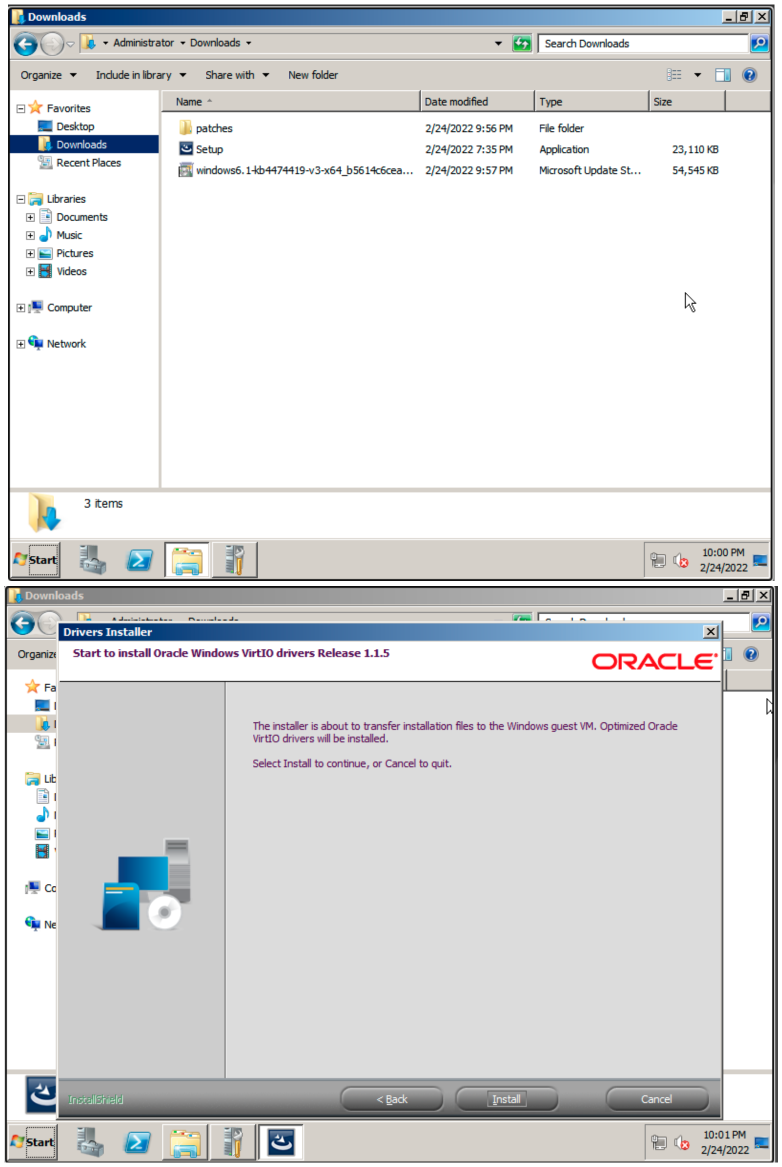
You can download from My Oracle Support (MOS) or from the Oracle Software Delivery Cloud (OSDN).
If downloading from OSDN:

1. Log into the Oracle Software Delivery Cloud
2. In the All Categories List, select **Release**.
3. Type "**Oracle Linux**" and click Search.
4. Select "**REL: Oracle Linux N.N.0.0.0 (Oracle Linux)**". The values of N will be different as new releases are made available.
5. Click on the "**Add to Cart**" button and then click on "**Checkout**" in the right upper corner. On the following window, select "**x86-64**" and click on the "**Continue**" button
6. Accept the "Oracle Standard Terms and Restrictions" to continue and, on the following window, click on "**VNNNNNNN.zip - Oracle VirtIO Drivers Version for Microsoft Windows N.N.N**" to download the drivers. The current version at this writing is V1009702-01.zip. The version number will change as new releases are delivered).
7. Download the zip file to your Windows VM,

If downloading from MOS:

1. Sign into https://support.oracle.com/portal/
2. Click on Patches & Updates tab.
3. In the Patch Search page's Patch Name or Number field, enter 27637937
4. Click the Patch Name to the left of "Oracle VirtIO driver version 1.1.7" for Release 7.9.0.0.0
5. Click the Download button and download p27637937_79000_MSWIN-x86-64.zi

Install the drivers by double clicking on the Setup file. When complete, restart and shutdown.



Name	Date modified	Type	Size
patches	2/24/2022 9:56 PM	File folder	
Setup	2/24/2022 7:35 PM	Application	23,110 KB
windows6.1-kb4474419-v3-x64_b5614c6cea...	2/24/2022 9:57 PM	Microsoft Update St...	54,545 KB

Start to install Oracle Windows VirtIO drivers Release 1.1.5

The installer is about to transfer installation files to the Windows guest VM. Optimized Oracle VirtIO drivers will be installed.

Select Install to continue, or Cancel to quit.

Figure 6 Install drivers on Microsoft Windows

Prepare cloud-init or cloudbase-init (optional)

Customers are encouraged to install the `cloud-init` (Linux) or `cloudbase-init` (Solaris, Windows) toolkits. They are the industry-standard, vendor agnostic methods for cross-platform instance initialization that identify the cloud the instance is running on during boot, read any provided metadata from the cloud and initialize the system accordingly. It has serveral capabilities, among which is converting from password-based `ssh` to using a key pair exchange as described in section 6.4 of the User Guide and at https://cloudinit.readthedocs.io/en/latest/

On Linux, install the `cloud-init` and `oci-utils` packages suitable for your Linux distribution. For Oracle Linux 8 this is:

```
# dnf install yum-utils
# yum-config-manager --enable ol8_addons
# dnf install cloud-init oci-utils
```

On Solaris, issue '`pkg install cloudbase-init`' as described in the Solaris 11.4 document https://docs.oracle.com/cd/E37838_01/html/E60974/gpxcg.html

In both Linux and Solaris you would then `ssh` into the launched instance with userid **`opc`** without a password, using the key pair exchange described in the references cited above.

The preceding steps disable password-based authentication. If you want to continue using password based ssh authentication on the target system to keep it more like how it operated on the source platform, it can be re-enabled before export by editing `/etc/cloud.cfg` and adding the line:

```
ssh_pwauth:   1
```

Alternatively, this can be done before export by commenting out the following line in `/etc/ssh/sshd_config`:

```
# PasswordAuthentication no
```

On Windows instances, use cloudbase-init, as described at https://cloudbase-init.readthedocs.io/en/latest/

Prepare startup and file system

The image only contains the boot disk, so disable any boot-time services and applications that require other disks. References to non-boot file systems must be removed until re-created later. If the boot image refers to other disks for swap, `/tmp`, or `/var/logs` they must be temporarily moved to the boot volume.

On Linux: edit `/etc/fstab` and remove entries to file system mounts that are not on the boot disk. Ensure that boot disk partitions are not directly referenced by device names such as `/dev/xvdb`. Instead use "`LABEL=`" or "`UUID=`". References to LVM devices don't need to be modified if referring to the boot disk. Run the 'pvs' command to ensure that the only LVM physical volume (PV) is the boot volume. If a volume group (VG) is on other physical volumes it should be commented out of `/etc/fstab`. If a logical volume consists of both the boot and other disks then the migration is more complicated and out of the scope of this document.

On Solaris: edit `/etc/vfstab` to remove any mounts not on the boot disk. Issue "`zpool export $POOLNAME`" for non-root ZFS pools. Typically, those are ZFS pools other than "`rpool`".

In all cases, disable any services or applications that launch at boot time and require full configuration, since the initial launch on PCA X9-2 will not have the complete operational environment.

EXPORT BOOT DISK

Copy the boot disk from the source environment to a utility server that will be used to prepare the image for upload to PCA. Use a method that depends on the source hypervisor and management platform.

For example, on Oracle VM or PCA 2.4 obtain the `.img` file for the boot disk by copying it from the storage repository shown in the VM's `vm.cfg` file. The command would be run on the compute node or Oracle VM Server (OVS) and look like

```
cd /OVS/Repositories/repository_id/VirtualDisks
scp virtual-disk_id.img utilityhost:/data/virtual-disk_id.raw
```

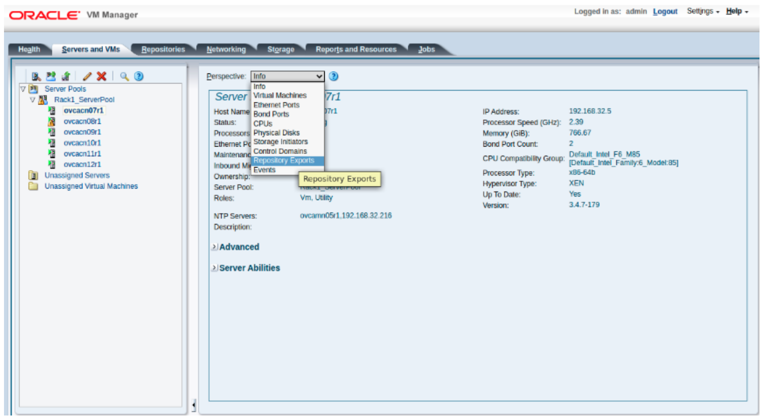
In the above example, the output file is renamed with the file extension `.raw`, an optional change that indicates that the Oracle VM `.img` file uses 'raw disk' image format.

