

---

# **Craft Providers**

***Release 1.4.2***

**Canonical Ltd.**

**Sep 15, 2022**



**PUBLIC APIS:**

<b>1</b>	<b>Executors</b>	<b>3</b>
1.1	Abstract Executor . . . . .	3
1.2	LXD Executor . . . . .	5
1.3	Multipass Executor . . . . .	8
<b>2</b>	<b>Bases</b>	<b>13</b>
2.1	Abstract Base . . . . .	13
2.2	Builddd Base . . . . .	15
<b>3</b>	<b>craft_providers package</b>	<b>19</b>
3.1	Subpackages . . . . .	19
3.2	Submodules . . . . .	73
3.3	Module contents . . . . .	78
<b>4</b>	<b>Indices and tables</b>	<b>83</b>
	<b>Python Module Index</b>	<b>85</b>
	<b>Index</b>	<b>87</b>



Here you will find all of the provider documentation...



## EXECUTORS

### 1.1 Abstract Executor

**class** `craft_providers.Executor`

Bases: `abc.ABC`

Interfaces to execute commands and move data in/out of an environment.

**abstract** `delete()`

Delete instance.

**Return type** `None`

**abstract** `execute_popen(command, *, cwd=None, env=None, **kwargs)`

Execute a command in instance, using `subprocess.Popen()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (`List[str]`) – Command to execute.
- **env** (`Optional[Dict[str, Optional[str]]]`) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments to pass.
- **cwd** (`Optional[Path]`) –

**Return type** `Popen`

**Returns** `Popen` instance.

**abstract** `execute_run(command, *, cwd=None, env=None, **kwargs)`

Execute a command using `subprocess.run()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (`List[str]`) – Command to execute.
- **env** (`Optional[Dict[str, Optional[str]]]`) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to `subprocess.run()`.
- **cwd** (`Optional[Path]`) –

**Return type** `CompletedProcess`

**Returns** Completed process.

**Raises** `subprocess.CalledProcessError` – if command fails and `check` is `True`.

**abstract exists()**

Check if instance exists.

**Return type** `bool`

**Returns** `True` if instance exists.

**abstract pull\_file(\*, source, destination)**

Copy a file from the environment to host.

**Parameters**

- **source** (`PurePath`) – Environment file to copy.
- **destination** (`Path`) – Host file path to copy to. Parent directory (`destination.parent`) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **ProviderError** – On error copying file.

**Return type** `None`

**abstract push\_file(\*, source, destination)**

Copy a file from the host into the environment.

**Parameters**

- **source** (`Path`) – Host file to copy.
- **destination** (`PurePath`) – Target environment file path to copy to. Parent directory (`destination.parent`) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **ProviderError** – On error copying file.

**Return type** `None`

**abstract push\_file\_io(\*, destination, content, file\_mode, group='root', user='root')**

Create or replace a file with specified content and file mode.

**Parameters**

- **destination** (`PurePath`) – Path to file.
- **content** (`BytesIO`) – Contents of file.
- **file\_mode** (`str`) – File mode string (e.g. `'0644'`).
- **group** (`str`) – File owner group.
- **user** (`str`) – File owner user.

**Return type** `None`

**temporarily\_pull\_file(\*, source, missing\_ok=False)**

Copy a file from the environment to a temporary file in the host.

This is mainly a layer above `pull_file` that pulls the file into a temporary path which is cleaned later.



Works as a context manager, provides the file path in the host as target.

#### Parameters

- **source** (Path) – Environment file to copy.
- **missing\_ok** (bool) – Do not raise an error if the file does not exist in the environment; in this case the target will be None.

#### Raises

- **FileNotFoundError** – If source file or destination's parent directory does not exist (and *missing\_ok* is False).
- **ProviderError** – On error copying file content.

**Return type** Generator[Optional[Path], None, None]

## 1.2 LXD Executor

```
class craft_providers.lxd.LXDInstance(*, name, default_command_environment=None, project='default',
                                     remote='local', lxc=None)
```

Bases: [craft\\_providers.executor.Executor](#)

LXD Instance Lifecycle.

#### Parameters

- **name** (str) –
- **default\_command\_environment** (Optional[Dict[str, Optional[str]]]) –
- **project** (str) –
- **remote** (str) –
- **lxc** (Optional[LXC]) –

**delete**(*force=True*)

Delete instance.

**Parameters** **force** (bool) – Delete even if running.

**Raises** [LXDError](#) – On unexpected error.

**Return type** None

**execute\_popen**(*command, \*, cwd=None, env=None, \*\*kwargs*)

Execute a command in instance, using subprocess.Popen().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via *env* parameter.

#### Parameters

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments to pass.
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**execute\_run**(*command*, \*, *cwd=None*, *env=None*, *\*\*kwargs*)

Execute a command using `subprocess.run()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to `subprocess.run()`.
- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** `subprocess.CalledProcessError` – if command fails and `check` is True.

**exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**Raises** `LXDError` – On unexpected error.

**is\_mounted**(\*, *host\_source*, *target*)

Check if path is mounted at target.

**Parameters**

- **host\_source** (Path) – Host path to check.
- **target** (PurePath) – Instance path to check.

**Return type** bool

**Returns** True if `host_source` is mounted at `target`.

**Raises** `LXDError` – On unexpected error.

**is\_running()**

Check if instance is running.

**Return type** bool

**Returns** True if instance is running.

**Raises** `LXDError` – On unexpected error.

**launch**(\*, *image*, *image\_remote*, *map\_user\_uid=False*, *ephemeral=False*, *uid=None*)

Launch instance.

**Parameters**

- **image** (str) – Image name to launch.
- **image\_remote** (str) – Image remote name.
- **map\_user\_id** – Whether id mapping should be used.
- **uid** (Optional[int]) – If `map_user_id` is True, the host user ID to map to instance root.
- **ephemeral** (bool) – Flag to enable ephemeral instance.

- **map\_user\_uid** (bool) –

Raises **LXDError** – On unexpected error.

**Return type** None

**mount**(\**, host\_source, target, device\_name=None*)

Mount host source directory to target mount point.

Checks first to see if already mounted. If no device name is given, it will be generated with the format “disk-*{target.as\_posix()}*”.

**Parameters**

- **host\_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.
- **device\_name** (Optional[str]) – Name for disk device.

Raises **LXDError** – On unexpected error.

**Return type** None

**pull\_file**(\**, source, destination*)

Copy a file from the environment to host.

**Parameters**

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (*destination.parent*) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **LXDError** – On unexpected error copying file.

**Return type** None

**push\_file**(\**, source, destination*)

Copy a file from the host into the environment.

**Parameters**

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (*destination.parent*) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **LXDError** – On unexpected error copying file.

**Return type** None

**push\_file\_io**(\**, destination, content, file\_mode, group='root', user='root'*)

Create or replace file with content and file mode.

**Parameters**

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.

- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File group owner/id.
- **user** (str) – File user owner/id.

Raises ***LXDError*** – On unexpected error.

**Return type** None

**start()**

Start instance.

Raises ***LXDError*** – on unexpected error.

**Return type** None

**stop()**

Stop instance.

Raises ***LXDError*** – on unexpected error.

**Return type** None

**supports\_mount()**

Check if instance supports mounting from host.

**Return type** bool

**Returns** True if mount is supported.

**unmount(target)**

Unmount mount target shared with host.

**Parameters** **target** (Path) – Target shared with host to unmount.

Raises ***LXDError*** – On failure to unmount target.

**Return type** None

**unmount\_all()**

Unmount all mounts shared with host.

Raises ***LXDError*** – On failure to unmount target.

**Return type** None

## 1.3 Multipass Executor

**class** craft\_providers.multipass.**MultipassInstance**(\*, name, multipass=None)

Bases: ***craft\_providers.executor.Executor***

Multipass Instance Lifecycle.

**Parameters**

- **name** (str) – Name of multipass instance.
- **multipass** (Optional[***Multipass***]) –

**delete()**

Delete instance and purge.

**Return type** None

**execute\_popen**(*command*, \*, *cwd=None*, *env=None*, *\*\*kwargs*)

Execute process in instance using subprocess.Popen().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via env parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments for subprocess.Popen().
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**execute\_run**(*command*, \*, *cwd=None*, *env=None*, *\*\*kwargs*)

Execute command using subprocess.run().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via env parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to subprocess.run().
- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** **subprocess.CalledProcessError** – if command fails and check is True.

**exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**Raises** **MultipassError** – On unexpected failure.

**is\_mounted**(\*, *host\_source*, *target*)

Check if path is mounted at target.

**Parameters**

- **host\_source** (Path) – Host path to check.
- **target** (PurePath) – Instance path to check.

**Return type** bool

**Returns** True if host\_source is mounted at target.

**Raises** **MultipassError** – On unexpected failure.

**is\_running()**

Check if instance is running.

**Return type** bool

**Returns** True if instance is running.

**Raises** *MultipassError* – On unexpected failure.

**launch**(\*, *image*, *cpus*=2, *disk\_gb*=256, *mem\_gb*=2)

Launch instance.

