Containers on Compute Engine

Software containers are a convenient way to run your apps in multiple isolated user-space instances. You can run containers on Linux or Windows Server public VM images (/compute/docs/images#os-compute-support), or on a Container-Optimized OS (/container-optimized-os/docs) image. Containers let your apps run with fewer dependencies on the host virtual machine (VM) and run independently from other containerized apps that you deploy to the same VM instance. These characteristics make containerized apps more portable, easier to deploy, and easier to maintain at scale.

This document describes some of the more common container technologies that you can use to run containers on Compute Engine instances. You can use these technologies on most of the public VM images (/compute/docs/images#os-compute-support) that Compute Engine provides.

Run containers on Compute Engine when you need complete control over your container environment and your container orchestration tools. Alternatively, you can use Google Kubernetes Engine (/kubernetes-engine) to simplify cluster management and container orchestration tasks so that you don't need to manage the underlying VM instances.

Container technologies that run on Compute Engine

In general, Compute Engine instances can run almost any container technology or tool. You can run several different types of containers on modern Linux operating systems and you can also run Docker on Windows Server (#docker_on_windows) 2016 or later. The following list includes several common tools that you can use to run and manage containerized apps:

- Docker (https://docs.docker.com/) and Podman (https://podman.io/) are two popular container technologies that let you run containerized apps.
- Kubernetes (https://kubernetes.io/) is a container orchestration platform that you can use to manage and scale your running containers across multiple instances or within a hybrid-cloud environment.
- Containers on Compute Engine (/compute/docs/containers/deploying-containers) provide an easy way to deploy containers to Compute Engine VM instances or managed instance groups (/compute/docs/instance-groups#managed_instance_groups).

- You can convert your existing systems into LXD images (https://www.ubuntu.com/cloud/lxd) and run them within Compute Engine VM instances for a lift-and-shift migration solution. LXD runs on Ubuntu images.

Additionally, you can use Container Registry (/container-registry/docs) to manage container image versions. Container Registry serves as a central location to store and manage your container images before you deploy those images to Kubernetes on Compute Engine or to Google Kubernetes Engine (/kubernetes-engine) clusters.

Container-optimized VM images

Compute Engine provides several public VM images (/compute/docs/images#os-compute-support) that you can use to create instances and run your container workloads. Some of these public VM images have a minimalistic container-optimized operating system that includes newer versions of Docker, Podman, or Kubernetes preinstalled. The following public image families are designed specifically to run containers:

- Container-Optimized OS from Google (/container-optimized-os/docs)
    - Includes: Docker, Kubernetes
    - Image project: `cos-cloud`
    - Image family: `cos-stable`
- Fedora CoreOS (https://getfedora.org/en/coreos/)
    - Includes: Podman, Docker
    - Image project: `fedora-coreos-cloud`
    - Image family: `fedora-coreos-stable`
- Ubuntu (https://www.ubuntu.com/)
    - Includes: LXD
    - Image project: `ubuntu-os-cloud`
    - Image family: `ubuntu-2004-lts`
- Windows (https://www.microsoft.com/)

- Includes: Docker
- Image project: `windows-cloud`
- Image family: `windows-1909-core-for-containers`

If you need to run specific container tools and technologies on images that do not include them by default, install (#installing) those technologies manually.

Installing container technologies on your instances

To launch a single container on an instance, you can specify a container image when you create an instance (/compute/docs/instances/create-start-instance#from-container-image). Compute Engine automatically supplies an up-to-date Container-Optimized OS image with Docker installed and launches your container when the VM starts up. For more information, see Deploying containers on VMs (/compute/docs/containers/deploying-containers).

Alternatively, you can run your container workloads on Compute Engine using whatever container technologies and orchestration tools that you need. You can create an instance (/compute/docs/instances/create-start-instance) from a public VM image (/compute/docs/images#os-compute-support) and then install the container technologies that you want. For example:

- Install Docker on Compute Engine instances so that you can run your Docker container images on those instances.
  - Install Docker on Linux instances (https://docs.docker.com/engine/installation/linux/)
  - Install Docker on Windows Server instances (#docker_on_windows)
- Install Podman (https://podman.io/) on Compute Engine instances as an alternative to the Docker container runtime.
- Install Kubernetes (https://kubernetes.io/docs/getting-started-guides/kubeadm/) on your instances to provide container orchestration for both Docker and Open Container Initiative (OCI) containers.

In some situations, you might require specific versions of these technologies to ensure that they operate together correctly. For example, Kubernetes usually runs best with specific versions of Docker. Typically, you can install the latest versions of these technologies for the best result.