PCA 2.x Details

On PCA 2.4, storage repositories are mounted to the compute nodes, which by default are on PCA's private networks and do not have a plumbed connection to the datacenter network. In that case, `scp` the `.img` file to either of the two management nodes, or create a read-only Oracle VM repository export to the management nodes, and then `scp` to an external host. Depending on your network and storage configuration, you may be able to NFS mount the Oracle VM or PCA storage repository or its clone to the utility server, and access the files directly. This is discussed in the appendix on bulk import.

That process, excerpted here, is described in the technical paper Oracle Private Cloud Appliance Backup Guide at https://www.oracle.com/technetwork/server-storage/private-cloud-appliance/documentation/pca-backup-x8-0-6-5676918.pdf This can be used to bulk export all the VMs in an Oracle VM storage repository.

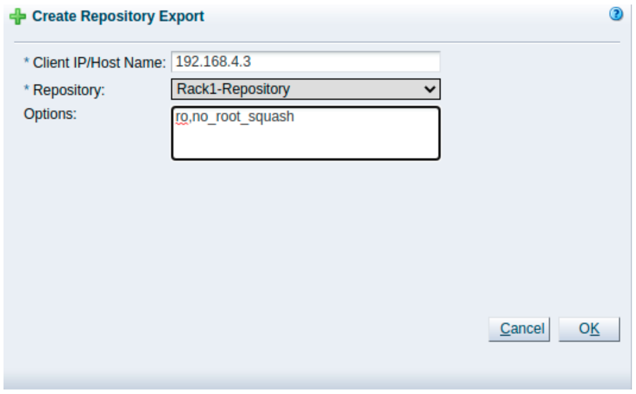
Navigate in the Oracle VM Manager User Interface to the Servers and VMs tab. Under Server Pools, expand the server pool name to show the server names, and highlight one of the servers to select it. Select Repository Exports from the Perspective view in the management panel.



Roles: Vm, Utility
NTP Servers: ovcamn05r1,192.168.32.216
Description:
IP Address: 192.168.32.5
Processor Speed (GHz): 2.39
Memory (GiB): 766.67
Bond Port Count: 2
CPU Compatibility Group: Default_Intel_F6_M85 [Default_Intel_Family:6_Model:85]
Processor Type: x86-64b
Hypervisor Type: XEN
Up To Date: Yes
Version: 3.4.7-179

Figure 7. Create NFS export on PCA X8-2 – part 1

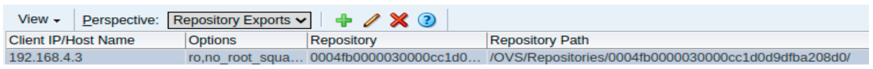
Click on the "+" icon to create a repository export, select the repository to be exported, the IP address or hostname of the host that will mount the NFS share, which would be the address of either of the management nodes, and specify the options "`ro,no_root_squash`".



* Client IP/Host Name: 192.168.4.3
* Repository: Rack1-Repository
Options: ro,no_root_squash

Figure 8. Create NFS export on PCA X8-2 – part 2

Click on the OK button and view the export.



Client IP/Host Name	Options	Repository	Repository Path
192.168.4.3	ro,no_root_squa…	0004fb0000030000cc1d0…	/OVS/Repositories/0004fb0000030000cc1d0d9dfba208d0/

Figure 9. Create NFS export on PCA X8-2 – part 3

Copy the Repository Path and issue a mount command using the selected compute node as the NFS server. The management node will have a read-only view of the storage repository and you can copy disk images directly from the repository's VirtualDisks directory without having to scp it from a compute node first.

```
[root@ovcamn05r1 ~]# mount 192.168.4.5:/OVS/Repositories/repository_id/ /mnt
[root@ovcamn05r1 ~]# ls -la /mnt
total 2077
drwxr-xr-x   8 root root  3896 Aug 10  2019 .
dr-xr-xr-x. 22 root root  4096 Jan  5 20:10 ..
drwx------   9 root root  3896 Jan 24 14:10 Assemblies
drwx------   2 root root  3896 Dec  9 14:23 ISOs
drwxr-xr-x   2 root root  3896 Aug  8  2019 lost+found
-rw-------   1 root root 15014 Mar  9 11:48 .ovsmeta
-rw-------   1 root root   151 Sep 10  2020 .ovsrepo
drwx------  11 root root  3896 Nov 25 16:12 Templates
drwx------   2 root root  8192 Mar  9 11:48 VirtualDisks
drwx------  53 root root  3896 Mar  9 11:48 VirtualMachines
```

If you wish, you can optimize the mount options, for example "`mount -o ro,bg,hard,rsize=524288,wsize=524288,nfsvers=3,tcp 192.168.4.5:/OVS/Repositories/` *`repository_id`* `/mnt`"

Oracle VM VirtualBox disk images

Oracle VM VirtualBox usually stores disk images in the directory "`VirtualBox VMs`" in a subdirectory with the same name as the VM. The VM information from the user interface provides the complete path.

```
$ vboxmanage showvminfo "Solaris 11.4"|more
Name:                        Solaris 11.4
Groups:                      /
Guest OS:                    Oracle Solaris 11 (64-bit)
UUID:                        3a4ce795-d6b1-44f8-a16b-8b7fb0fbb4a8
Config file:                 /Users/jeff/VirtualBox VMs/Solaris 11.4/Solaris 11.4.vbox
Snapshot folder:             /Users/jeff/VirtualBox VMs/Solaris 11.4/Snapshots
Log folder:                  /Users/jeff/VirtualBox VMs/Solaris 11.4/Logs

… snip …
```

Look in that folder for the `.vdi` or `.vmdk` disk image for the VM's boot disk, and `scp` that to the utility host.

KVM disk images

Virtual machine disk images can be located by issuing the command shown below. Locate the disk image for the VM you want to export to PCA, and `scp` that to the utility host.

```
$ virsh vol-list default
 Name                     Path
------------------------------------------------------------------------------
 guest-ol8-minimal-serial.qcow2 /home/admin/.local/share/libvirt/images/guest-ol8-
minimal-serial.qcow2
 guest-ol8-minimal.qcow2 /home/admin/.local/share/libvirt/images/guest-ol8-minimal.qcow2
 guest-ol8.qcow2 /home/admin/.local/share/libvirt/images/guest-ol8.qcow2
 guest-aspen-ol7.qcow2 /home/admin/.local/share/libvirt/images/guest-aspen-ol7.qcow2
 juniper-kvm-ol7.qcow2 /home/admin/.local/share/libvirt/images/juniper-kvm-ol7.qcow2
```

VMware disk images

VMware images can be exported to a "virtual appliance" which is a `tar` file containing the disk image. Use a similar procedure to obtain the virtual appliance, then use the `tar` command to extract the `.vmdk` file from the `.ova` file. Then copy it to the utility host. Alternatively, mount the VMware Datastore to the utility server and access the files directly. This is discussed in the appendix on bulk import.

Speed up export by compressing disk image files

Disk image files are large, so copying them to other hosts can take a long time. This may go much faster if you use compression, especially for sparse disk images. You can optionally compress the disk image file using a command like `gzip` or `lzop` to reduce the time to copy to the utility host. Here is an example that uses `lzop` and pipes output to `scp`:

```
lzop -1 < virtual-disk_id.img | ssh utilityhost 'lzop -d > /data/ virtual-
sdisk_id.raw'
```

We use `lzop` for compression in this case because a fast network is available, so compression speed was more important than compression ratio. If you have a slower network, you may opt for another approach such as using `gzip` or `bzip2`.

```
gzip virtual-disk_id.img
scp virtual-disk_id.img.gz utilityhost:/data/
```

All of these methods, or using no compression at all, are equally valid and depends on the administrator's preferences.

SET UP THE UTILITY HOST (ONE TIME OPERATION)

Create a utility host that runs Oracle Linux. This can be a virtual machine or a bare metal instance. Prepare the utility host by installing the OCI CLI, as described in section 1.2 of the User Guide.

Also, use the following commands to install `qemu-img` and `virt-sparsify`. This example assumes an Oracle Linux 7 host:

```
$ sudo yum upgrade
$ sudo yum-config-manager --enable ol7_kvm_utils
$ sudo yum install qemu-img libguestfs-tools
```

TRANSFORM BOOT DISK

The boot disk may need to be converted into the open standard `qcow2` format used by the PCA hypervisor. Perform this on the utility host that has access to OCI CLI, `qemu-img` command and other optional utilities.

> **Note**: It is recommended, but not strictly necessary, to convert disk images already in `vmdk` or `qcow2` format and package it in a `.oci` file. The `.oci` file includes the disk image in `qcow2` format and image metadata. That ensures the correct metadata for the image is provided with the disk contents.