**Parameters**

- **image** (str) – Name of image to create the instance with.
- **instance\_cpus** – Number of CPUs.
- **instance\_disk\_gb** – Disk allocation in gigabytes.
- **instance\_mem\_gb** – Memory allocation in gigabytes.
- **instance\_name** – Name of instance to use/create.
- **instance\_stop\_time\_mins** – Stop time delay in minutes.
- **cpus** (int) –
- **disk\_gb** (int) –
- **mem\_gb** (int) –

**Raises** *MultipassError* – On unexpected failure.

**Return type** None

**mount**(\*, *host\_source*, *target*)

Mount host *host\_source* directory to target mount point.

Checks first to see if already mounted.

**Parameters**

- **host\_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

**Raises** *MultipassError* – On unexpected failure.

**Return type** None

**pull\_file**(\*, *source*, *destination*)

Copy a file from the environment to host.

**Parameters**

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (*destination.parent*) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- *MultipassError* – On unexpected error copying file.

**Return type** None

**push\_file**(\*, *source*, *destination*)

Copy a file from the host into the environment.

**Parameters**

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (destination.parent) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **MultipassError** – On unexpected error copying file.

**Return type** None

**push\_file\_io**(\*, destination, content, file\_mode, group='root', user='root')

Create or replace file with content and file mode.

Multipass transfers data as “ubuntu” user, forcing us to first copy a file to a temporary location before moving to a (possibly) root-owned location and with appropriate permissions.

**Parameters**

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File group owner/id.
- **user** (str) – File user owner/id.

**Return type** None

**start**()

Start instance.

**Raises** **MultipassError** – On unexpected failure.

**Return type** None

**stop**(\*, delay\_mins=0)

Stop instance.

**Parameters** **delay\_mins** (int) – Delay shutdown for specified minutes.

**Raises** **MultipassError** – On unexpected failure.

**Return type** None

**unmount**(target)

Unmount mount target shared with host.

**Parameters** **target** (Path) – Target shared with host to unmount.

**Raises** **MultipassError** – On failure to unmount target.

**Return type** None

**unmount\_all**()

Unmount all mounts shared with host.

**Raises** **MultipassError** – On failure to unmount target.

**Return type** None





## 2.1 Abstract Base

**class** `craft_providers.Base`

Bases: `abc.ABC`

Interface for providers to configure instantiated environments.

Defines how to setup/configure an environment that has been instantiated by a provider and prepare it for some operation, e.g. execute build. It must account for:

- (1) the OS type and version.
- (2) the provided image that was launched, e.g. bootstrapping a minimal image versus a more fully featured one.
- (3) any dependencies that are required for the operation to complete, e.g. installed applications, networking configuration, etc. This includes any environment configuration that the application will assume is available.

**Variables** `compatibility_tag` – Tag/Version for variant of build configuration and setup. Any change to this version would indicate that prior [versioned] instances are incompatible and must be cleaned. As such, any new value should be unique to old values (e.g. incrementing). It is suggested to extend this tag, not overwrite it, e.g.: `compatibility_tag = f'{appname}-{Base.compatibility_tag}.{appversion}'` to ensure base compatibility levels are maintained.

**abstract** `get_command_environment()`

Get command environment to use when executing commands.

**Return type** `Dict[str, Optional[str]]`

**Returns** Dictionary of environment, allowing None as a value to indicate that a value should be unset.

**abstract** `setup(*, executor, retry_wait=0.25, timeout=None)`

Prepare base instance for use by the application.

Wait for environment to become ready and configure it. At completion of setup, the executor environment should have networking up and have all of the installed dependencies required for subsequent use by the application.

Setup should not be called more than once in a given instance to refresh/update the environment, use *warmup* for that.

If timeout is specified, abort operation if time has been exceeded.

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).

- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- **BaseCompatibilityError** – if instance is incompatible.
- **BaseConfigurationError** – on other unexpected error.

**Return type** None

**abstract wait\_until\_ready**(\* , executor, retry\_wait=0.25, timeout=None)

Wait until base instance is ready.

Ensure minimum-required boot services are running. This would be used when starting an environment's container/VM after already [recently] running setup(), e.g. rebooting the instance. Allows the environment to be used without the cost incurred by re-executing the steps unnecessarily.

If timeout is specified, abort operation if time has been exceeded.

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- **BaseCompatibilityError** – if instance is incompatible.
- **BaseConfigurationError** – on other unexpected error.

**Return type** None

**abstract warmup**(\* , executor, retry\_wait=0.25, timeout=None)

Prepare a previously created and setup instance for use by the application.

Ensure the instance is still valid and wait for environment to become ready.

If timeout is specified, abort operation if time has been exceeded.

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- **BaseCompatibilityError** – if instance is incompatible.
- **BaseConfigurationError** – on other unexpected error.

**Return type** None

## 2.2 Buildd Base

**class** `craft_providers.bases.BuilddBase`(\*, *alias*, *compatibility\_tag*=None, *environment*=None, *hostname*='craft-buildd-instance', *snaps*=None, *packages*=None)

Bases: `craft_providers.base.Base`

Support for Ubuntu minimal buildd images.

### Variables

- **compatibility\_tag** – Tag/Version for variant of build configuration and setup. Any change to this version would indicate that prior [versioned] instances are incompatible and must be cleaned. As such, any new value should be unique to old values (e.g. incrementing). It is suggested to extend this tag, not overwrite it, e.g.: `compatibility_tag = f'{appname}-{BuildBase.compatibility_tag}.{appversion}'` to ensure base compatibility levels are maintained.
- **instance\_config\_path** – Path to persistent environment configuration used for compatibility checks (or other data). Set to `/etc/craft-instance.conf`, but may be overridden for application-specific reasons.
- **instance\_config\_class** – Class defining instance configuration. May be overridden with an application-specific subclass of `InstanceConfiguration` to enable application-specific extensions.

### Parameters

- **alias** (*BuilddBaseAlias*) – Base alias / version.
- **environment** (Optional[Dict[str, Optional[str]]]) – Environment to set in `/etc/environment`.
- **hostname** (str) – Hostname to configure.
- **snaps** (Optional[List[*Snap*]]) – Optional list of snaps to install on the base image.
- **packages** (Optional[List[str]]) – Optional list of system packages to install on the base image.
- **compatibility\_tag** (Optional[str]) –

**get\_command\_environment()**

Get command environment to use when executing commands.

**Return type** Dict[str, Optional[str]]

**Returns** Dictionary of environment, allowing None as a value to indicate that a value should be unset.

**instance\_config\_class**

alias of `craft_providers.bases.instance_config.InstanceConfiguration`

**setup**(\*, *executor*, *retry\_wait*=0.25, *timeout*=None)

Prepare base instance for use by the application.

Wait for environment to become ready and configure it. At completion of setup, the executor environment should have networking up and have all of the installed dependencies required for subsequent use by the application.

Setup may be called more than once in a given instance to refresh/update the environment.

If timeout is specified, abort operation if time has been exceeded.

**Guarantees provided by this setup:**

- configured /etc/environment
- configured hostname
- networking available (IP & DNS resolution)
- apt cache up-to-date
- snapd configured and ready
- system services are started and ready

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

#### Raises

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

**Return type** None

**wait\_until\_ready**(\* , executor, retry\_wait=0.25, timeout=None)

Wait until base instance is ready.

Ensure minimum-required boot services are running. This would be used when starting an environment's container/VM after already [recently] running setup(), e.g. rebooting the instance. Allows the environment to be used without the cost incurred by re-executing the steps unnecessarily.

If timeout is specified, abort operation if time has been exceeded.

Guarantees provided by this wait:

- networking available (IP & DNS resolution)
- system services are started and ready

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises** *ProviderError* – on timeout or unexpected error.

**Return type** None

**warmup**(\* , executor, retry\_wait=0.25, timeout=None)

Prepare a previously created and setup instance for use by the application.

Ensure the instance is still valid and wait for environment to become ready.

**Guarantees provided by this wait:**

- OS and instance config are compatible
- networking available (IP & DNS resolution)
- system services are started and ready

If timeout is specified, abort operation if time has been exceeded.

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

**Return type** None



## CRAFT\_PROVIDERS PACKAGE

### 3.1 Subpackages

#### 3.1.1 `craft_providers.actions` package

##### Submodules

##### `craft_providers.actions.snap_installer` module

Helpers for snap commands.

**exception** `craft_providers.actions.snap_installer.SnapInstallationError`(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: `craft_providers.errors.ProviderError`

Unexpected error during snap installation.

##### Parameters

- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

`craft_providers.actions.snap_installer.inject_from_host(*, executor, snap_name, classic)`

Inject snap from host snap.

**Raises** `SnapInstallationError` – on unexpected error.

##### Parameters

- **executor** (`Executor`) –
- **snap\_name** (str) –
- **classic** (bool) –

**Return type** None

`craft_providers.actions.snap_installer.install_from_store(*, executor, snap_name, channel, classic)`

Install snap from store into target.

Perform installation using method which prevents refreshing.