Installing Docker on Windows Server images

Windows Server 2016 and later versions include container support. If you plan to run Docker containers on a Windows Server instance, Google recommends starting with the Windows Server for Containers public image (/compute/docs/images#windows_server). This image has the following components installed:

- Docker (https://www.docker.com/docker-windows).
- Microsoft's `servercore` (https://hub.docker.com/_/microsoft-windows-servercore) and `nanoserver` (https://hub.docker.com/_/microsoft-windows-nanoserver) container images.
- All required fixes for known issues (/compute/docs/containers#windows_known_issues).

If you want to install Docker on the Windows Server base image and run containerized apps instead of using the Windows Server for Containers image, then follow the steps outlined below.

Start by creating a Windows Server instance (/compute/docs/instances/windows/creating-managing-windows-instances) using a Windows Server 2016 or later public image (/compute/docs/images#os-compute-support). For the best container support, we recommend that you use the most recent semi-annual release (https://docs.microsoft.com/en-us/windows-server/get-started-19/servicing-channels-19) of Windows Server, such as Windows Server, version 1909.

Install Docker

1. Connect to the Windows instance (/compute/docs/instances/connecting-to-instance#windows).
2. Run PowerShell as an administrator.
3. Install Docker from the Microsoft repositories:

```
PS C:\> Install-Module -Name DockerMsftProvider -Repository PSGallery -Force
```

```
PS C:\> Install-Package -Name docker -ProviderName DockerMsftProvider
```

4. Run the following commands to work around known issues (#windows_known_issues) with Windows containers on Compute Engine:

   - Disable Receive Segment Coalescing:

     ```
     PS C:\> netsh netkvm setparam 0 *RscIPv4 0
     ```

   - Enable IPv6:

     ```
     PS C:\> reg add HKLM\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters `
     /v DisabledComponents /t REG_DWORD /d 0x0 /f
     ```

5. Restart the instance:

   ```
   PS C:\> Restart-Computer -Force
   ```

Additional setup steps

At this point you can use Docker to run containers in the instance. For example, the following command downloads the Windows `nanoserver` container image and runs a command prompt inside a `nanoserver` container:

```
\> docker run -it mcr.microsoft.com/windows/nanoserver:1909 cmd.exe
```

Note that there is a known issue with Docker's default network MTU that affects connectivity to the instance and connectivity from containers to the internet. To work around this issue, first set the MTU for all network interfaces (both Ethernet and vEthernet) to `1460` by running the following commands in a PowerShell terminal on the instance:

```
\> Get-NetAdapter | Where-Object Name -like "*Ethernet*" | ForEach-Object {
etsh interface ipv4 set subinterface $_.InterfaceIndex mtu=1460 store=persistent
```

```
\> netsh interface ipv4 show subinterfaces
U  MediaSenseState   Bytes In  Bytes Out  Interface
-  ---------------  ---------  ---------  -------------
67295                1          0          0  Loopback Pseudo-Interface 1
0                1     306804     668688  Ethernet
0                1          0       1282  vEthernet (nat)
```

Even after repairing the instance's MTU, connectivity from containers to the internet might be unstable because, by default, the container's network interface also uses an MTU of `1500`. For commands to set the MTU correctly for every container, see the container MTU (#container_mtu) section.

You might need to periodically re-execute these MTU commands as you configure Docker networking. For full details, see the known issues (#mtu_failures) section.

Running Windows containers

There are many resources available for getting started with Windows containers:

- The Running Windows containers on Compute Engine (https://codelabs.developers.google.com/codelabs/cloud-windows-containers-computeengine/#0) codelab provides an introduction to Windows containers on Compute Engine.
- Microsoft provides extensive Windows containers documentation (https://docs.microsoft.com/en-us/virtualization/windowscontainers/).
- Docker Hub (https://hub.docker.com/) can be used as a repository for storing and pulling Windows containers.

Known issues with Windows containers

Containers are incompatible across Windows versions

Containers built on earlier versions of Windows don't work in Compute Engine instances running more recent versions of Windows. Docker pulls the Windows Server 2016 version of a container by default. This means that running the following command in an instance running Windows Server version 1709 or newer results in an error:

```
\> docker run -it mcr.microsoft.com/windows/nanoserver cmd.exe
r: Error response from daemon: container
8bbcba4e91792be65f3c40b5a1aee062f02fbc60a78444b47d043438069 encountered an
 during CreateContainer: failure in a Windows system call: The operating
m of the container does not match the operating system of the host.
370101)
```