Process the boot disk

Perform this operation on the boot disk to create a disk image file in `qcow2` format. This is necessary for disk images from Oracle VM or PCA 2.x. This is optional for disk images in `vmdk` format (VMware or Oracle VirtualBox) unless you plan to place them in a `.oci` file as described below.

On the utility node, issue commands like:

```
qemu-img -p convert -f raw -O qcow2 virtual-disk_id.raw output.QCOW2
qemu-img convert -f vdi -O qcow2 ol8.vdi output.QCOW2
qemu-img convert -f vmdk -O qcow2 myVMwareVM.vmdk output.QCOW2
```

Use "`-f raw`" for virtual disks from Oracle VM and PCA 2.x, use "`-f vdi`" for `.vdi` files from Oracle VM VirtualBox, and use "`-f vmdk`" for `vmdk` files from VMware. It is recommended but not necessary to convert a `vmdk` image because PCA can use that file type. See below in the Import section.

> **Note:** The output file <u>must</u> be named `output.QCOW2` to be used in a `.oci` file.

> **Note:** The `qemu-img` command can take a substantial time to run depending on the image file size and the system speed. Use the flag "`-p`" to show progress displayed as a percentage in the form `(nn.nn/100%).`

You may be able to reduce the size of the disk image by using the `virt-sparsify` tool. This tool works on many file systems: ext2/3/4, xfs, btrfs, NTFS, LVM. and reduces disk image size by eliminating disk blocks not owned by a file. The amount of reduction varies based on the contents of the disk image before exporting. Depending on your utility environment, you may have to also issue "`export LIBGUESTFS_BACKEND=direct`" before running the command:

```
$ virt-sparsify --in-place output.QCOW2
```

Preparing for import: the OCI file type

This step is optional but recommended.

PCA X9-2 image and Oracle Cloud Infrastructure (OCI) images use the file type `.oci`, which has two components: an `image_metadata.json` file that describes the image in JSON format, and the actual disk image named `output.QCOW2` prepared in the previous step. The `image_metadata.json` file describes the image's settings (BIOS vs. UEFI, how virtual devices are implemented), and what operating system it runs. The examples shown in Appendix 2 can be used with little change. Select the BIOS or UEFI version based on the source and (for neatness sake) change the operating system description. The file can be compressed, as shown below, which will help reduce network transfer time when it is imported.

```
$ tar zcf MyImage.oci image_metadata.json output.QCOW2
```

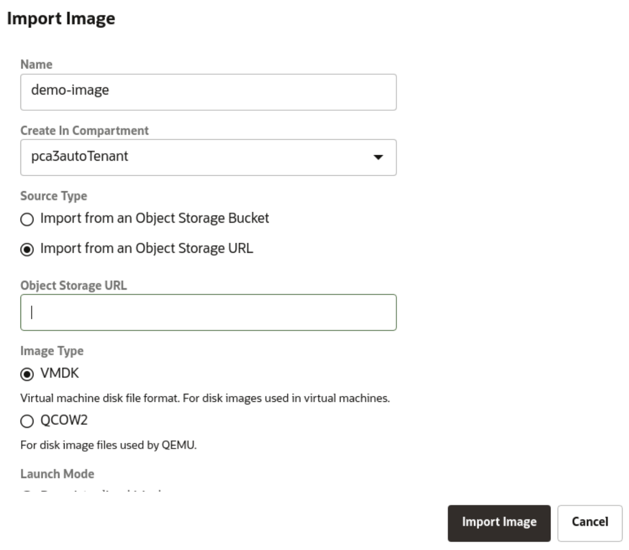
IMPORT TO PCA

Once the `vmdk`, `qcow2 or oci` file is available, it can be imported into PCA without further change. Upload it from any web server that has network connectivity to the PCA management nodes. The file can be copied to an existing web server, or you can create an impromptu web server using a Python command similar to the following (adjusting the port number as needed). If using Python 2.x use `python -m HTTPSimpleServer 8080` and If using Python 3.x use `python3 -m http.server 8080`

Images are imported into PCA using either the PCA Compute Enclave browser user interface (BUI) or the OCI command line interface (CLI). See the User Guide section 5.1 for a full description.

Import using browser interface

If using the browser interface, log into the Compute Enclave, go to the Compute Images page, click on 'Import Image', and then fill in details on the pop-up:


Import Image

Name: demo-image

Create In Compartment: pca3autoTenant

Source Type
- ○ Import from an Object Storage Bucket
- ● Import from an Object Storage URL

Object Storage URL: |

Image Type
- ● VMDK
  Virtual machine disk file format. For disk images used in virtual machines.
- ○ QCOW2
  For disk image files used by QEMU.

Launch Mode

Figure 10. Import custom image using browser user interface

Select 'Object Storage URL' and enter the URL to the file containing the boot image. Specify VMDK if it is a `.vmdk` file. Use QCOW2 if this is either a `.qcow2` file or a `.oci` file containing both metadata and the `output.QCOW2` disk image file. Click on the 'Import image' button to start the import. This will take you to a page showing the status of the image, initially 'Importing' and then 'Available' when the import completes.

NOTE: The user interface only has one Launch Mode – "Paravirtualized". If you import a UEFI image via the browser interface it might be imported with BIOS instead of UEFI. The workaround in the PCA Release Notes 3.5.10 is to use the CLI to launch the instance and specify the launch options there. See the details below on the section describing instance launch with CLI.

The time it takes to perform the import depends on the size of the image and the speed of the network between the PCA system and the web server. The user interface is not locked during the import so you can proceed to other tasks or log off. Check on the status of the image by viewing the image's display.

Import using OCI CLI

If using the OCI CLI, use a line like the example below, substituting in the image URL and the OCID for your tenancy. The import command displays the import work unit, and you can issue a 'get' command to display its status.

```
$ oci compute image import from-object-uri \
   --uri http://mywebserver/shares/export/images/mydemo.oci \
   --display-name "demo-image" --compartment-id $OCI_CLI_TENANCY
{
  "data": {
    "agent-features": null,
    "base-image-id": null,
    "compartment-id": "ocid1.tenancy.unique_id",
    "create-image-allowed": true,
    "defined-tags": {},
    "display-name": "demo-image",
    "freeform-tags": {},
    "id": "ocid1.image.unique_id ",
    "launch-mode": "PARAVIRTUALIZED",
    "launch-options": null,
    "lifecycle-state": "IMPORTING",
    "operating-system": "UNAVAILABLE",
    "operating-system-version": "UNAVAILABLE",
    "size-in-mbs": 0,
    "time-created": "2022-02-03T00:41:24.982617+00:00"
  },
  "etag": "ae01e390-eab5-449c-b78d-c655235af23f",
  "opc-work-request-id": "ocid1.workrequest.unique_id"
}
```

View the import via the browser interface, or check via the CLI, using the OCID displayed in the import-image command.

```
$ oci compute image get --image-id ocid1.image.unique_id
{
  "data": {
    "agent-features": null,
    "base-image-id": null,
    "compartment-id": "unique_id",
    "create-image-allowed": true,
    "defined-tags": {},
    "display-name": "demo-image",
    "freeform-tags": {},
    "id": "ocid1.image.unique_id",
    "launch-mode": "CUSTOM",
    "launch-options": {
      "boot-volume-type": "PARAVIRTUALIZED",
      "firmware": "BIOS",
      "is-consistent-volume-naming-enabled": false,
      "is-pv-encryption-in-transit-enabled": false,
      "network-type": "PARAVIRTUALIZED",
      "remote-data-volume-type": "PARAVIRTUALIZED"
    },
    "lifecycle-state": "AVAILABLE"
    "operating-system": "Fedora Linux"
    "operating-system-version": "35"
    "size-in-mbs": 512000
    "time-created": "2022-02-03T00:41:24.982617+00:00
  }
  "etag": "d96dda18-e10b-4d87-83ab-425bb5cf582e
}
```

Import vmdk or qcow2 file via CLI without .oci file

PCA supports `vmdk` and `qcow2` file types so you can import them directly without creating a `.oci` file by adding **`--source-image-type VMDK`** or **`--source-image-type QCOW2`**. We recommend creating the `.oci` file as above, but this is supported:

For example:

