



登临 Hamming™ 容器组件 使用手册

DL-DG/SW-020A-11

2025-01-20

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州上海登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

email : support@denglin.ai

更新历史

版本	更新描述
11	container runtime 新增 dl-docker 工具。
10	组件更名：denglin-sdk改为denglin-driver，denglin-sdk-manager改为denglin-version-manager。 v2取消对DisableSDKMount参数支持。
09	新增对 podman 的支持。 对于 docker 运行时，卸载容器组件命令中新增自动删除容器选项。
08	新增对containerd和crio的支持。
07	新增支持的架构和系统。 新增容器运行时日志选项。 新增挂载 SDK 选项。
06	删除 ddocker 工具及相关说明。
05	更新安装步骤，并增加卸载步骤，更新配置与使用说明。 更新系统及软件版本要求。
04	文档重命名为 登临系统软件容器组件使用手册 ，并将登临系统软件 Kubernetes 组件的相关内容移到了文档 登临系统软件 Kubernetes 组件使用手册 。
03	新增了 ddocker, CRI-O, containerd, rootless docker, swarm, denglin-gpu-exporter, Helm, dssi 的相关说明。
02	章节 2 系统及软件版本要求 : 增加了 NFD 的版本要求。 新增了章节 3 组件介绍 ，4 安装 ，5 部署与使用 ，6 部署示例 ，详细介绍了组件，提供了安装和部署的详细步骤，以及示例。
01	第一次发布。

章节目录

章节目录

1 概述

1.1 功能

1.2 术语和缩略语

2 系统及软件版本要求

3 安装与卸载

3.1 安装

3.2 卸载

4 配置

4.1 配置登临 runtime

4.2 配置 denglin_container.conf

5. 使用

5.1 Docker 启动

5.2 dl-docker 启动

1 概述

1.1 功能

登临系统软件 DSS (DengLin System Software) 为登临 Goldwasser™ 人工智能加速卡 (GPGPU) 在硬件加速计算的各种应用场景提供应用层支持，包括对容器和 Kubernetes 使用环境的支持及通用的系统管理和监测。

本文档主要介绍登临系统软件 DSS 容器组件软件包的功能、安装步骤、配置以及用法，指导用户快速安装搭建软件应用环境，在容器中使用登临 GPGPU。

1.2 术语和缩略语

术语	简介
DSS	DengLin System Software 的简称，即登临系统软件
Hamming SDK	登临 Goldwasser™ 人工智能加速卡的软件包
GPGPU	通用GPU
Kubernetes	用于自动部署，扩展和管理容器化应用程序的开源系统
Runtime	容器运行时，为容器设置环境变量，挂载 Hamming SDK 文件，分配 GPGPU 设备

2 系统及软件版本要求

目前 DSS 容器组件在如下环境中能顺利安装、运行：

GPGPU 版本	架构	操作系统
V1	x86_64, aarch64, loongarch64, sw_64	Ubuntu version 16.04 and later versions Centos version 7.6 and later versions Kylin version 10 and later versions Loongnix version 8 and later versions Uos version 20 and later versions
V2	x86_64	Ubuntu version 16.04 and later versions Centos version 7.6 and later versions Kylin version 10 and later versions OpenEuler 22.03 and later versions

支持的其他软件及版本如下：

- Docker version 17.12.0 and later versions
- Containerd version v1.5.2 and later versions
- CRI-O version v1.20.1 and later versions
- Podman version v1.6.4 and later versions

3 安装与卸载

3.1 安装

登临 GPGPU 容器运行依赖宿主机上的 kmd，kmd 在 SDK 安装过程中会自动安装，安装步骤请见《登临Hamming SDK安装指南》。

用户应确保系统及所需软件版本正确。执行安装命令后系统会自动安装运行登临 GPGPU 容器所需文件。

执行安装命令：

```
sudo bash denglin_hamming_<device_version>-container-<version>-<platform>-<arch>.run [--target <extract_path>] -- [runtime] [options]
```