Microsoft's Windows container version compatibility (https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/version-compatibility) page contains more information. To work around Windows container version incompatibilities, specify the tag corresponding to your Windows version when pulling and running containers. For example, in a Windows Server, version 1909 instance, use the following command to run a command prompt in the version 1909 `nanoserver` container instead of the default 2016 container:

```
\> docker run -it mcr.microsoft.com/windows/nanoserver:1909 cmd.exe
```

MTU incompatibilities affect instance and container connectivity

When you create a container network on a Windows instance using the `docker network create` or `New-VMSwitch` commands, the MTU of the instance's network interface is typically forced to `1500` (https://github.com/Microsoft/SDN/issues/84). The default network interface inside of a new Docker container also typically uses an MTU of `1500` (https://github.com/moby/moby/issues/35683). Google Cloud supports an MTU of only `1460`, so when the MTU is forced to `1500` you might experience the following issues:

- The RDP session can stop and you might be unable to reconnect. This is known to happen when creating a transparent container network (https://docs.microsoft.com/en-us/virtualization/windowscontainers/container-networking/network-drivers-topologies) .
- DNS resolution inside the container might fail.
- DNS resolution is successful, but establishing an HTTP connection from the container to the internet might fail.

Working around these limitations requires two steps: setting the MTU for the instance's network interfaces (#instance_mtu) and setting the MTU for the container network interfaces (#container_mtu).

1. Setting the MTU for the Windows instance's network interfaces

Run the following command in a PowerShell terminal on the Windows instance to set the MTU for all network interfaces (both Ethernet and vEthernet):

```
\> Get-NetAdapter | Where-Object Name -like "*Ethernet*" | ForEach-Object {
etsh interface ipv4 set subinterface $_.InterfaceIndex mtu=1460 store=persistent
```

Check that the instance's Ethernet and vEthernet interface MTUs are set to `1460` using this command:

```
\> netsh interface ipv4 show subinterfaces
U  MediaSenseState   Bytes In  Bytes Out  Interface
-  ---------------  ---------  ---------  -------------
67295                1          0          0  Loopback Pseudo-Interface 1
0                1  628295912    2613170  Ethernet
0                1      37793     223909  vEthernet (nat)
```

If you are unable to run these commands because you can no longer connect to an instance by using RDP, you can connect to the instance through the serial console (/compute/docs/instances/interacting-with-serial-console), start a `cmd` prompt and run the `netsh` commands there to repair the MTU. To avoid having to do this, we recommend executing any `docker network ...` or `New-VMSwitch` commands as part of a script that also executes the MTU repair command.

2. Setting the MTU for the Windows container network interfaces

The MTU for a Windows container must be set while the container is running, either from inside the container or from the instance hosting the container. If PowerShell is available in your container, you can run this command interactively or from a script in the container to correctly set the MTU:

```
\> Get-NetAdapter | Where-Object Name -like "vEthernet*" | ForEach-Object {
etsh interface ipv4 set subinterface $_.InterfaceIndex mtu=1460 store=persistent
```

Or, you can run this command on the Windows instance to set the MTU for all running containers:

```
\> Get-NetIPInterface -IncludeAllCompartments |
re-Object InterfaceAlias -like "vEthernet*" |
-NetIPInterface -IncludeAllCompartments -NlMtuBytes 1460
```

Docker containers fail to start

Starting a container with `docker run` can fail with the following error:

```
ogram Files\Docker\docker.exe: Error response from daemon: container ...
ntered an error during CreateContainer: failure in a Windows system call:
nt not found. (0x490)
```

This issue (https://github.com/moby/moby/issues/32595) occurs on Windows Server 2016 instances with Windows Update KB4015217 installed. To work around the problem, enable IPv6 on the instance using the following PowerShell command:

```
\> reg add HKLM\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters `
sabledComponents /t REG_DWORD /d 0x0 /f
```

After you enable IPv6, restart the instance:

```
\> Restart-Computer -Force
```

If this issue is corrected in future operating system updates, you can restore the original IPv6 setting:

```
\> reg add HKLM\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters `
sabledComponents /t REG_DWORD /d 0xff /f
```

Hyper-V containers fail to start

Hyper-V containers (https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/hyperv-container) are not supported on Compute Engine at this time.

What's next

- Create and start an instance (/compute/docs/instances/create-start-instance) that you can use to run container applications.
- Learn about Compute Engine instances (/compute/docs/instances).
- Learn more about Google Kubernetes Engine (/kubernetes-engine), which you can use to run your containers on Google Cloud without managing Compute Engine instances yourself.
- Learn more about Kubernetes (https://kubernetes.io/).
- Learn how to use Container Registry (/container-registry/docs) to store your container images privately within Google Cloud.

Last updated 2020-07-30 UTC.