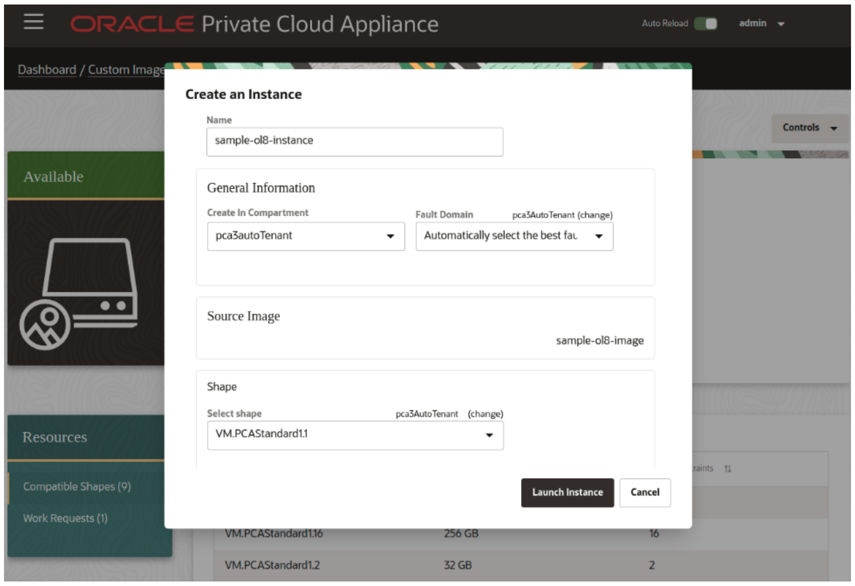
```
$ oci compute image import from-object-uri --uri http://mywebserver/images/myimage.vmdk
--display-name "myimage" --compartment-id $OCI_CLI_TENANCY \
    --source-image-type VMDK
{
  "data": {
    "agent-features": null,
    "base-image-id": null,
    "compartment-id": "ocid1.tenancy.unique_id",
    "create-image-allowed": true,
    "defined-tags": {},
    "display-name": "myimage",
    "freeform-tags": {},
    "id": "ocid1.image.unique_id",
    "launch-mode": "PARAVIRTUALIZED",
    "launch-options": null,
    "lifecycle-state": "IMPORTING",
    "operating-system": "UNAVAILABLE",
    "operating-system-version": "UNAVAILABLE",
    "size-in-mbs": 0,
    "time-created": "2022-02-03T01:31:12.529239+00:00"
  },
  "etag": "fa3c71ec-004b-4907-8952-828904b69aac",
  "opc-work-request-id": "ocid1.workrequest.unique_id"
}
$ oci compute image get --image-id ocid1.image.unique_id
{
  "data": {
    "agent-features": null,
    "base-image-id": null,
    "compartment-id": "ocid1.tenancy.unique_id",
    "create-image-allowed": true,
    "defined-tags": {},
    "display-name": "myimage",
    "freeform-tags": {},
    "id": "ocid1.image.unique_id",
    "launch-mode": "PARAVIRTUALIZED",
    "launch-options": {
      "boot-volume-type": "PARAVIRTUALIZED",
      "firmware": "BIOS",
      "is-consistent-volume-naming-enabled": false,
      "is-pv-encryption-in-transit-enabled": false,
      "network-type": "PARAVIRTUALIZED",
      "remote-data-volume-type": "PARAVIRTUALIZED"
    },
    "lifecycle-state": "AVAILABLE",
    "operating-system": "CUSTOM",
    "operating-system-version": "CUSTOM",
    "size-in-mbs": 512000,
    "time-created": "2022-02-03T01:31:12.529239+00:00"
  },
  "etag": "6564352c-4fea-4335-bad4-b6c9efeb05cd"
}
```

INITIAL LAUNCH ON PCA

You can launch the image without any special additional steps, as illustrated below. See section 6 of the User Guide for further details.

Instance launch using browser interface

From the page describing the image, click on the Controls drop-down menu, then click on 'Create Instance'. Fill in the pop-up window with details on the instance's name, fault domain, shape and virtual network and subnet. If using `cloud-init`, fill in the public key to permit password-free `ssh`.



Create an Instance

Name: sample-ol8-instance

General Information

Create In Compartment: pca3autoTenant

Fault Domain pca3AutoTenant (change): Automatically select the best fau

Source Image: sample-ol8-image

Shape

Select shape pca3AutoTenant (change): VM.PCAStandard1.1

VM.PCAStandard1.16	256 GB	16
VM.PCAStandard1.2	32 GB	2

Figure 11. Instance launch using browser user interface

The instance will be in Provisioning state while it is being created, and then is automatically launched. After the instance is provisioned, you can create a console connection to view the instance console, as described in the User Guide.

Instance launch using OCI CLI

You can use OCI CLI with a pattern like this, substituting in values for the tenancy, shape, source details, subnet, keys and names:

```
$ oci compute instance launch -c $OCI_CLI_TENANCY \
   --availability-domain ad1 --shape VM.PCAStandard1.2 \
   --source-details '{"boot-volume-size-ingbs":100, \
                      "imageId":"$MYIMAGEID"},"sourceType":"image"}' \
   --metadata '{"ssh_authorized_keys":"ssh-rsa ...$RESTOFMYKEY myuserid@mydesktop"}' \
   --display-name "myinstance" --assign-public-ip true --subnet-id $MYSUBNET_OCID
```

The CLI is scriptable and can be repeated to reduce manual actions. If you need to override the launch mode or parameters, specify them on the command line:

```
$ oci compute instance launch [...] \

  --launch-options '{"boot-volume-type":"PARAVIRTUALIZED","firmware":"UEFI_64", "is-
consistent-volume-naming-enabled":false,"is-pv-encryption-in-transit-enabled": false,
"network-type":"PARAVIRTUALIZED","remote-data-volume-type": "PARAVIRTUALIZED"}'
```

POST-LAUNCH: COMPLETE THE INSTANCE

Add block volumes (non-boot disks)

Previous steps created the boot environment on the PCA X9-2. Review the data collected in the preparation steps to size disks that needed to complete the image. Use the PCA Compute Enclave navigation to create block volumes and attach them to the instance, or use OCI CLI commands like:

```
oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --size-in-gbs 50
oci compute volume-attachment attach --instance-id $INSTANCE_OCID --volume-id
$VOLUME_OCID --type paravirtualized
```

Then, partition and create file systems on the block volumes using tools appropriate to the guest operating system, and copy contents from the source system. The approach will depend on the applications and nature of the data, so details are out of scope for this document.

If the VM on the source system uses NFS, SMB/CIFS or iSCSI targets for its data, it may be possible to expose them to the new instance without having to create new storage resources. Permissions and iSCSI parameters may need to be changed on the storage device.

Note: PCA X9-2 does not support Fibre Channel devices.

IMPORTING FROM ORACLE CLOUD INFRASTRUCTURE (OCI) TO PCA AND VICE-VERSA

PCA X9-2 is highly compatible with OCI, which simplifies the migration process, so is separately described here.

OCI to PCA

Before exporting from OCI, run `'cloud-init clean'` on a clone of the instance, so the next boot runs the services as a fresh instance. In the OCI console, select the instance, click on More Actions, and then Create Custom Image. Once the image has been created, go to your list of images, select that image and click Export and the name of the object bucket you want to use. Then, go to the Object Store, select the exported image file, and click Download. Rename it to have a `.oci` extension if you didn't specify that in the download step. If you use the OCI CLI the command would look like

```
oci os object get \
  --namespace-name <object_storage_namespace> \
  --bucket-name <bucket_name> \
  --name <object_name> \
  --file <file_location
```

Once you have downloaded the file you need to make two changes to the `image_metadata.json` file. Untar the file, edit `image_metadata.json` using an ASCII text editor like vi, pluma, gedit or Notepad, and set the value *false* for the variables *`pvEncryptionInTransitEnabled`* and *`consistentVolumeNamingEnabled`*.

Then re-create the `.oci` file using the `tar` command (`tar zcvf myimage.oci image_metadata.json output.QCOW2`) and directly import it into PCA using the browser interface or command line. No further transformation is required.

PCA to OCI

Moving the image in the other direction is very similar: run `'cloud-init clean'`, create a custom image from the image, export the image via an object bucket, download it, and then import into OCI. There is no need to change the JSON file. The command to download from PCA, if using the CLI, is the same as the example above.

Then log into OCI, go to Object Storage → Buckets and to the bucket you use for this procedure. Click Objects then Upload, and then specify the path to the `.oci` file. Once the file has been uploaded you can import it as a custom image into OCI: go to Compute → Custom Images and select Import, and then point to the bucket you used, selecting OCI as the file type.

TAKE A BACKUP

Create a backup after first successful boot for fallback after any further changes. See the Users Guide for details on creating a new custom image and taking boot and block volume backups

DATA MIGRATION, ACCEPTANCE TESTING AND CUTOVER

At this point you have a running instance on PCA. Copy over the application data using the tools appropriate to the operating system and application (e.g.: `rsync` or `scp` for file data in Linux or Solaris) and conduct user acceptance testing in parallel with undisturbed operation in the source system.

Once acceptance testing is successful, perform a final data synchronization from the source stem to the new environment. The procedures for this last, essential, step will be specific to the operating system and applications.

SUMMARY

This document provides a guide with instructions and process for moving workloads from source system to PCA X9-2. Architects and administrators transitioning virtual machines to PCA can use this as a basis for their migrations.

APPENDIX 1: SHAPES

PCA X9-2 supports the following standard shapes and flexible shapes. When sizing an instance, select the shape that provides sufficient CPU and memory capacity and accommodates expected growth. If the source platform server speed is slower than the X9-2 server, or the VM is overprovisioned with more capacity than it needs, you can select a smaller shape.