### Parameters

- **executor** (*Executor*) – Executor for target.
- **snap\_name** (str) – Name of snap to install.
- **channel** (str) – Channel to install from.
- **classic** (bool) – Install in classic mode.

**Raises** *SnapInstallationError* – on unexpected error.

**Return type** None

## Module contents

Executor helpers package.

### 3.1.2 craft\_providers.bases package

#### Submodules

#### craft\_providers.bases.buildd module

Buildd image(s).

```
class craft_providers.bases.buildd.BuilddBase(*, alias, compatibility_tag=None, environment=None,
                                             hostname='craft-buildd-instance', snaps=None,
                                             packages=None)
```

Bases: *craft\_providers.base.Base*

Support for Ubuntu minimal buildd images.

### Variables

- **compatibility\_tag** – Tag/Version for variant of build configuration and setup. Any change to this version would indicate that prior [versioned] instances are incompatible and must be cleaned. As such, any new value should be unique to old values (e.g. incrementing). It is suggested to extend this tag, not overwrite it, e.g.: `compatibility_tag = f'{appname}-{BuildBase.compatibility_tag}.{appversion}'` to ensure base compatibility levels are maintained.
- **instance\_config\_path** – Path to persistent environment configuration used for compatibility checks (or other data). Set to `/etc/craft-instance.conf`, but may be overridden for application-specific reasons.
- **instance\_config\_class** – Class defining instance configuration. May be overridden with an application-specific subclass of `InstanceConfiguration` to enable application-specific extensions.

### Parameters

- **alias** (*BuilddBaseAlias*) – Base alias / version.
- **environment** (Optional[Dict[str, Optional[str]]]) – Environment to set in `/etc/environment`.
- **hostname** (str) – Hostname to configure.



- **snaps** (Optional[List[*Snap*]]) – Optional list of snaps to install on the base image.
- **packages** (Optional[List[str]]) – Optional list of system packages to install on the base image.
- **compatibility\_tag** (Optional[str]) –

**alias:** *craft\_providers.bases.buildd.BuilddBaseAlias*

**compatibility\_tag:** str = 'buildd-base-v0'

**get\_command\_environment()**

Get command environment to use when executing commands.

**Return type** Dict[str, Optional[str]]

**Returns** Dictionary of environment, allowing None as a value to indicate that a value should be unset.

**instance\_config\_class**

alias of *craft\_providers.bases.instance\_config.InstanceConfiguration*

**instance\_config\_path:** pathlib.Path = PosixPath('/etc/craft-instance.conf')

**setup**(\* , executor, retry\_wait=0.25, timeout=None)

Prepare base instance for use by the application.

Wait for environment to become ready and configure it. At completion of setup, the executor environment should have networking up and have all of the installed dependencies required for subsequent use by the application.

Setup may be called more than once in a given instance to refresh/update the environment.

If timeout is specified, abort operation if time has been exceeded.

#### Guarantees provided by this setup:

- configured /etc/environment
- configured hostname
- networking available (IP & DNS resolution)
- apt cache up-to-date
- snapd configured and ready
- system services are started and ready

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

#### Raises

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

**Return type** None

**wait\_until\_ready**(\*, *executor*, *retry\_wait*=0.25, *timeout*=None)

Wait until base instance is ready.

Ensure minimum-required boot services are running. This would be used when starting an environment's container/VM after already [recently] running `setup()`, e.g. rebooting the instance. Allows the environment to be used without the cost incurred by re-executing the steps unnecessarily.

If timeout is specified, abort operation if time has been exceeded.

Guarantees provided by this wait:

- networking available (IP & DNS resolution)
- system services are started and ready

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

Raises *ProviderError* – on timeout or unexpected error.

Return type None

**warmup**(\*, *executor*, *retry\_wait*=0.25, *timeout*=None)

Prepare a previously created and setup instance for use by the application.

Ensure the instance is still valid and wait for environment to become ready.

**Guarantees provided by this wait:**

- OS and instance config are compatible
- networking available (IP & DNS resolution)
- system services are started and ready

If timeout is specified, abort operation if time has been exceeded.

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

Raises

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

Return type None

**class** `craft_providers.bases.buildd.BuilddBaseAlias`(*value*)

Bases: `enum.Enum`

Mappings for supported buildd images.

**BIONIC** = '18.04'

**FOCAL** = '20.04'

**JAMMY** = '22.04'

**XENIAL** = '16.04'

**class** craft\_providers.bases.buildd.**Snap**(\*\*data)

Bases: pydantic.main.BaseModel

Details of snap to install in the base.

#### Parameters

- **name** – name of snap
- **channel** – snap store channel to install from (default is stable) If channel is *None*, then the snap is injected from the host instead of being installed from the store.
- **classic** – true if snap is a classic snap (default is false)
- **data** (Any) –

**channel:** Optional[str]

**classic:** bool

**name:** str

**classmethod** **validate\_channel**(channel)

Validate that channel is not an empty string.

**Raises** *BaseConfigurationError* – if channel is empty

**craft\_providers.bases.buildd.default\_command\_environment()**

Provide default command environment dictionary.

The minimum environment for the buildd image to be configured and function properly. This contains the default environment found in Ubuntu's /etc/environment, replaced with the “secure\_path” defaults used by sudo for instantiating PATH. In practice it really just means the PATH set by sudo.

Default /etc/environment found in supported Ubuntu versions: PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:

/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

Default /etc/sudoers secure\_path found in supported Ubuntu versions:  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin

**Return type** Dict[str, Optional[str]]

**Returns** Dictionary of environment key/values.

## craft\_providers.bases.errors module

Base errors.

**exception** craft\_providers.bases.errors.**BaseCompatibilityError**(reason, \*, details=None)

Bases: *craft\_providers.errors.ProviderError*

Base configuration compatibility error.

#### Parameters

- **reason** (str) – Reason for incompatibility.
- **details** (Optional[str]) –

**brief:** str

**exception** `craft_providers.bases.errors.BaseConfigurationError`(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: `craft_providers.errors.ProviderError`

Error configuring the base.

**Parameters**

- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

## `craft_providers.bases.instance_config` module

Persistent instance config / datastore resident in provided environment.

**class** `craft_providers.bases.instance_config.InstanceConfiguration`(*\*\*data*)

Bases: `pydantic.main.BaseModel`

Instance configuration datastore.

**Parameters**

- **compatibility\_tag** – Compatibility tag for instance.
- **snaps** – dictionary of snaps and their revisions, e.g. snaps:  
     **snapcraft:** revision: “x100”  
     **charmcrafter:** revision: 834
- **data** (Any) –

**compatibility\_tag:** Optional[str]

**classmethod** `load(executor, config_path=PosixPath('/etc/craft-instance.conf'))`

Load an instance config file from an environment.

**Parameters**

- **executor** (`Executor`) – Executor for instance.
- **config\_path** (Path) – Path to configuration file. Default is `/etc/craft-instance.conf`.

**Return type** Optional[`InstanceConfiguration`]

**Returns** The InstanceConfiguration object or None, if the config does not exist or is empty.

**Raises** `BaseConfigurationError` – If the file cannot be loaded from the environment.

**marshal()**

Create a dictionary containing the InstanceConfiguration data.

**Return type** Dict[str, Any]

**Returns** The newly created dictionary.

**save(executor, config\_path=PosixPath('/etc/craft-instance.conf'))**

Save an instance config file to an environment.

**Parameters**

- **executor** (*Executor*) – Executor for instance.
- **config\_path** (Path) – Path to configuration file. Default is */etc/craft-instance.conf*.

**Return type** None

**snaps:** Optional[Dict[str, Dict[str, Any]]]

**classmethod** *unmarshal*(data)

Create and populate a new *InstanceConfig* object from dictionary data.

The unmarshal method validates the data in the dictionary and populates the corresponding fields in the *InstanceConfig* object.

**Parameters** *data* (Dict[str, Any]) – The dictionary data to unmarshal.

**Return type** *InstanceConfiguration*

**Returns** The newly created *InstanceConfiguration* object.

**Raises** *BaseConfigurationError* – If validation fails.

**classmethod** *update*(executor, data, config\_path=PosixPath('/etc/craft-instance.conf'))

Update an instance config file in an environment.

New values are added and existing values are updated. No data are removed. If there is no existing config to update, then a new config is created.

**Parameters**

- **executor** (*Executor*) – Executor for instance.
- **data** (Dict[str, Any]) – The dictionary to update instance with.
- **config\_path** (Path) –

**Return type** *InstanceConfiguration*

**Returns** The updated *InstanceConfiguration* object.

*craft\_providers.bases.instance\_config.update\_nested\_dictionaries*(config\_data, new\_data)

Recursively update a dictionary containing nested dictionaries.

New values are added and existing values are updated. No data are removed.

**Parameters**

- **config\_data** (Dict[str, Any]) – dictionary of config data to update.
- **new\_data** (Dict[str, Any]) – data to update *config\_data* with.

**Return type** Dict[str, Any]

## Module contents

Collection of bases used to configure build environments.

**exception** *craft\_providers.bases.BaseCompatibilityError*(reason, \*, details=None)

Bases: *craft\_providers.errors.ProviderError*

Base configuration compatibility error.