- 参数 `--target <extract_path>` 指定 `denglin_hamming_<device_version>-container-<version>-<platform>-<arch>.run` 的解压缩路径。如果不指定，默认解压到当前路径下 `${pwd}/denglin_hamming_<device_version>-container-<version>-<platform>-<arch>`。
- `runtime` 为容器运行时类型，可选 `docker`、`containerd`、`crio`、`podman`，默认为 `docker`。
- `options` 为可选配置，如不设置则为默认值，如下表所示：

名称	类型	描述	默认值
<code>--config</code>	string	配置文件路径	docker: /etc/docker/daemon.json containerd: /etc/containerd/config.toml crio:/etc/crio/crio.conf podman:/etc/containers/libpod.conf
<code>--socket</code>	string	socket 路径，对于 crio、podman 运行时不支持该选项	docker: /var/run/docker.sock containerd: /run/containerd/containerd.sock
<code>--runtime-name</code>	string	登临容器运行时名称	dirt
<code>--runtime-dir</code>	string	登临容器运行时相关文件安装位置，用于 K8S 环境须设置为 /usr/local/bin，即默认值	/usr/local/bin
<code>--set-as-default</code>	bool	是否设置登临容器运行时为默认，用于 K8S 环境须设置为 true	false

安装成功后会在屏幕上打印信息：

```
Successfully install files to the destination
```

3.2 卸载

切换到安装过程中 `denglin_hamming_<device_version>-container-<version>-<platform>-<arch>.run` 解压得到的目录。或执行以下命令重新解压 `denglin_hamming_<device_version>-container-<version>-<platform>-<arch>.run`，再切换到解压得到的目录。

```
bash denglin_hamming_<device_version>-container-<version>-<platform>-<arch>.run [--target <extract_path>] --noexec
```

- 参数 `--target <extract_path>` 指定 `denglin_hamming_<device_version>-container-<version>-<platform>-<arch>.run` 的解压缩路径。如果不指定，默认解压到当前路径下 `${pwd}/denglin_hamming_<device_version>-container-<version>-<platform>-<arch>`。
- `runtime` 为容器运行时类型，可选 `docker`、`containerd`、`crio`，默认为 `docker`。
- `options` 为可选配置，如不设置则为默认值。除 `--set-as-default` 不支持，其余配置与安装的配置相同，见 [3.1 安装](#) 步骤中的说明。

执行以下命令进行卸载：

```
sudo bash install_container.sh cleanup [runtime] [options]
```

- `runtime` 为容器运行时类型，可选 `docker`、`containerd`、`crio`、`podman`，默认为 `docker`。
- `options` 为可选配置，如不设置则为默认值，如下表所示：

名称	类型	描述	默认值
<code>--config</code>	string	配置文件路径	docker: /etc/docker/daemon.json containerd: /etc/containerd/config.toml crio:/etc/crio/crio.conf podman:/etc/containers/libpod.conf
<code>--socket</code>	string	socket 路径，对于 crio、podman 运行时不支持该选项	docker: /var/run/docker.sock containerd: /run/containerd/containerd.sock
<code>--runtime-name</code>	string	登临容器运行时名称	dlrt
<code>--runtime-dir</code>	string	登临容器运行时相关文件安装位置，用于 K8S 环境须设置为 /usr/local/bin，即默认值	/usr/local/bin

名称	类型	描述	默认值
--auto-delete	bool	该配置仅在 runtime 为 docker 或 podman 时有效， 设置为 true 会在卸载登临容器运行时前自动删除 所有使用该运行时的容器。 设置为 false 则不会自动删除使用登临容器运行 时的容器，应确认此时已经没有使用登临容器运 行时的容器存在，否则卸载登临运行时 会失败， 需要手动删除使用登临容器运行时的容 器后再次执行卸载。	false