Note that the shape also includes a maximum number of virtual network devices. If an instance needs more than provided by the smallest shape meeting its CPU cores and memory requirements, you'll have to promote to a larger shape.

List of Standard Shapes

Shape Name	Cores (OCPUs)	Memory (GB)	Virtual NICS
VM.PCAStandard1.1	1	16	2
VM.PCAStandard1.2	2	32	2
VM.PCAStandard1.4	4	64	4
VM.PCAStandard1.8	8	128	8
VM.PCAStandard1.16	16	256	16
VM.PCAStandard1.24	24	384	24
VM.PCAStandard1.32	32	512	24
VM.PCAStandard1.48	48	768	24
VM.PCAStandard1.Max	60	960	24

Flexible Shapes

Alternatively, you can use flexible shapes by selecting the shape name `VM.PCAStandard1.Flex`. This lets you customize the instance's number of cores and amount of memory and create instances that precisely meet your workload requirements, while optimizing performance and using resources efficiently. For example, you could have an instance with 3 cores and 17 GB of RAM if that is the size that meets the instance's needs.

Instances with flexible shapes can have between 1 and 32 cores (OCPUs), and between 1 GB and 64GB RAM per core, up to a maximum of 512GB. If more cores or memory are needed, use one of the larger standard shapes. An instance with one core can have two virtual NICs, instances with two to 24 cores can have 1 VNIC per OCPU, and instances with 25 to 32 cores have a maximum of 24 VNICs.

Since cores and memory aren't implied by the shape name, the instance must be launched with size information provided in the browser interface, or by specifying on the `oci compute instance launch` command, e.g.: "`--shape-config '{"ocpus":6, "memoryInGBs":94}'`

Sample shapetools.py

The following sample program can be used to select the standard shape that matches a virtual machine's size, or conversely, to display the VM cores and memory for a given shape name:

```
#!/usr/bin/env python3
from sys import argv
import argparse
shapelist = [("VM.PCAStandard1.1",  1,  16),
            ("VM.PCAStandard1.2",   2,  32),
            ("VM.PCAStandard1.4",   4,  64),
            ("VM.PCAStandard1.8",   8, 128),
            ("VM.PCAStandard1.16", 16, 256),
            ("VM.PCAStandard1.24", 24, 384),
            ("VM.PCAStandard1.32", 32, 512),
            ("VM.PCAStandard1.48", 48, 768),
            ("VM.PCAStandard1.Max",60, 960 ) ]
def size2shape(vcpus,mb) :
   ''' size2shape returns the smallest standard shape matching an instance vcpus and memory in MB '''
   instanceshape="No matching shape"
   VMcores = (int(vcpus)+1)//2              # round up CPU threads to cores, assumes hyperthreading
   VMmem   = (int(mb)+1023)//1024           # round up memory from MB to next GB
   for s in range(len(shapelist)) :
       shape = shapelist[s]
       shapename, shapecores, shapemem = shape
       if VMcores <= shapecores and VMmem <= shapemem :
          instanceshape = shapename
          break
   return (instanceshape)

def shape2cores(instanceshape)  # if this wasn't an afterthought it would have used a dictionary.
   ''' provide number of cores for a given fixed shape '''
   instancecores = 0
   for s in range(len(shapelist)) :
       shape = shapelist[s]
       shapename, shapecores, shapemem = shape
       if shapename==instanceshape :
          instancecores = shapecores
          break
   return (instancecores)

if __name__ == "__main__" :
    args = len(argv) - 1
    if args < 2 :
       print ("Specify number of vCPU threads and RAM size in MB.")
       exit()
    scriptname, vcpus, mb = argv
    print (size2shape(vcpus,mb))
```

APPENDIX 2: IMAGE_METADATA.JSON EXAMPLES

The contents below can be used as the basis of the `image_metadata.json` file included in a .oci bundle for upload. Select the BIOS or UEFI version as determined by the source environment Values for "`operatingSystem`" and "`operatingSystemVersion`" should be set to the values for the imported image, and other fields should be left as shown.

BIOS images

```
{
  "version": 2,
  "externalLaunchOptions": {
    "firmware": "BIOS",
    "networkType": "PARAVIRTUALIZED",
    "bootVolumeType": "PARAVIRTUALIZED",
    "remoteDataVolumeType": "PARAVIRTUALIZED",
    "localDataVolumeType": "PARAVIRTUALIZED",
    "launchOptionsSource": "CUSTOM",
    "pvAttachmentVersion": 1,
    "pvEncryptionInTransitEnabled": false,
    "consistentVolumeNamingEnabled": false
  },
  "imageCapabilityData": null,
  "imageCapsFormatVersion": null,
  "operatingSystem": "Oracle Linux",
  "operatingSystemVersion": "8.5"
}
```

UEFI images

```
{
  "version": 2,
  "externalLaunchOptions": {
    "firmware": "UEFI_64",
    "networkType": "PARAVIRTUALIZED",
    "bootVolumeType": "PARAVIRTUALIZED",
    "remoteDataVolumeType": "PARAVIRTUALIZED",
    "localDataVolumeType": "PARAVIRTUALIZED",
    "launchOptionsSource": "CUSTOM",
    "pvAttachmentVersion": 1,
    "pvEncryptionInTransitEnabled": false,
    "consistentVolumeNamingEnabled": false
  },
  "imageCapabilityData": null,
  "imageCapsFormatVersion": null,
  "operatingSystem": "Oracle Linux",
  "operatingSystemVersion": "7.9"
}
```

APPENDIX 3: BULK IMPORT SAMPLE PROGRAMS

This document focuses on importing individual VMs. This appendix shows sample, demonstration scripts in Python, offered without support or warranty, that show how to partially automate migration from a VMware or Oracle VM / PCA 2.x source platform.

Note: for production migration we recommend working with Oracle ACS or Oracle partners like Rackware, Coriolis or Commvault that have enterprise-grade solutions.

Purpose and methodology

The sample programs are passed the file locations of the source platform's virtual machine's metadata and disk files, and the location of a directory to place transformed disk images, commands and descriptive metadata.

For each VM, the scripts generate Linux commands

1. transform the virtual boot disk into the format for PCA using `qemu-img`,
2. create a `.oci` file
3. Import the image file into PCA using OCI CLI commands
4. Launch the instance on PCA using OCI CLI commands

5. Define block volumes using OCI CLI commands

The commands are incomplete and not runnable without editing since the OCIDs for the target compartment or subnets are not available. They can be post-processed to add that information, or simply used as a model, or the scripts could be modified to include that information.

The scripts also generate metadata files in JSON format for each VM, and check for error conditions, such as VM sizes that have no matching shape.

VM content locations on VMware and Oracle VM source platforms

On VMware, virtual machine information are contained in a Datastore, which may be a LUN or an NFS share. Each directory contains VM description files (a `.vmx` file) and disk images. Disk images are in `.vmdk` files, which may be ASCII descriptor files or files containing the actual disk image contents.

Oracle VM and PCA 2.x place virtual information in storage repositories, mounted to Oracle VM Server compute nodes under a path /OVS/Repositories/$UUID, where $UUID is a long unique identifier string, Virtual machine descriptions are stored in an ASCII text file named `vm.cfg` in the `VirtualMachines` directory of the repository, with a unique directory for each VM. Virtual disk image files are stored in the `VirtualDisks` directory.

VMware sample – vmwparse.py

The script is passed the directory of a VMware Datastore and the directory of a location to store the script's outputs. The Datastore can be a LUN or an NFS mount. Ensure it is mounted read-only and that permissions are set so the script can read the Datastore contents.

The generated command files are stored in a subdirectory 'commands', with one file per VM, containing standard Linux commands and OCI CLI commands.

Two options are generated for processing the boot disk image. One option converts the disk image into QCOW2 format and creates a `.oci` file containing image metadata. The other option uses the VMware disk images in `vmdk` format, since that can be directly used by PCA. If the `vmdk` option is used, the image import and instance launch commands use additional operands. The script also specifies UEFI_64 is the source VM firmware is UEFI.