**Parameters**

- **reason** (str) – Reason for incompatibility.
- **details** (Optional[str]) –

**brief:** str

**exception** craft\_providers.bases.**BaseConfigurationError**(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: [craft\\_providers.errors.ProviderError](#)

Error configuring the base.

#### Parameters

- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

**class** craft\_providers.bases.**BuildBase**(*\*, alias, compatibility\_tag=None, environment=None, hostname='craft-build-base-instance', snaps=None, packages=None*)

Bases: [craft\\_providers.base.Base](#)

Support for Ubuntu minimal build images.

#### Variables

- **compatibility\_tag** – Tag/Version for variant of build configuration and setup. Any change to this version would indicate that prior [versioned] instances are incompatible and must be cleaned. As such, any new value should be unique to old values (e.g. incrementing). It is suggested to extend this tag, not overwrite it, e.g.: `compatibility_tag = f'{appname}-{BuildBase.compatibility_tag}.{appversion}'` to ensure base compatibility levels are maintained.
- **instance\_config\_path** – Path to persistent environment configuration used for compatibility checks (or other data). Set to `/etc/craft-instance.conf`, but may be overridden for application-specific reasons.
- **instance\_config\_class** – Class defining instance configuration. May be overridden with an application-specific subclass of `InstanceConfiguration` to enable application-specific extensions.

#### Parameters

- **alias** ([BuildBaseAlias](#)) – Base alias / version.
- **environment** (Optional[Dict[str, Optional[str]]]) – Environment to set in `/etc/environment`.
- **hostname** (str) – Hostname to configure.
- **snaps** (Optional[List[[Snap](#)]]) – Optional list of snaps to install on the base image.
- **packages** (Optional[List[str]]) – Optional list of system packages to install on the base image.
- **compatibility\_tag** (Optional[str]) –

**compatibility\_tag:** str = 'build-base-v0'

**get\_command\_environment**()

Get command environment to use when executing commands.

**Return type** Dict[str, Optional[str]]

**Returns** Dictionary of environment, allowing None as a value to indicate that a value should be unset.

**instance\_config\_class**alias of `craft_providers.bases.instance_config.InstanceConfiguration`**instance\_config\_path:** `pathlib.Path = PosixPath('/etc/craft-instance.conf')`**setup**(\*, *executor*, *retry\_wait*=0.25, *timeout*=None)

Prepare base instance for use by the application.

Wait for environment to become ready and configure it. At completion of setup, the executor environment should have networking up and have all of the installed dependencies required for subsequent use by the application.

Setup may be called more than once in a given instance to refresh/update the environment.

If timeout is specified, abort operation if time has been exceeded.

**Guarantees provided by this setup:**

- configured /etc/environment
- configured hostname
- networking available (IP & DNS resolution)
- apt cache up-to-date
- snapd configured and ready
- system services are started and ready

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

**Return type** None**wait\_until\_ready**(\*, *executor*, *retry\_wait*=0.25, *timeout*=None)

Wait until base instance is ready.

Ensure minimum-required boot services are running. This would be used when starting an environment's container/VM after already [recently] running setup(), e.g. rebooting the instance. Allows the environment to be used without the cost incurred by re-executing the steps unnecessarily.

If timeout is specified, abort operation if time has been exceeded.

Guarantees provided by this wait:

- networking available (IP & DNS resolution)
- system services are started and ready

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).

- **timeout** (Optional[float]) – Timeout in seconds.

Raises *ProviderError* – on timeout or unexpected error.

Return type None

**warmup**(\**, executor, retry\_wait=0.25, timeout=None*)

Prepare a previously created and setup instance for use by the application.

Ensure the instance is still valid and wait for environment to become ready.

**Guarantees provided by this wait:**

- OS and instance config are compatible
- networking available (IP & DNS resolution)
- system services are started and ready

If timeout is specified, abort operation if time has been exceeded.

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

Return type None

**class** *craft\_providers.bases.BuildddBaseAlias*(*value*)

Bases: *enum.Enum*

Mappings for supported builddd images.

**BIONIC** = '18.04'

**FOCAL** = '20.04'

**JAMMY** = '22.04'

**XENIAL** = '16.04'

### 3.1.3 *craft\_providers.lxd* package

#### Submodules

#### *craft\_providers.lxd.errors* module

LXD Errors.

**exception** *craft\_providers.lxd.errors.LXDError*(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: *craft\_providers.errors.ProviderError*

Unexpected LXD error.

**Parameters**



- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

**exception** `craft_providers.lxd.errors.LXDInstallationError`(*reason*, \*, *details=None*)

Bases: `craft_providers.lxd.errors.LXDError`

LXD Installation Error.

**Parameters**

- **reason** (str) – Reason for install failure.
- **details** (Optional[str]) – Optional details to include.

**brief:** str

## craft\_providers.lxd.installer module

LXD Provider.

`craft_providers.lxd.installer.ensure_lxd_is_ready`(\*, *remote='local'*,  
*lxc=<craft\_providers.lxd.lxc.LXC object>*,  
*lxd=<craft\_providers.lxd.lxd.LXD object>*)

Ensure LXD is ready for use.

**Raises** `LXDError` – on error.

**Parameters**

- **remote** (str) –
- **lxc** (`LXC`) –
- **lxd** (`LXD`) –

**Return type** None

`craft_providers.lxd.installer.install`(*sudo=True*)

Install LXD.

Install application, using sudo if specified.

**Return type** str

**Returns** LXD version.

**Raises**

- `LXDInstallationError` – on installation error.
- `LXDError` – on unexpected error.

**Parameters** *sudo* (bool) –

`craft_providers.lxd.installer.is_initialized`(\*, *remote*, *lxc*)

Verify that LXD has been initialized and configuration looks valid.

If LXD has been installed but the user has not initialized it (lxd init), the default profile won't have devices configured. Trying to launch an instance or create a project using this profile will result in failures.

**Return type** bool

**Returns** True if initialized, else False.

**Parameters**

- **remote** (str) –
- **lxc** (*LXC*) –

`craft_providers.lxd.installer.is_installed()`

Check if LXD is installed (and found on PATH).

**Return type** bool

**Returns** True if lxd is installed.

`craft_providers.lxd.installer.is_user_permitted()`

Check if user has permissions to connect to LXD.

**Return type** bool

**Returns** True if user has correct permissions.

### **craft\_providers.lxd.launcher module**

LXD Instance Provider.

`craft_providers.lxd.launcher.launch(name, *, base_configuration, image_name, image_remote, auto_clean=False, auto_create_project=False, ephemeral=False, map_user_uid=False, uid=None, use_snapshots=False, project='default', remote='local', lxc=<craft_providers.lxd.lxc.LXC object>)`

Create, start, and configure instance.

If `auto_clean` is enabled, automatically delete an existing instance that is deemed to be incompatible, rebuilding it with the specified environment.

**Parameters**

- **name** (str) – Name of instance.
- **base\_configuration** (*Base*) – Base configuration to apply to instance.
- **image\_name** (str) – LXD image to use, e.g. “20.04”.
- **image\_remote** (str) – LXD image to use, e.g. “ubuntu”.
- **auto\_clean** (bool) – Automatically clean instance, if incompatible.
- **auto\_create\_project** (bool) – Automatically create LXD project, if needed.
- **ephemeral** (bool) – Create ephemeral instance.
- **map\_user\_uid** (bool) – Map host uid/gid to instance’s root uid/gid.
- **uid** (Optional[int]) – The uid to be mapped, if `map_user_id` is enabled.
- **use\_snapshots** (bool) – Use LXD snapshots for bootstrapping images.
- **project** (str) – LXD project to create instance in.
- **remote** (str) – LXD remote to create instance on.
- **lxc** (*LXC*) – LXC client.

**Return type** *LXDInstance*

**Returns** LXD instance.

**Raises**

- *BaseConfigurationError* – on unexpected error configuration base.
- *LXDError* – on unexpected LXD error.

### **craft\_providers.lxd.lxc module**

LXC wrapper.

**class** `craft_providers.lxd.lxc.LXC(*, lxc_path=PosixPath('lxc'))`

Bases: `object`

Wrapper for lxc command-line interface.

**Parameters** `lxc_path` (`Path`) –

**config\_device\_add\_disk**(`*, instance_name, source, path, device, project='default', remote='local'`)

Mount host source directory to target mount point.

**Parameters**

- `instance_name` (`str`) – Name of instance.
- `source` (`Path`) – Host path.
- `path` (`PurePath`) – Mount target in instance.
- `device` (`str`) – Name of device.
- `project` (`str`) – Name of LXD project.
- `remote` (`str`) – Name of LXD remote.

**Raises** *LXDError* – on unexpected error.

**Return type** `None`

**config\_device\_remove**(`*, instance_name, device, project='default', remote='local'`)

Mount host source directory to target mount point.

**Parameters**

- `instance_name` (`str`) – Name of instance.
- `device` (`str`) – Name of device.
- `project` (`str`) – Name of LXD project.
- `remote` (`str`) – Name of LXD remote.