卸载成功后会在屏幕上打印信息：

```
Successfully remove files
```

4 配置

4.1 配置登临 runtime

Docker

软件包安装成功后，会自动添加登临 runtime 到配置文件 `/etc/docker/daemon.json` 中，并重启 docker 以启用配置。安装后会更新配置如下：

```
$ cat /etc/docker/daemon.json
{
  "runtimes": {
    "dlrt": {
      "path": "/usr/local/bin/denglin_runtime",
      "args": []
    }
  }
}
```

其中 `dlrt` 是为登临 runtime 定义的名字，`path` 的值是二进制文件 `denglin_runtime` 的绝对路径。

安装成功后 `dlrt` 不是默认运行时，如需将 `dlrt` 设置为默认运行时，通过 `default-runtime` 进行设置，并重启 docker 以启用配置。配置如下：

```
$ cat /etc/docker/daemon.json
{
  "default-runtime": "dlrt",
  "runtimes": {
    "dlrt": {
      "path": "/usr/local/bin/denglin_runtime",
      "args": []
    }
  }
}
```

注意：

在 `rootless` 模式下，首先需要将 `daemon.json` 文件放到用户 `home` 目录下的 `.config/docker` 中，例如 `/home/testuser/.config/docker/daemon.json`；然后执行命令 `systemctl --user restart docker` 使配置生效。如果用户没有权限访问日志的路径，需要将日志输出路径修改为用户有权访问的路径。

containerd

软件包安装成功后，会自动添加登临 runtime 到配置文件 `/etc/containerd/config.toml` 中，并重启 containerd 以启用配置。安装后会更新配置如下：

```
$ cat /etc/containerd/config.toml
version = 2

[plugins]
```

```
[plugins."io.containerd.grpc.v1.cri"]

[plugins."io.containerd.grpc.v1.cri".containerd]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.dlrt]
  privileged_without_host_devices = false
  runtime_type = "io.containerd.runc.v2"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.dlrt.options]
  BinaryName = "/usr/local/bin/denglin_runtime"
```

其中 `dlrt` 是为登临 runtime 定义的名字，`BinaryName` 的值是二进制文件 `denglin_runtime` 的绝对路径。

安装成功后 `dlrt` 不是默认运行时，如需将 `dlrt` 设置为默认运行时，通过 `default_runtime_name` 进行设置，并重启 `containerd` 以启用配置。配置如下：

```
$ cat /etc/containerd/config.toml
version = 2

[plugins]

[plugins."io.containerd.grpc.v1.cri"]

[plugins."io.containerd.grpc.v1.cri".containerd]
  default_runtime_name = "dlrt"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.dlrt]
  privileged_without_host_devices = false
  runtime_type = "io.containerd.runc.v2"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.dlrt.options]
  BinaryName = "/usr/local/bin/denglin_runtime"
```

crio

软件包安装成功后，会自动添加登临 runtime 到配置文件 `/etc/crio/crio.conf` 中，并重启 `crio` 以启用配置。安装后会更新配置如下：

```
$ cat /etc/crio/crio.conf
[crio]

[crio.runtime]

[crio.runtime.runtimes]

[crio.runtime.runtimes.dlrt]
runtime_path = "/usr/local/bin/denglin_runtime"
runtime_root = "/run/runc"
runtime_type = "oci"
```

其中 `dlrt` 是为登临 runtime 定义的名字，`runtime_path` 的值是二进制文件 `denglin_runtime` 的绝对路径。

安装成功后 `dlrt` 不是默认运行时，如需将 `dlrt` 设置为默认运行时，通过 `default_runtime` 进行设置，并重启 crio 以启用配置。配置如下：

```
$ cat /etc/crio/crio.conf
[crio]

[crio.runtime]
default_runtime = "dlrt"

[crio.runtime.runtimes]

[crio.runtime.runtimes.dlrt]
runtime_path = "/usr/local/bin/denglin_runtime"
runtime_root = "/run/runc"
runtime_type = "oci"
```

podman

软件包安装成功后，会自动添加登临 runtime 到配置文件 `/etc/containers/libpod.conf` 中。安装后会更新配置如下：

```
$ cat /etc/containers/libpod.conf
runtime = "runc"
runtime_supports_json = ["dlrt", "runc"]
[runtimes]
dlrt = ["/usr/local/bin/denglin_runtime"]
runc = ["/usr/bin/runc"]
```