Sample output for a VMware VM with multiple disks, edited for appearance for line wraps. Note the different options for importing the boot volume and launching the instance.

```
BOOTVOL IMPORT OPTION 1 (.oci): qemu-img convert -p -f vmdk \
   -O qcow2 /mnt/vmware/TestLotsOfDiskTypes/TestLotsOfDiskTypes-flat.vmdk
output/bootdisks/output.QCOW2
BOOTVOL IMPORT OPTION 1 (.oci): cp -f image_metadata.json-BIOS image_metadata.json
BOOTVOL IMPORT OPTION 1 (.oci): tar zcvf output/bootdisks/TestLotsOfDiskTypes.oci
image_metadata.json \                    output/bootdisks/output.QCOW2
BOOTVOL IMPORT OPTION 1 (.oci): oci compute image import from-object-uri \
  --uri $IMAGEURL/TestLotsOfDiskTypes.oci -c $OCI_CLI_TENANCY --display-name
"TestLotsOfDiskTypes_image"

BOOTVOL IMPORT OPTION 2 (.vmdk): oci compute image import from-object-uri \
  --uri $IMAGEURL/TestLotsOfDiskTypes/TestLotsOfDiskTypes-flat.vmdk -c $OCI_CLI_TENANCY
  --launch-mode PARAVIRTUALIZED --source-image-type VMDK --display-name
"TestLotsOfDiskTypes_image" \

LAUNCH OPTION 1 (.oci):oci compute instance launch --display-name TestLotsOfDiskTypes \
   --availability-domain ad1 -c $OCI_COMPARTMENT --subnet-id $SUBNET --assign-public-ip true \
   --shape VM.PCAStandard1.Flex --shape-config '{"ocpus":1, "memoryInGBs":2}' \
   --source-details '{"bootVolumeSizeInGBs":'16',"imageId":"$IMAGEOCID","sourceType":"image"}'

LAUNCH OPTION 2 (.vmdk):oci compute instance launch --display-name TestLotsOfDiskTypes \
   --availability-domain ad1 -c $OCI_COMPARTMENT --subnet-id $SUBNET --assign-public-ip true \
   --shape VM.PCAStandard1.Flex --shape-config '{"ocpus":1, "memoryInGBs":2}' \
   --source-details  '{"bootVolumeSizeInGBs":'16',"imageId":"$IMAGEOCID","sourceType":"image"}'
\
   --launch-options '{"boot-volume-type":"PARAVIRTUALIZED","firmware":"BIOS", \
        "is-consistent-volume-naming-enabled":false,"is-pv-encryption-in-transit-enabled":false,
\
        "network-type":"PARAVIRTUALIZED","remote-data-volume-type": "PARAVIRTUALIZED"}'

oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --size-in-gbs 24
   ### for TestLotsOfDiskTypes_1-flat.vmdk
oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --size-in-gbs 32
   ### for TestLotsOfDiskTypes_2-flat.vmdk
oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --size-in-gbs 40
   ### for TestLotsOfDiskTypes_3-flat.vmdk
oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --size-in-gbs 48
   ### for TestLotsOfDiskTypes_4-flat.vmdk
```

The metadata output can be used for other post processing:

```
$ cat TestLotsOfDiskTypes.json |jq
{
  "SOURCEPLATFORM": "VMWARE",
  "VMNAME": "TestLotsOfDiskTypes",
  "UUID": "56 4d 1d f7 97 3c 94 51-18 2c 7d 43 04 84 bc 40",
  "INSTANCENAME": "TestLotsOfDiskTypes",
  "SHAPE": "VM.PCAStandard1.Flex --shape-config '{\"ocpus\":1, \"memoryInGBs\":2}'",
  "CORES": 1,
  "MEMORY": "2048",
  "LAUNCHMODE": "BIOS",
  "OS": "oraclelinux6-64",
  "BOOTDISK": "/mnt/vmware/TestLotsOfDiskTypes/TestLotsOfDiskTypes-flat.vmdk",
  "BLOCKVOLS": [
    [
      "TestLotsOfDiskTypes_1-flat.vmdk",
      24
    ],
    [
      "TestLotsOfDiskTypes_2-flat.vmdk",
      32
    ],
    [
      "TestLotsOfDiskTypes_3-flat.vmdk",
      40
    ],
    [
      "TestLotsOfDiskTypes_4-flat.vmdk",
      48
    ]
  ]
}
```

```
#!/usr/bin/env python3
# vmwparse: parse .vmx files from a VMware Datastore and create CLI and hypervisor-neutral
metadata for export and import to PCA X9-2.
#
# Arguments:
#      1. Path to VMware Datastore
#      2. Path to store output, which will have subdirectories metadata, bootdisks, commands
#      example: $ ./vmwparse.py /home/me/projects/PCAbulk/VMWar/vmfs
/home/me/projects/PCAbulk/VMWare/output
# Note: this script assumes that hyperthreading is enabled on source platform. If not, just set
cores = threads.

import os
import sys
import stat
import json
from shapetools import *

if (len(sys.argv)>=2 and sys.argv[1]=="?") or len(sys.argv)<3 :
    print ("Usage is:", sys.argv[0], " <Datastore path> <outputpath>  Trailing '/' optionsl")
    quit()

sourceFS   = sys.argv[1].rstrip('/')
outputFS   = sys.argv[2].rstrip('/')
print(f'VMware Datastore is at {sourceFS} and output is stored under {outputFS}')
metadataFS   = outputFS+"/metadata/";  os.system("mkdir "+metadataFS)
commandsFS   = outputFS+"/commands/";  os.system("mkdir "+commandsFS)
bootdisksFS  = outputFS+"/bootdisks/"; os.system("mkdir "+bootdisksFS)

totalcores = totalgb = vmcount = toobigcount = 0; toobigVMs = ""
GB=1024*1024*1024

# constants for PCA 3.0.x: maximums for cores and memory (in GB), and maximums for flex shapes
MAXCORES        = 60
MAXGB          = 960
MAXFLEXCORES  = 32
MAXFLEXGB     = 512
MINBLOCKVOLGB = 50
launch_options_BIOS = '--launch-options \'{"boot-volume-
type":"PARAVIRTUALIZED","firmware":"BIOS","is-consistent-volume-naming-enabled":false,"is-pv-
encryption-in-transit-enabled": false,"network-type":"PARAVIRTUALIZED","remote-data-volume-
type": "PARAVIRTUALIZED"}\''
launch_options_UEFI = '--launch-options \'{"boot-volume-
type":"PARAVIRTUALIZED","firmware":"UEFI_64","is-consistent-volume-naming-enabled":false,"is-pv-
encryption-in-transit-enabled": false,"network-type":"PARAVIRTUALIZED","remote-data-volume-
type": "PARAVIRTUALIZED"}\''

SOURCEPLATFORM = 'VMWARE'   # constant for all VMware, inserted in metadata to denote source
platform

nameMapFile    = metadataFS+"namemap.csv"
nameFD         = open(nameMapFile,'w')

VMs = os.listdir(sourceFS)
for vmid in VMs:
    if vmid.startswith('.') : continue
    vmcount = vmcount + 1
    vmx = open(sourceFS+"/"+vmid+"/"+vmid+".vmx")
    print(f'\n--------{vmid}--------------')