**Raises** *LXDError* – on unexpected error.

**Return type** `None`

**config\_device\_show**(`*, instance_name, project='default', remote='local'`)

Show full device configuration.

**Parameters**

- `instance_name` (`str`) – Name of instance.
- `project` (`str`) – Name of LXD project.
- `remote` (`str`) – Name of LXD remote.

Raises ***LXDError*** – on unexpected error.

**Return type** Dict[str, Any]

**config\_set**(\* , instance\_name, key, value, project='default', remote='local')  
Set instance\_name configuration key.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **key** (str) – Config key name.
- **value** (str) – Config key value.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises ***LXDError*** – on unexpected error.

**Return type** None

**delete**(\* , instance\_name, force=False, project='default', remote='local')  
Delete instance.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **force** (bool) – Force deletion if running.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises ***LXDError*** – on unexpected error.

**Return type** None

**exec**(\* , command, instance\_name, cwd=None, mode=None, project='default', remote='local',  
runner=<function run>, \*\*kwargs)  
Execute command in instance\_name with specified runner.

**Parameters**

- **command** (List[str]) – Command to execute in the instance.
- **instance\_name** (str) – Name of instance to execute in.
- **cwd** (Optional[str]) – Optional current working directory for command.
- **mode** (Optional[str]) – Override terminal mode Valid options include: “auto”, “interactive”, “non-interactive”. lxd default is “auto”.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.
- **runner** (Callable) – Execution function to invoke, e.g. subprocess.run or Popen. First argument is finalized command with the attached kwargs.
- **kwargs** – Additional kwargs for runner.

**Returns** Runner’s instance.

**file\_pull**(\* , instance\_name, source, destination, create\_dirs=False, recursive=False, project='default',  
remote='local')  
Retrieve file from instance\_name.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **source** (PurePath) – Path in environment to pull.
- **destination** (Path) – Path in host to write to.
- **create\_dirs** (bool) – Create any directories necessary.
- **recursive** (bool) – Recursively transfer files.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**file\_push**(\* , instance\_name, source, destination, create\_dirs=False, recursive=False, gid=None, uid=None, mode=None, project='default', remote='local')

Create file with content and file mode.

**Parameters**

- **instance\_name** (str) – Name of instance to push file to.
- **source** (Path) – Path in host to push.
- **destination** (PurePath) – Path in environment to write to.
- **create\_dirs** (bool) – Create any directories necessary.
- **recursive** (bool) – Recursively transfer files.
- **gid** (Optional[int]) – Optional gid to set on push (lxd's default is -1).
- **uid** (Optional[int]) – Optional uid to set on push (lxd's default is -1).
- **mode** (Optional[str]) – Optional file mode to set on file.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**has\_image**(image\_name, \*, project='default', remote='local')

Check if image with given alias name is present.

**Parameters**

- **image\_name** – Name of image alias.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Return type bool

**image\_copy**(\* , image, image\_remote, alias=None, project='default', remote='local')

Copy image.

**Parameters**

- **instance\_name** – Optional instance name.

- **alias** (Optional[str]) – New alias to add to image.
- **image** (str) – Image to copy.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.
- **image\_remote** (str) –

Raises **LXDError** – on unexpected error.

Return type None

**image\_delete**(\*, image, project='default', remote='local')

Delete image.

Parameters

- **image** (str) – Image to delete.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**image\_list**(\*, project='default', remote='local')

List images.

Parameters

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Return type List[Dict[str, Any]]

**info**(\*, instance\_name=None, project='default', remote='local')

Show instance or server information.

Parameters

- **instance\_name** (Optional[str]) – Optional instance name.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type Dict[str, Any]

**launch**(\*, instance\_name, image, image\_remote, config\_keys=None, ephemeral=False, project='default', remote='local')

Launch instance.

Parameters

- **instance\_name** (str) – Name of instance to launch.
- **image** (str) – Name of image to use.
- **image\_remote** (str) – Name of image's remote.
- **config\_keys** (Optional[Dict[str, str]]) – Configuration keys to set.
- **ephemeral** (bool) – Use ephemeral instance.

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

**Return type** None

**list**(\*, *project='default', remote='local'*)

List instances and their status.

**Parameters**

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Return type** List[Dict[str, Any]]

**Returns** List of containers and their info.

Raises **LXDError** – on unexpected error.

**list\_names**(\*, *project='default', remote='local'*)

List container names.

A helper to get a list of container names from list().

**Parameters**

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Return type** List[str]

**Returns** List of container names.

Raises **LXDError** – on unexpected error.

**profile\_edit**(\*, *profile, config, project='default', remote='local'*)

Set profile configuration.

**Parameters**

- **profile** (str) – Name of profile.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.
- **config** (Dict[str, Any]) –

Raises **LXDError** – on unexpected error.

**Return type** None

**profile\_show**(\*, *profile, project='default', remote='local'*)

Get profile configuration.

**Parameters**

- **profile** (str) – Name of profile.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

**Return type** Dict[str, Any]

**project\_create**(\* , project, remote='local')

Create project.

**Parameters**

- **project** (str) – Name of LXD project to create.
- **remote** (str) – Name of LXD remote to create project on.

**Raises** *LXDError* – on unexpected error.

**Return type** None

**project\_delete**(\* , project, remote='local')

Delete project, if it exists.

**Parameters**

- **project** (str) – Name of LXD project to delete.
- **remote** (str) – Name of LXD remote.

**Raises** *LXDError* – on unexpected error.

**Return type** None

**project\_list**(remote='local')

Get list of projects.

**Parameters** **remote** (str) – Name of LXD remote to query.

**Return type** List[str]

**Returns** List of project names.

**Raises** *LXDError* – on unexpected error.

**publish**(\* , instance\_name, alias=None, force=False, image\_remote='local', project='default', remote='local')

Publish image from instance.

**Parameters**

- **instance\_name** (str) – Name of instance to publish image from.
- **alias** (Optional[str]) – New alias to define at target.
- **force** (bool) – Force publishing of image, even if container is running.
- **image\_remote** (str) – Name of remote to publish image to.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote instance is found on.

**Raises** *LXDError* – on unexpected error.

**Return type** None

**remote\_add**(\* , remote, addr, protocol='simplestreams')

Add a public remote.

**Parameters**

- **remote** (str) – Name of remote to add.
- **addr** (str) – Address of remote.



- **protocol** (str) – Name of protocol (“simplestreams” or “lxd”).

Raises **LXDError** – on unexpected error.

Return type None

**remote\_list()**

Get list of remotes.

Return type Dict[str, Any]

Returns dictionary with remote name mapping to config.

**start**(\**, instance\_name, project='default', remote='local'*)

Start container.

Parameters

- **instance\_name** (str) – Name of instance to start.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**stop**(\**, instance\_name, force=False, timeout=- 1, project='default', remote='local'*)

Stop container.

Parameters

- **instance\_name** (str) – Name of instance to stop.
- **force** (bool) – Force instance to stop.
- **timeout** (int) – Timeout in seconds. -1 is no timeout.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**class** `craft_providers.lxd.lxc.StdinType`(*value*)

Bases: `enum.Enum`

Mappings for input stream to pass to stdin for lxc commands.

**INTERACTIVE** = -3

**NULL** = None

`craft_providers.lxd.lxc.load_yaml`(*data*)

Load yaml without additional resolvers.

LXD may return YAML that has datetimes that are not valid when parsed to `datetime.datetime()`. Instead just use the base loader and avoid resolving this type (and others).

**craft\_providers.lxd.lxd module**

LXD command-line interface helpers.

**class** `craft_providers.lxd.lxd.LXD(*, lxd_path=PosixPath('lxd'))`

Bases: `object`

Interface to *lxd* command-line.

**Parameters** `lxd_path` (`Path`) – Path to *lxd*.

**Variables** `minimum_required_version` – Minimum *lxd* version required for compatibility.

**init**(`*, auto=False, sudo=False`)

Initialize LXD.

Sudo is required if user is not in *lxd* group.

**Parameters**

- **auto** (`bool`) – Use default settings.
- **sudo** (`bool`) – Use *sudo* to invoke *init*.

**Return type** `None`

**is\_supported\_version**()

Check if LXD version is supported.

A helper to check if LXD meets minimum supported version for craft-providers (currently  $\geq 4.0$ ).

**Return type** `bool`

**Returns** True if installed version is supported.

**minimum\_required\_version** = `'4.0'`

**version**()

Query LXD version.

The version is of the format: `<major>.<minor>[.<micro>]`

Version examples: - 4.13 - 4.0.5 - 2.0.12

**Return type** `str`

**Returns** Version string.

**wait\_ready**(`*, sudo=False, timeout=None`)

Wait until LXD is ready.

Sudo is required if user is not in *lxd* group.

**Parameters**

- **sudo** (`bool`) – Use *sudo* to invoke *waitready*.
- **timeout** (`Optional[int]`) – Timeout in seconds.

**Return type** `None`

**craft\_providers.lxd.lxd\_instance module**

LXD Instance Executor.