其中 `dlrt` 是为登临 runtime 定义的名字，`runtimes.dlrt` 中的值是二进制文件 `denglin_runtime` 的绝对路径。

安装成功后 `dlrt` 不是默认运行时，如需将 `dlrt` 设置为默认运行时，通过 `runtime` 进行设置。配置如下：

```
$ cat /etc/containers/libpod.conf
runtime = "dlr"
runtime_supports_json = ["dlrt", "runc"]
[runtimes]
  dlrt = ["/usr/local/bin/denglin_runtime"]
  runc = ["/usr/bin/runc"]
```

4.2 配置 denglin_container.conf

用户可以通过修改配置文件 `denglin_container.conf` 的内容设置容器的特性。配置文件 `denglin_container.conf` 必须放在 `/etc/denglin` 目录下，配置信息以 JSON 格式保存。各配置项的含义如下表：

配置项	类型	默认值	支持版本	含义
OutputDlrtLog	bool	false	v1, v2	日志是否开启。true：开启日志，false：关闭日志
LogLevel	string	info	v1, v2	日志级别。由高到低依次为 trace，debug，info，warning，error，fatal，panic
DlrtLogPath	string	空	v1, v2	日志文件路径
DisableSDKMount	bool	true	v1	SDK 挂载是否禁用，禁用后不在容器内挂载宿主机的 SDK。true：禁用 SDK 挂载，false：启用 SDK 挂载

容器中可以使用的宿主机或者容器内的 SDK，两种使用方式如下：

- 使用宿主机 SDK：设置 `/etc/denglin/denglin_container.conf` 配置项 `DisableSDKMount` 为 `false`，启动容器会挂载并使用宿主机上的 SDK。
- 使用容器内 SDK：设置 `/etc/denglin/denglin_container.conf` 配置项 `DisableSDKMount` 为 `true`，启动容器不会挂载宿主机上的 SDK，需要在容器内安装 SDK，可以在制作镜像时或在容器启动后安装 SDK，容器内安装 SDK 步骤请见《登临Hamming SDK安装指南》。

5. 使用

- 由于 `podman` 兼容 `docker` 命令，因此为了避免重复说明，请在安装了 `podman` 的环境下，将示例命令 `docker` 替换为 `podman`。
- `containerd` 和 `crio` 需要配合 K8S 使用，使用说明详见《登临Hamming K8S使用手册》。
- 示例使用宿主机 SDK，即设置 `/etc/denglin/denglin_container.conf` 配置项 `DisableSDKMount` 为 `false`。
- 下面是命令行启动登临 GPGPU 容器的示例。如果将登临 runtime 设置为默认运行时，可以省略 `--runtime=dllrt` 选项，设置方式参考[4.1 配置登临 runtime](#)。

5.1 Docker 启动

- 通过索引 `<GPUDevicesIndex>` 指定所需要的 GPGPU。如果需要多个 GPGPU，使用逗号隔开。

```
docker run --runtime=dllrt -e DENGlin_DEVICES=<GPUDevicesIndex> ubuntu:18.04 dlsmi
```

例如：

```
docker run --runtime=dllrt -e DENGlin_DEVICES=1 ubuntu:18.04 dlsmi
```

```
docker run --runtime=dllrt -e DENGlin_DEVICES=0,1,2 ubuntu:18.04 dlsmi
```

- 通过 `GPU-<UUID>` 指定所需要的 GPGPU。如果需要多个 GPGPU，使用逗号隔开。

```
docker run --runtime=dllrt -e DENGlin_DEVICES=GPU-<UUID> ubuntu:18.04 dlsmi
```