    vmname = vmid; vcpus = '2'; memory = "2048"; guestOS = 'none'; bootdisk = bootfile = ''; bv
= []
```

```
    uuid = vmid
    launchmode    = 'BIOS'; launch_options = launch_options_BIOS # unless overridden
    for v in vmx.readlines() :
        v = v.strip("\n")
        if len(v)==0 : continue
        keyword= v.split('=')[0].strip()
        value  = v.split('=')[1].strip().strip("'").strip('"')
        if   keyword=="displayName" : vmname = value
        elif keyword=="uuid.bios"   : uuid   = value
        elif keyword.endswith('fileName') and value.endswith('vmdk') :  # is a disk or disk
descriptor file
          filename = value[:-5]            # strip off the .vmdk'
          diskfile = filename+"-flat.vmdk" # the actual disk file
          fullfilename = sourceFS.rstrip('/')+"/"+vmid+"/"+diskfile
          try :
              filestat=os.stat(fullfilename)
              diskbytes = getattr(filestat,"st_size")
              diskGB    = (diskbytes+GB-1)//GB
              print(f'disk size for {fullfilename} is {diskGB}GB {diskbytes} bytes')
          except :
              print(f'Unable to os.stat({fullfilename})')
              diskGB = MINBLOCKVOLGB
          s = filename.rfind('_')          # see if filename is of form diskfn_NN.vmdk (block)
or diskfn.vmdk (boot)
          seqno = filename[s+1:]
          if s  == -1 or not seqno.isnumeric() :  # found the boot disk
             bootdisk = sourceFS+"/"+vmid+"/"+diskfile
             bootfile = diskfile; bootdiskGB = diskGB
             print (f'   Bootdisk {bootdisk} {bootdiskGB}GB')
          else :   # have block volume
             bv.append([diskfile,diskGB])
             print (f'   Blockdisk {diskfile}  {fullfilename} {diskGB}GB blockdisk ')
        elif keyword == "numvcpus":     maxvcpus = value
        elif keyword == "memSize" :     memory   = value; mem = int(memory)  # memory in MB
        elif keyword == "guestOS" :     guestOS  = value
        elif keyword == "firmware" and value == "efi" :
             launchmode = "UEFI_64"
             launch_options = launch_options_UEFI

    threads = int(maxvcpus)
    cores   = (threads+1)//2      # Assumes hyperthreading is enabled. If not, just set cores =
threads
    totalcores = totalcores + cores
    gb=(int(mem)+1023)//1024
    totalgb = totalgb+gb

    if cores > MAXCORES or gb > MAXGB :
       print(f'    #### VM {vmname} exceeds max shape: core {cores} memory {gb}GB ####')
       shape = "NO MATCHING SHAPE"
       toobigcount = toobigcount + 1;
       toobigVMs   = toobigVMs + " " + vmid
    elif cores <= MAXFLEXCORES and gb <= MAXFLEXGB and gb//cores <= 64 :
       shapeconfig = "--shape-config '{\"ocpus\":" +str(cores)+ ", \"memoryInGBs\":" + str(gb)+
"}'"
       shape =  "VM.PCAStandard1.Flex " + shapeconfig
    else :
       shape   = size2shape(threads,mem)

    print (f'   VM {vmname}, core/OCPUs {cores}, memory {memory}')
    vmdict = {'SOURCEPLATFORM': SOURCEPLATFORM, 'VMNAME': vmname, 'UUID': uuid, 'INSTANCENAME': 
vmid, 'SHAPE': shape, 'CORES': cores, 'MEMORY': memory, 'LAUNCHMODE': launchmode, 'OS': guestOS, 
'BOOTDISK': bootdisk, 'BLOCKVOLS': bv}

    convertcmd      = 'BOOTVOL IMPORT OPTION 1 (.oci): qemu-img convert -p -f vmdk -O qcow2 
'+bootdisk+' '+bootdisksFS+'output.QCOW2'
    print(convertcmd)
    launchmodecmd   = 'BOOTVOL IMPORT OPTION 1 (.oci): cp -f image_metadata.json-'+launchmode +' 
image_metadata.json'
    print(launchmodecmd)
    tardotocicmd    = 'BOOTVOL IMPORT OPTION 1 (.oci): tar zcvf '+ bootdisksFS+vmid+'.oci 
image_metadata.json '+bootdisksFS+'output.QCOW2'
    print(tardotocicmd)
    displaynameoperand = ' --display-name "'+vmid+'_image"'
    imageimportcmd1 = 'BOOTVOL IMPORT OPTION 1 (.oci): oci compute image import from-object-uri 
--uri $IMAGEURL/'+ vmid+'.oci  -c $OCI_CLI_TENANCY'+displaynameoperand
    print(imageimportcmd1)
    imageimportcmd2 = 'BOOTVOL IMPORT OPTION 2 (.vmdk): oci compute image import from-object-uri 
--uri $IMAGEURL/'+ vmid + "/" + bootfile +' -c $OCI_CLI_TENANCY --launch-mode PARAVIRTUALIZED --
source-image-type VMDK'+displaynameoperand
    print(imageimportcmd2)
    launchcmd = 'oci compute instance launch --display-name ' + vmname + ' --availability-domain 
ad1 -c $OCI_COMPARTMENT --subnet-id $SUBNET --assign-public-ip true --shape ' + shape
    sourcedetails = ' --source-details 
\'{"bootVolumeSizeInGBs":\''+str(bootdiskGB)+'\',"imageId":"$IMAGEOCID","sourceType":"image"}\''
    launchcmd =  launchcmd + sourcedetails
    launchcmd1 = launchcmd # for .oci files
    launchcmd1 = "LAUNCH OPTION 1 (.oci):" + launchcmd
    print(launchcmd1)
    launchcmd2 = "LAUNCH OPTION 2 (.vmdk):" + launchcmd + launch_options  # for .vmdk alone
    print(launchcmd2)
    with open(commandsFS+vmid+'.sh','w') as cmdFD :
         cmdFD.write(convertcmd+"\n")
         cmdFD.write(launchmodecmd+"\n")
         cmdFD.write(tardotocicmd+"\n")
         cmdFD.write(imageimportcmd1+"\n")
         cmdFD.write(imageimportcmd2+"\n")
         cmdFD.write(launchcmd1+"\n")
         cmdFD.write(launchcmd2+"\n")
         for b in range(len(bv)) :
            bvcreate = 'oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --
size-in-gbs ' + str(bv[b][1]) + '  ### for ' + bv[b][0]
            print(bvcreate)
            cmdFD.write(bvcreate+"\n")

    nameMap = '"'+vmid+'","'+vmname+'"\n'
    nameFD.write(nameMap)
    with open(metadataFS+vmid+'.json','w') as j :
         json.dump(vmdict,j)
print(f'Parse done: {vmcount} VMs, {totalcores} cores, {totalgb} GB')
if toobigcount != 0 : print(f'{toobigcount} VMs were too large: {toobigVMs}')
nameFD.close()
```