**class** craft\_providers.lxd.lxd\_instance.**LXDInstance**(\*, name, default\_command\_environment=None, project='default', remote='local', lxc=None)

Bases: [craft\\_providers.executor.Executor](#)

LXD Instance Lifecycle.

**Parameters**

- **name** (str) –
- **default\_command\_environment** (Optional[Dict[str, Optional[str]]]) –
- **project** (str) –
- **remote** (str) –
- **lxc** (Optional[LXC]) –

**delete**(force=True)

Delete instance.

**Parameters** **force** (bool) – Delete even if running.

**Raises** [LXDError](#) – On unexpected error.

**Return type** None

**execute\_popen**(command, \*, cwd=None, env=None, \*\*kwargs)

Execute a command in instance, using subprocess.Popen().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via env parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments to pass.
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**execute\_run**(command, \*, cwd=None, env=None, \*\*kwargs)

Execute a command using subprocess.run().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via env parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to subprocess.run().
- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** `subprocess.CalledProcessError` – if command fails and check is True.

**exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**Raises** `LXDError` – On unexpected error.

**is\_mounted**(*\*, host\_source, target*)

Check if path is mounted at target.

**Parameters**

- **host\_source** (Path) – Host path to check.
- **target** (PurePath) – Instance path to check.

**Return type** bool

**Returns** True if host\_source is mounted at target.

**Raises** `LXDError` – On unexpected error.

**is\_running()**

Check if instance is running.

**Return type** bool

**Returns** True if instance is running.

**Raises** `LXDError` – On unexpected error.

**launch**(*\*, image, image\_remote, map\_user\_uid=False, ephemeral=False, uid=None*)

Launch instance.

**Parameters**

- **image** (str) – Image name to launch.
- **image\_remote** (str) – Image remote name.
- **map\_user\_id** – Whether id mapping should be used.
- **uid** (Optional[int]) – If map\_user\_id is True, the host user ID to map to instance root.
- **ephemeral** (bool) – Flag to enable ephemeral instance.
- **map\_user\_uid** (bool) –

**Raises** `LXDError` – On unexpected error.

**Return type** None

**mount**(*\*, host\_source, target, device\_name=None*)

Mount host source directory to target mount point.

Checks first to see if already mounted. If no device name is given, it will be generated with the format “disk-{target.as\_posix()}”.

**Parameters**

- **host\_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

- **device\_name** (Optional[str]) – Name for disk device.

Raises **LXDError** – On unexpected error.

Return type None

**pull\_file**(\*, *source*, *destination*)

Copy a file from the environment to host.

Parameters

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (destination.parent) must exist.

Raises

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **LXDError** – On unexpected error copying file.

Return type None

**push\_file**(\*, *source*, *destination*)

Copy a file from the host into the environment.

Parameters

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (destination.parent) must exist.

Raises

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **LXDError** – On unexpected error copying file.

Return type None

**push\_file\_io**(\*, *destination*, *content*, *file\_mode*, *group*='root', *user*='root')

Create or replace file with content and file mode.

Parameters

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File group owner/id.
- **user** (str) – File user owner/id.

Raises **LXDError** – On unexpected error.

Return type None

**start**()

Start instance.

Raises **LXDError** – on unexpected error.

Return type None

### **stop()**

Stop instance.

**Raises** *LXDError* – on unexpected error.

**Return type** None

### **supports\_mount()**

Check if instance supports mounting from host.

**Return type** bool

**Returns** True if mount is supported.

### **unmount(target)**

Unmount mount target shared with host.

**Parameters** *target* (Path) – Target shared with host to unmount.

**Raises** *LXDError* – On failure to unmount target.

**Return type** None

### **unmount\_all()**

Unmount all mounts shared with host.

**Raises** *LXDError* – On failure to unmount target.

**Return type** None

## **craft\_providers.lxd.project module**

Project helper utilities.

`craft_providers.lxd.project.create_with_default_profile(*, lxc, project, profile='default',  
profile_project='default', remote='local')`

Create a project with a valid default profile.

LXD does not set a valid profile on newly created projects. This will create a project and set the profile to match the specified profile, typically the default.

### **Parameters**

- **project** (str) – Name of project to create.
- **remote** (str) – Name of remote.
- **profile\_name** – Name of profile to copy.
- **lxc** (*LXC*) –
- **profile** (str) –
- **profile\_project** (str) –

**Raises** *LXDError* – on unexpected error.

**Return type** None

`craft_providers.lxd.project.purge(*, lxc, project, remote='local')`

Purge a project including its instances and images.

The lxc command does not provide a straight-forward option to purge a project. This helper will purge anything related to a specified one.

### **Parameters**

- **project** (str) – Name of project to delete.
- **remote** (str) – Name of remote.
- **lxc** (LXC) –

Raises *LXDError* – on unexpected error.

Return type None

## craft\_providers.lxd.remotes module

Remote helper utilities.

`craft_providers.lxd.remotes.configure_buldd_image_remote(lxc=<craft_providers.lxd.lxc.LXC object>)`

Configure buldd remote, adding remote as required.

**Parameters** **lxc** (LXC) – LXC client.

**Return type** str

**Returns** Name of remote to pass to launcher.

## Module contents

LXD environment provider.

`class craft_providers.lxd.LXC(*, lxc_path=PosixPath('lxc'))`

Bases: object

Wrapper for lxc command-line interface.

**Parameters** **lxc\_path** (Path) –

`config_device_add_disk(*, instance_name, source, path, device, project='default', remote='local')`

Mount host source directory to target mount point.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **source** (Path) – Host path.
- **path** (PurePath) – Mount target in instance.
- **device** (str) – Name of device.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises *LXDError* – on unexpected error.

Return type None

`config_device_remove(*, instance_name, device, project='default', remote='local')`

Mount host source directory to target mount point.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **device** (str) – Name of device.

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**config\_device\_show**(\* , instance\_name, project='default', remote='local')

Show full device configuration.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type Dict[str, Any]

**config\_set**(\* , instance\_name, key, value, project='default', remote='local')

Set instance\_name configuration key.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **key** (str) – Config key name.
- **value** (str) – Config key value.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**delete**(\* , instance\_name, force=False, project='default', remote='local')

Delete instance.

**Parameters**

- **instance\_name** (str) – Name of instance.
- **force** (bool) – Force deletion if running.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**exec**(\* , command, instance\_name, cwd=None, mode=None, project='default', remote='local',  
runner=<function run>, \*\*kwargs)

Execute command in instance\_name with specified runner.

**Parameters**

- **command** (List[str]) – Command to execute in the instance.
- **instance\_name** (str) – Name of instance to execute in.
- **cwd** (Optional[str]) – Optional current working directory for command.



- **mode** (Optional[str]) – Override terminal mode Valid options include: “auto”, “interactive”, “non-interactive”. lxd default is “auto”.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.
- **runner** (Callable) – Execution function to invoke, e.g. subprocess.run or Popen. First argument is finalized command with the attached kwargs.
- **kwargs** – Additional kwargs for runner.

**Returns** Runner’s instance.

**file\_pull**(\* , instance\_name, source, destination, create\_dirs=False, recursive=False, project='default', remote='local')

Retrieve file from instance\_name.

#### Parameters

- **instance\_name** (str) – Name of instance.
- **source** (PurePath) – Path in environment to pull.
- **destination** (Path) – Path in host to write to.
- **create\_dirs** (bool) – Create any directories necessary.
- **recursive** (bool) – Recursively transfer files.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Raises** [LXDError](#) – on unexpected error.

**Return type** None

**file\_push**(\* , instance\_name, source, destination, create\_dirs=False, recursive=False, gid=None, uid=None, mode=None, project='default', remote='local')

Create file with content and file mode.

#### Parameters

- **instance\_name** (str) – Name of instance to push file to.
- **source** (Path) – Path in host to push.
- **destination** (PurePath) – Path in environment to write to.
- **create\_dirs** (bool) – Create any directories necessary.
- **recursive** (bool) – Recursively transfer files.
- **gid** (Optional[int]) – Optional gid to set on push (lxd’s default is -1).
- **uid** (Optional[int]) – Optional uid to set on push (lxd’s default is -1).
- **mode** (Optional[str]) – Optional file mode to set on file.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Raises** [LXDError](#) – on unexpected error.

**Return type** None

**has\_image**(*image\_name*, \*, *project*='default', *remote*='local')

Check if image with given alias name is present.

**Parameters**

- **image\_name** – Name of image alias.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Return type** bool

**image\_copy**(\*, *image*, *image\_remote*, *alias*=None, *project*='default', *remote*='local')

Copy image.

**Parameters**

- **instance\_name** – Optional instance name.
- **alias** (Optional[str]) – New alias to add to image.
- **image** (str) – Image to copy.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.
- **image\_remote** (str) –

**Raises** *LXDError* – on unexpected error.

**Return type** None

**image\_delete**(\*, *image*, *project*='default', *remote*='local')

Delete image.

**Parameters**

- **image** (str) – Image to delete.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Raises** *LXDError* – on unexpected error.

**Return type** None

**image\_list**(\*, *project*='default', *remote*='local')

List images.