例如：

```
docker run --runtime=dllrt -e DENGlin_DEVICES=GPU-ad2367dd-a40e-6b86-6fc3-c44a2cc92c7e
ubuntu:18.04 dlsmi
```

```
docker run --runtime=dllrt -e DENGlin_DEVICES=GPU-ad2367dd-a40e-6b86-6fc3-c44a2cc92c7e,GPU-
16a23983-e73e-0945-2095-cdeb50696982 ubuntu:18.04 dlsmi
```

- 在 MIG 模式下，通过 `MIG-GPU-<UUID>/<GPU instance ID>` 指定所需要的 device。如果需要多个 device，使用逗号隔开：

```
docker run --runtime=dllrt -e DENGlin_DEVICES=MIG-GPU-<UUID>/<GPU instance ID> ubuntu:18.04
dlsmi
```

例如：

```
docker run --runtime=dllrt -e DENGlin_DEVICES=MIG-GPU-ad2367dd-a40e-6b86-6fc3-c44a2cc92c7e/0
ubuntu:18.04 dlsmi
```

```
docker run --runtime=dllrt -e DENGlin_DEVICES=MIG-GPU-ad2367dd-a40e-6b86-6fc3-
c44a2cc92c7e/0,MIG-GPU-16a23983-e73e-0945-2095-cdeb50696982/1 ubuntu:18.04 dlsmi
```

注意：

其中 `<UUID>` 为 GPGPU 实体卡的 UUID，并非 MIG 设备的 UUID。

- 在 MIG 模式下，通过 `<GPUDevicesIndex>:<MIGDeviceIndex>` 指定所需要的 device。如果需要多个 device，使用逗号隔开：

```
docker run --runtime=dllrt -e DENGlin_DEVICES=<GPUDevicesIndex>:<MIGDeviceIndex>
ubuntu:18.04 dlsmi
```

例如：

```
docker run --runtime=d1rt -e DENGILIN_DEVICES=0:0 ubuntu:18.04 d1smi
```

```
docker run --runtime=d1rt -e DENGILIN_DEVICES=0:0,0:1,2:3 ubuntu:18.04 d1smi
```

- 使用 `all` 指定机器上全部的 GPGPU，包括 MIG device。

```
docker run --runtime=d1rt -e DENGILIN_DEVICES=all ubuntu:18.04 d1smi
```

有三种参数格式用于指定将要使用的 GPGPU：

- 普通模式下：

```
<GPUDevicesIndex>
```

```
GPU-<UUID>
```

```
all
```

- MIG 模式下：

```
MIG-GPU-<UUID>/<GPU instance ID>
```

```
<GPUDevicesIndex>:<MIGDeviceIndex>
```

```
all
```

在 MIG 模式下，用户可以将登临 GPGPU 切分成多个 GPU instance。GPU instance 之间独立工作，互不干扰。使用时可以利用每个 GPU instance 执行不同的任务，从而提高利用率。每个 GPU instance 都有对应的 ID (GPU instance ID) 和 Index (MIGDeviceIndex)，用户可以通过 GPU instance ID 和 MIGDeviceIndex 指定使用哪个 GPU instance。

可以使用 `d1smi` 命令查看主机上的登临 GPGPU 的详细信息，包括 GPU instance ID，MIGDeviceIndex，内存大小，温度等。使用命令 `d1smi -L` 可以获得每个设备对应的 UUID。

5.2 d1-docker 启动

为了简化docker的使用，当 `runtime` 为 `docker` 的时候，安装包提供了 `d1-docker` 脚本，只需要通过 `DL_GPU` 环境变量设置需要挂载的设备即可完成5.1章节的操作，设备格式和上面的描述相同，如果 `DL_GPU` 为空或者是不指定，则行为和docker一致。

- 挂载所有GPGPU设备

```
DL_GPU=all d1-docker run --rm ubuntu:18.04 d1smi
```

- 挂载MIG设备

```
DL_GPU='0:0,0:1' d1-docker run --rm ubuntu:18.04 d1smi
```

- 不挂载GPGPU设备

```
DL_GPU='' d1-docker run --rm ubuntu:18.04 ls
```

```
d1-docker run --rm ubuntu:18.04 ls
```

登临科技保密材料