Oracle VM sample – ovmrepoparse.py

The script is passed the directory of an Oracle VM storage repository and the directory to store output. The repository can be a LUN or an NFS share. Ensure it is mounted at the mountpoint `/OVS/Repositories`, is read-only and that permissions are set so the script can read the repository's `VirtualDisks` and `VirtualMachines`.

Since Oracle VM and PCA 2.x user a raw disk format and are always BIOS mode, the output is simpler than the VMware output, but is otherwise similar:

```
qemu-img convert -p -f raw -O qcow2 \
/OVS/Repositories/0004fb0000030000cc1d0d9dfba208d0/VirtualDisks/0004fb000012000008033187d7ded003
.img \ output/bootdisks/output.QCOW2
cp -f image_metadata.json-BIOS image_metadata.json
tar zcvf output/bootdisks/0004fb000006000051706efe190dcb16.oci image_metadata.json \
output/bootdisks/output.QCOW2
oci compute image import from-object-uri --uri
$IMAGEURLoutput/bootdisks/0004fb000006000051706efe190dcb16.oci --compartment-id $OCI_CLI_TENANCY
--display-name "Exalogic01_image"
oci compute instance launch --display-name "Exalogic01" --availability-domain ad1 -c
$OCI_COMPARTMENT --subnet-id $SUBNET --assign-public-ip true --shape VM.PCAStandard1.Flex --
shape-config '{"ocpus":4, "memoryInGBs":48}' --source-details
'{"bootVolumeSizeInGBs":'64',"imageId":"$IMAGEOCID","sourceType":"image"}'
```

```
#!/usr/bin/env python3
# ovmrepoparse: parse vm.cfg files from an Oracle VM repository and create CLI and hypervisor-
neutral metadata for export and import to PCA X9-2.
#
# Arguments:
#      1. Path to Oracle VM repository (what would be mounted on OVS as /OVS/Repositories/$UUID)
#      2. Path to store output, which will have subdirectories metadata, bootdisks, commands
#      example: $ ./ovmrepoparse.py /home/savit/projects/PCAbulk/OVM/TestRepository
/home/savit/projects/PCAbulk/OVM/output
# Note: this code assumes that hyperthreading is enabled on source platform. If not, just set
cores == thread

import os
import sys
import stat
import json
from shapetools import *

if (len(sys.argv)>=2 and sys.argv[1]=="?") or len(sys.argv)<3 :
    print ("Usage is:", sys.argv[0], " <OVM repo path> <outputpath>  Trailing '/' optional. ")
    quit()

sourceFS   = sys.argv[1].rstrip('/')
VirtualMachinesDir = sourceFS+"/VirtualMachines"
outputFS   = sys.argv[2].rstrip('/')
print(f'OVM repository is at {sourceFS}, VM configurations at {VirtualMachinesDir} and output is
stored under {outputFS}')
metadataFS   = outputFS+"/metadata/";  os.system("mkdir "+metadataFS)
commandsFS   = outputFS+"/commands/";  os.system("mkdir "+commandsFS)
bootdisksFS  = outputFS+"/bootdisks/"; os.system("mkdir "+bootdisksFS)

totalcores = totalgb = vmcount = toobigcount = 0; toobigVMs = ""; errorVMcount = 0; errorVMs =
''
GB=1024*1024*1024

# constant for PCA 3.0.x maximums for cores and memory (in GB), and maximums for using flex
shapes
MAXCORES        = 60
MAXGB          = 960
MAXFLEXCORES  = 32
MAXFLEXGB     = 512
MINBLOCKVOLGB = 50
launch_options_BIOS = '--launch-options \'{"boot-volume-
type":"PARAVIRTUALIZED","firmware":"BIOS","is-consistent-volume-naming-enabled":false,"is-pv-
encryption-in-transit-enabled": false,"network-type":"PARAVIRTUALIZED","remote-data-volume-
type": "PARAVIRTUALIZED"}\''

# constants for all OVM or PCA2.x VMs
SOURCEPLATFORM = 'OVM'
LAUNCHMODE     = 'BIOS'

nameMapFile    = metadataFS+"namemap.csv"
nameFD         = open(nameMapFile,'w')

VMs = os.listdir(VirtualMachinesDir)
for vmid in VMs:
    vmcount = vmcount + 1
    vmcfg = open(VirtualMachinesDir+"/"+vmid+"/vm.cfg")
    print(f'\n-------- {vmid} --------')
    vmname = vcpus = maxvcpus = memory = ""; guestOS = 'none'; bootdisk=''; blockdisk=''; bv =
[]; bootdiskfound = 0
```

```
    for v in vmcfg.readlines() :
        v = v.strip("\n")
        if len(v)==0 : continue
        keyword= v.split('=')[0].strip()
        value  = v.split('=')[1].strip().strip("'")
        if keyword=="OVM_simple_name" : vmname = value
        elif keyword=="disk" and value != '[]' : # seen in the wild - can have null disks
          disks=value.replace("'","").strip("[").strip("]").split(',')
          howmanydisks = len(disks)//3
          for diskcounter in range(howmanydisks) : # disk item in OVM vm.cfg disk= line is a 3-
tuple
              d=diskcounter*3
              diskitem = disks[d].strip().strip("'").strip('"')   # file:/PATH/TO/xxxxx.img
              diskdev  = disks[d+1]    # e.g. xvda or xvdb:cdrom
              diskmode = disks[d+2]    # e.g. r, w, w!
              if len(diskitem) == 0 or diskdev.find("cdrom") > -1 : continue
              if diskitem[0:5] != 'file:' :           # this sample program only handles vdisks
                 errorVMcount = errorVMcount+1; errorVMs = errorVMs + " " + vmid
                 print(f'{vmid}+"/vm.cfg has unsupported {v}, disk item {diskitem}')
                 break
              diskloc  = diskitem[5:]  # strip off the file: prefix

              try :         # get the disk image file size
                 filestat=os.stat(diskloc)
                 diskbytes = getattr(filestat,"st_size")
                 diskGB    = (diskbytes+GB-1)//GB
              except :
                 print(f'VM {vmid} has disk in unmounted repo. Using default size. Check
{diskloc}')
                 diskGB = MINBLOCKVOLGB
              if bootdisk == '' :                   # if first non-cdrom it's our boot disk
                 bootdisk = bootfile = diskloc; bootdiskGB = diskGB; bootdiskfound = 1
                 print(f'Boot disk size for {bootdisk} is {bootdiskGB}GB {diskbytes} bytes')
              else :
                 bv.append([diskloc,diskGB])
                 print (f'   Blockdisk {diskloc} {diskGB}GB')
        elif keyword == "vcpus" :       vcpus    = value
        elif keyword == "maxvcpus":     maxvcpus = value
        elif keyword == "memory"  :     memory   = value; mem = int(memory)  # memory in MB
        elif keyword == "OVM_os_type" : guestOS  = value

    if bootdiskfound == 0  :                # we never found a disk. This VM was never fully
defined
       print(f'VM {vmname} {vmid} has no disks. Skipping')
       errorVMcount = errorVMcount+1; errorVMs = errorVMs + " " + vmid
       continue

    if maxvcpus == '' :                     # found in real life, set to vcpus if present, else
1
       if vcpus != '' : maxvcpus = vcpus
       else :
          maxvcpus = '1'

    threads = int(maxvcpus)
    cores   = (threads+1)//2                     # Assumes hyperthreading. If not, just set
cores=threads
    totalcores = totalcores + cores
    gb=(int(mem)+1023)//1024
    totalgb = totalgb+gb
    if cores > MAXCORES or gb > MAXGB :
       print(f' #### VM {vmname} / {vmid} exceeds max shape: core {cores} memory {gb}GB ####')
```

```
        shape = "NO MATCHING SHAPE"
        errorVMcount = errorVMcount+1; errorVMs = errorVMs + " " + vmid
    elif cores <= MAXFLEXCORES and gb <= MAXFLEXGB :
        shapeconfig = "--shape-config '{\"ocpus\":" +str(cores)+ ", \"memoryInGBs\":" + str(gb)+
"}'"
        shape =  "VM.PCAStandard1.Flex " + shapeconfig
    else :
        shape   = size2shape(threads,mem)
    print (f'---- VM id {vmid}, VM name {vmname}, vcpus {vcpus}, maxvcpus {maxvcpus}, threads
{threads}, memory {memory}, shape {shape}')
    vmdict = {'SOURCEPLATFORM': SOURCEPLATFORM, 'VMNAME': vmname, 'UUID': vmid, 'INSTANCENAME':
vmid, 'SHAPE': shape, 'CORES': cores, 'MEMORY': memory, 'LAUNCHMODE': LAUNCHMODE, 'OS': guestOS,
'BOOTDISK': bootdisk, 'BLOCKVOLS': bv}
    convertcmd = 'qemu-img convert -p -f raw -O qcow2 '+bootfile+' '+bootdisksFS+'output.QCOW2'
    print(convertcmd)
    launchmodecmd   = 'cp -f image_metadata.json-BIOS image_metadata.json'
    print(launchmodecmd)
    tardotocicmd    = 'tar zcvf '+ bootdisksFS+vmid+'.oci image_metadata.json
'+bootdisksFS+'output.QCOW2'
    print(tardotocicmd)
    imageimportcmd  = 'oci compute image import from-object-uri --uri $IMAGEURL'+
bootdisksFS+vmid+'.oci --compartment-id $OCI_CLI_TENANCY'+' --display-name "'+vmname+'_image"'
    print(imageimportcmd)
    launchcmd = 'oci compute instance launch --display-name "' + vmname + '" --availability-
domain ad1 -c $OCI_COMPARTMENT --subnet-id $SUBNET --assign-public-ip true '
    launchcmd = launchcmd + '--shape ' + shape
    sourcedetails = ' --source-details
\'{"bootVolumeSizeInGBs":\''+str(bootdiskGB)+'\',"imageId":"$IMAGEOCID","sourceType":"image"}\''
    launchcmd = launchcmd + sourcedetails
    print(launchcmd)
    with open(commandsFS+vmid+'.sh','w') as cmdFD :
         cmdFD.write(convertcmd+"\n")
         cmdFD.write(launchmodecmd+"\n")
         cmdFD.write(tardotocicmd+"\n")
         cmdFD.write(imageimportcmd+"\n")
         cmdFD.write(launchcmd+"\n")
         for b in range(len(bv)) :
            bvcreate = 'oci bv volume create --availability-domain ad1 -c $OCI_COMPARTMENT --
size-in-gbs ' + str(bv[b][1]) + '  ### for ' + bv[b][0]
            print(bvcreate)
            cmdFD.write(bvcreate+"\n")

    nameMap = '"'+vmid+'","'+vmname+'"\n'
    nameFD.write(nameMap)
    with open(metadataFS+vmid+'.json','w') as j :
         json.dump(vmdict,j)
print(f'Parse done: {vmcount} VMs, {totalcores} cores, {totalgb} GB')
if errorVMcount != 0 : print(f'{errorVMcount} VMs had a configuration error: {errorVMs}')
nameFD.close()
```

CONCLUSION

Following the steps in this technical paper, a customer can import virtual machines from legacy hypervisor platforms to Oracle Private Cloud Appliance X9-2.

REFERENCES

See these reference documents for additional information:

- Oracle PCA 3.0.1 documentation library: https://docs.oracle.com/en/engineered-systems/private-cloud-appliance/
    - Oracle PCA Concepts Guide: https://docs.oracle.com/en/engineered-systems/private-cloud-appliance/3.0/concept-3.0.1/index.html
    - Oracle PCA User Guide: https://docs.oracle.com/en/engineered-systems/private-cloud-appliance/3.0/user-3.0.1/index.html
- Bring Your Own Image page for Oracle Cloud Infrastructure: https://docs.oracle.com/en-us/iaas/Content/Compute/References/bringyourownimage.htm
- "Introducing VirtIO Drivers with Oracle Solaris on OCI" at https://blogs.oracle.com/solaris/post/introducing-virtio-drivers-with-oracle-solaris-on-oci
- "Announcing Oracle VirtIO Drivers 1.1.5 for Microsoft Windows" at https://blogs.oracle.com/linux/post/announcing-oracle-virtio-drivers-115-for-microsoft-windows
- Removing Oracle VM Paravirtualization Drivers for Windows https://docs.oracle.com/en/virtualization/oracle-vm-pv-drivers/3.4.5/winpv/vmwpv-uninstall.html
- Oracle VM Server for x86 and Oracle VM Manager documentation library at https://docs.oracle.com/en/virtualization/oracle-vm/index.html
- Oracle VM VirtualBox documentation at https://docs.oracle.com/en/virtualization/virtualbox/index.html
- Oracle Linux KVM User's Guide https://docs.oracle.com/en/operating-systems/oracle-linux/kvm-user/
- cloud-init documentation https://cloudinit.readthedocs.io/en/latest/
- Solaris 11.4 documentation on cloudbase-init https://docs.oracle.com/cd/E37838_01/html/E60974/gpxcg.html
- "Introducing VirtIO Drivers with Oracle Solaris on OCI" at https://blogs.oracle.com/solaris/post/introducing-virtio-drivers-with-oracle-solaris-on-oci
- Windows cloudbase-init documentation https://cloudbase-init.readthedocs.io/en/latest/
- Rackware and PCA: https://blogs.oracle.com/oracle-systems/post/rackware-and-oracle-pca-x9-2---delivering-migration-backup-and-disaster-recovery-together
- Oracle Private Cloud Appliance 2.4 Backup Guide at https://www.oracle.com/technetwork/server-storage/private-cloud-appliance/documentation/pca-backup-x8-0-6-5676918.pdf

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Oracle Private Cloud Appliance X9-2
Workload Import
November 2222
Author: [OPTIONAL]
Contributing Authors: [OPTIONAL]