**Parameters**

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Return type** List[Dict[str, Any]]

**info**(\*, *instance\_name*=None, *project*='default', *remote*='local')

Show instance or server information.

**Parameters**

- **instance\_name** (Optional[str]) – Optional instance name.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises ***LXDError*** – on unexpected error.

**Return type** Dict[str, Any]

**launch**(\*, instance\_name, image, image\_remote, config\_keys=None, ephemeral=False, project='default', remote='local')

Launch instance.

#### Parameters

- **instance\_name** (str) – Name of instance to launch.
- **image** (str) – Name of image to use.
- **image\_remote** (str) – Name of image's remote.
- **config\_keys** (Optional[Dict[str, str]]) – Configuration keys to set.
- **ephemeral** (bool) – Use ephemeral instance.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises ***LXDError*** – on unexpected error.

**Return type** None

**list**(\*, project='default', remote='local')

List instances and their status.

#### Parameters

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Return type** List[Dict[str, Any]]

**Returns** List of containers and their info.

Raises ***LXDError*** – on unexpected error.

**list\_names**(\*, project='default', remote='local')

List container names.

A helper to get a list of container names from list().

#### Parameters

- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

**Return type** List[str]

**Returns** List of container names.

Raises ***LXDError*** – on unexpected error.

**profile\_edit**(\*, profile, config, project='default', remote='local')

Set profile configuration.

#### Parameters

- **profile** (str) – Name of profile.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

- **config** (Dict[str, Any]) –

Raises **LXDError** – on unexpected error.

Return type None

**profile\_show**(\*, *profile*, *project*='default', *remote*='local')

Get profile configuration.

Parameters

- **profile** (str) – Name of profile.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type Dict[str, Any]

**project\_create**(\*, *project*, *remote*='local')

Create project.

Parameters

- **project** (str) – Name of LXD project to create.
- **remote** (str) – Name of LXD remote to create project on.

Raises **LXDError** – on unexpected error.

Return type None

**project\_delete**(\*, *project*, *remote*='local')

Delete project, if it exists.

Parameters

- **project** (str) – Name of LXD project to delete.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**project\_list**(*remote*='local')

Get list of projects.

Parameters **remote** (str) – Name of LXD remote to query.

Return type List[str]

Returns List of project names.

Raises **LXDError** – on unexpected error.

**publish**(\*, *instance\_name*, *alias*=None, *force*=False, *image\_remote*='local', *project*='default', *remote*='local')

Publish image from instance.

Parameters

- **instance\_name** (str) – Name of instance to publish image from.
- **alias** (Optional[str]) – New alias to define at target.
- **force** (bool) – Force publishing of image, even if container is running.

- **image\_remote** (str) – Name of remote to publish image to.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote instance is found on.

Raises **LXDError** – on unexpected error.

Return type None

**remote\_add**(\*, *remote*, *addr*, *protocol*='simplestreams')

Add a public remote.

Parameters

- **remote** (str) – Name of remote to add.
- **addr** (str) – Address of remote.
- **protocol** (str) – Name of protocol (“simplestreams” or “lxd”).

Raises **LXDError** – on unexpected error.

Return type None

**remote\_list**()

Get list of remotes.

Return type Dict[str, Any]

Returns dictionary with remote name mapping to config.

**start**(\*, *instance\_name*, *project*='default', *remote*='local')

Start container.

Parameters

- **instance\_name** (str) – Name of instance to start.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**stop**(\*, *instance\_name*, *force*=False, *timeout*=- 1, *project*='default', *remote*='local')

Stop container.

Parameters

- **instance\_name** (str) – Name of instance to stop.
- **force** (bool) – Force instance to stop.
- **timeout** (int) – Timeout in seconds. -1 is no timeout.
- **project** (str) – Name of LXD project.
- **remote** (str) – Name of LXD remote.

Raises **LXDError** – on unexpected error.

Return type None

**class** craft\_providers.lxd.**LXD**(\*, *lxd\_path*=PosixPath('lxd'))

Bases: object

Interface to *lxd* command-line.

**Parameters** `lxd_path` (Path) – Path to lxd.

**Variables** `minimum_required_version` – Minimum lxd version required for compatibility.

**init**(\*, *auto=False*, *sudo=False*)

Initialize LXD.

Sudo is required if user is not in lxd group.

**Parameters**

- **auto** (bool) – Use default settings.
- **sudo** (bool) – Use sudo to invoke init.

**Return type** None

**is\_supported\_version**()

Check if LXD version is supported.

A helper to check if LXD meets minimum supported version for craft-providers (currently  $\geq 4.0$ ).

**Return type** bool

**Returns** True if installed version is supported.

**minimum\_required\_version** = '4.0'

**version**()

Query LXD version.

The version is of the format: <major>.<minor>[.<micro>]

Version examples: - 4.13 - 4.0.5 - 2.0.12

**Return type** str

**Returns** Version string.

**wait\_ready**(\*, *sudo=False*, *timeout=None*)

Wait until LXD is ready.

Sudo is required if user is not in lxd group.

**Parameters**

- **sudo** (bool) – Use sudo to invoke waitready.
- **timeout** (Optional[int]) – Timeout in seconds.

**Return type** None

**exception** `craft_providers.lxd.LXDError`(*brief: str*, *details: Optional[str] = None*, *resolution: Optional[str] = None*)

Bases: [craft\\_providers.errors.ProviderError](#)

Unexpected LXD error.

**Parameters**

- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

**exception** `craft_providers.lxd.LXDInstallationError(reason, *, details=None)`

Bases: [craft\\_providers.lxd.errors.LXDError](#)

LXD Installation Error.

#### Parameters

- **reason** (str) – Reason for install failure.
- **details** (Optional[str]) – Optional details to include.

**brief:** str

**class** `craft_providers.lxd.LXDInstance(*, name, default_command_environment=None, project='default', remote='local', lxc=None)`

Bases: [craft\\_providers.executor.Executor](#)

LXD Instance Lifecycle.

#### Parameters

- **name** (str) –
- **default\_command\_environment** (Optional[Dict[str, Optional[str]]]) –
- **project** (str) –
- **remote** (str) –
- **lxc** (Optional[LXC]) –

**delete**(*force=True*)

Delete instance.

**Parameters** **force** (bool) – Delete even if running.

**Raises** [LXDError](#) – On unexpected error.

**Return type** None

**execute\_popen**(*command, \*, cwd=None, env=None, \*\*kwargs*)

Execute a command in instance, using `subprocess.Popen()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

#### Parameters

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments to pass.
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**execute\_run**(*command, \*, cwd=None, env=None, \*\*kwargs*)

Execute a command using `subprocess.run()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

#### Parameters

- **command** (List[str]) – Command to execute.

- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to subprocess.run().
- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** **subprocess.CalledProcessError** – if command fails and check is True.

**exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**Raises** **LXDError** – On unexpected error.

**is\_mounted**(\*, *host\_source*, *target*)

Check if path is mounted at target.

**Parameters**

- **host\_source** (Path) – Host path to check.
- **target** (PurePath) – Instance path to check.

**Return type** bool

**Returns** True if host\_source is mounted at target.

**Raises** **LXDError** – On unexpected error.

**is\_running()**

Check if instance is running.

**Return type** bool

**Returns** True if instance is running.

**Raises** **LXDError** – On unexpected error.

**launch**(\*, *image*, *image\_remote*, *map\_user\_uid=False*, *ephemeral=False*, *uid=None*)

Launch instance.

**Parameters**

- **image** (str) – Image name to launch.
- **image\_remote** (str) – Image remote name.
- **map\_user\_id** – Whether id mapping should be used.
- **uid** (Optional[int]) – If map\_user\_id is True, the host user ID to map to instance root.
- **ephemeral** (bool) – Flag to enable ephemeral instance.
- **map\_user\_uid** (bool) –

**Raises** **LXDError** – On unexpected error.

**Return type** None

**mount**(\*, *host\_source*, *target*, *device\_name=None*)

Mount host source directory to target mount point.



Checks first to see if already mounted. If no device name is given, it will be generated with the format “disk-`{target.as_posix()}`”.

#### Parameters

- **host\_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.
- **device\_name** (Optional[str]) – Name for disk device.

Raises **LXDError** – On unexpected error.

Return type None

**pull\_file**(\*, *source*, *destination*)

Copy a file from the environment to host.

#### Parameters

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (`destination.parent`) must exist.

#### Raises

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **LXDError** – On unexpected error copying file.

Return type None

**push\_file**(\*, *source*, *destination*)

Copy a file from the host into the environment.

#### Parameters

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (`destination.parent`) must exist.

#### Raises

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **LXDError** – On unexpected error copying file.

Return type None

**push\_file\_io**(\*, *destination*, *content*, *file\_mode*, *group*=‘root’, *user*=‘root’)

Create or replace file with content and file mode.

#### Parameters

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. ‘0644’).
- **group** (str) – File group owner/id.
- **user** (str) – File user owner/id.

Raises **LXDError** – On unexpected error.

Return type None

**start()**

Start instance.

**Raises** *LXDError* – on unexpected error.

**Return type** None

**stop()**

Stop instance.

**Raises** *LXDError* – on unexpected error.

**Return type** None

**supports\_mount()**

Check if instance supports mounting from host.

**Return type** bool

**Returns** True if mount is supported.

**unmount(target)**

Unmount mount target shared with host.

**Parameters** *target* (Path) – Target shared with host to unmount.

**Raises** *LXDError* – On failure to unmount target.

**Return type** None

**unmount\_all()**

Unmount all mounts shared with host.

**Raises** *LXDError* – On failure to unmount target.

**Return type** None

**craft\_providers.lxd.configure\_builddd\_image\_remote(lxc=<craft\_providers.lxd.lxc.LXC object>)**

Configure builddd remote, adding remote as required.

**Parameters** *lxc* (*LXC*) – LXC client.

**Return type** str

**Returns** Name of remote to pass to launcher.

**craft\_providers.lxd.ensure\_lxd\_is\_ready(\*, remote='local', lxc=<craft\_providers.lxd.lxc.LXC object>, lxd=<craft\_providers.lxd.lxd.LXD object>)**

Ensure LXD is ready for use.

**Raises** *LXDError* – on error.

**Parameters**

- **remote** (str) –
- **lxc** (*LXC*) –
- **lxd** (*LXD*) –

**Return type** None

**craft\_providers.lxd.install(sudo=True)**

Install LXD.

Install application, using sudo if specified.

**Return type** str

**Returns** LXD version.

**Raises**

- *LXDInstallationError* – on installation error.
- *LXDError* – on unexpected error.

**Parameters** *sudo* (bool) –

`craft_providers.lxd.is_initialized(*, remote, lxc)`

Verify that LXD has been initialized and configuration looks valid.

If LXD has been installed but the user has not initialized it (lxd init), the default profile won't have devices configured. Trying to launch an instance or create a project using this profile will result in failures.

**Return type** bool

**Returns** True if initialized, else False.

**Parameters**

- **remote** (str) –
- **lxc** (*LXC*) –

`craft_providers.lxd.is_installed()`

Check if LXD is installed (and found on PATH).

**Return type** bool

**Returns** True if lxd is installed.

`craft_providers.lxd.is_user_permitted()`

Check if user has permissions to connect to LXD.

**Return type** bool

**Returns** True if user has correct permissions.

`craft_providers.lxd.launch(name, *, base_configuration, image_name, image_remote, auto_clean=False, auto_create_project=False, ephemeral=False, map_user_uid=False, uid=None, use_snapshots=False, project='default', remote='local', lxc=<craft_providers.lxc.LXC object>)`

Create, start, and configure instance.

If `auto_clean` is enabled, automatically delete an existing instance that is deemed to be incompatible, rebuilding it with the specified environment.

**Parameters**

- **name** (str) – Name of instance.
- **base\_configuration** (*Base*) – Base configuration to apply to instance.
- **image\_name** (str) – LXD image to use, e.g. “20.04”.
- **image\_remote** (str) – LXD image to use, e.g. “ubuntu”.
- **auto\_clean** (bool) – Automatically clean instance, if incompatible.
- **auto\_create\_project** (bool) – Automatically create LXD project, if needed.
- **ephemeral** (bool) – Create ephemeral instance.
- **map\_user\_uid** (bool) – Map host uid/gid to instance's root uid/gid.
- **uid** (Optional[int]) – The uid to be mapped, if `map_user_id` is enabled.

- **use\_snapshots** (bool) – Use LXD snapshots for bootstrapping images.
- **project** (str) – LXD project to create instance in.
- **remote** (str) – LXD remote to create instance on.
- **lxc** (*LXC*) – LXC client.

**Return type** *LXDInstance*

**Returns** LXD instance.

**Raises**

- *BaseConfigurationError* – on unexpected error configuration base.
- *LXDError* – on unexpected LXD error.

### 3.1.4 craft\_providers.multipass package

#### Submodules

#### craft\_providers.multipass.errors module

Multipass Errors.

**exception** craft\_providers.multipass.errors.**MultipassError**(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: *craft\_providers.errors.ProviderError*

Unexpected Multipass error.

**Parameters**

- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

**exception** craft\_providers.multipass.errors.**MultipassInstallationError**(*reason, \*, details=None*)

Bases: *craft\_providers.multipass.errors.MultipassError*

Multipass Installation Error.

**Parameters**

- **reason** (str) – Reason for install failure.
- **details** (Optional[str]) – Optional details to include.

**brief:** str

### craft\_providers.multipass.installer module

Multipass Provider.

`craft_providers.multipass.installer.install()`

Install Multipass.

**Return type** str

**Returns** Multipass version.

**Raises** *MultipassInstallationError* – on error.

`craft_providers.multipass.installer.is_installed()`

Check if Multipass is installed (and found on PATH).

**Return type** bool

**Returns** True if multipass is installed.

### craft\_providers.multipass.multipass module

API provider for Multipass.

This implementation interfaces with multipass using the *multipass* command-line utility.

**class** `craft_providers.multipass.multipass.Multipass(*, multipass_path=PosixPath('multipass'))`

Bases: object

Wrapper for multipass command.

**Parameters** `multipass_path` (Path) – Path to multipass command to use.

**Variables** `minimum_required_version` – Minimum required version for compatibility.

**delete**(\*, instance\_name, purge=True)

Passthrough for running multipass delete.

**Parameters**

- **instance\_name** (str) – The name of the instance\_name to delete.
- **purge** – Flag to purge the instance\_name's image after deleting.

**Raises** *MultipassError* – on error.

**Return type** None

**exec**(\*, command, instance\_name, runner=<function run>, \*\*kwargs)

Execute command in instance\_name with specified runner.

**Parameters**

- **command** (List[str]) – Command to execute in the instance.
- **instance\_name** (str) – Name of instance to execute in.
- **runner** (Callable) – Execution function to invoke, e.g. subprocess.run or Popen. First argument is finalized command with the attached kwargs.
- **kwargs** – Additional kwargs for runner.

**Returns** Runner's instance.

**info**(\*, instance\_name)

Get information/state for instance.

**Return type** Dict[str, Any]

**Returns** Parsed json data from info command.

**Raises** *MultipassError* – On error.

**Parameters** *instance\_name* (str) –

**is\_supported\_version()**

Check if Multipass version is supported.

A helper to check if Multipass meets minimum supported version for craft-providers.

**Return type** bool

**Returns** True if installed version is supported.

**launch**(\*, *instance\_name*, *image*, *cpus=None*, *mem=None*, *disk=None*)

Launch multipass VM.

**Parameters**

- **instance\_name** (str) – The name the launched instance will have.
- **image** (str) – Name of image to create the instance with.
- **cpus** (Optional[str]) – Amount of virtual CPUs to assign to the launched instance.
- **mem** (Optional[str]) – Amount of RAM to assign to the launched instance.
- **disk** (Optional[str]) – Amount of disk space the launched instance will have.

**Raises** *MultipassError* – on error.

**Return type** None

**list()**

List names of VMs.

**Return type** List[str]

**Returns** Data from stdout if instance exists, else None.

**Raises** *MultipassError* – On error.

**minimum\_required\_version** = '1.7'

**mount**(\*, *source*, *target*, *uid\_map=None*, *gid\_map=None*)

Mount host source path to target.

**Parameters**

- **source** (Path) – Path of local directory to mount.
- **target** (str) – Target mount points, in <name>[:<path>] format, where <name> is an instance name, and optional <path> is the mount point. If omitted, the mount point will be the same as the source's absolute path.
- **uid\_map** (Optional[Dict[str, str]]) – A mapping of user IDs for use in the mount of the form <host-id> -> <instance-id>. File and folder ownership will be mapped from <host-id> to <instance-id> inside the instance.
- **gid\_map** (Optional[Dict[str, str]]) – A mapping of group IDs for use in the mount of the form <host-id> -> <instance-id>. File and folder ownership will be mapped from <host-id> to <instance-id> inside the instance.

**Return type** None

**start**(\*, *instance\_name*)

Start VM instance.

**Parameters** *instance\_name* (str) – the name of the instance to start.

**Raises** *MultipassError* – on error.

**Return type** None

**stop**(\*, *instance\_name*, *delay\_mins*=0)

Stop VM instance.

**Parameters**

- **instance\_name** (str) – the name of the instance\_name to stop.
- **delay\_mins** (int) – Delay shutdown for specified number of minutes.

**Raises** *MultipassError* – on error.

**Return type** None

**transfer**(\*, *source*, *destination*)

Transfer to destination path with source IO.

**Parameters**

- **source** (str) – The source path, prefixed with <name:> for a path inside the instance.
- **destination** (str) – The destination path, prefixed with <name:> for a path inside the instance.

**Raises** *MultipassError* – On error.

**Return type** None

**transfer\_destination\_io**(\*, *source*, *destination*, *chunk\_size*=4096)

Transfer from source file to destination IO.

Note that this can't use std{in,out}=open(...) due to LP #1849753.

**Parameters**

- **source** (str) – The source path, prefixed with <name:> for a path inside the instance.
- **destination** (BufferedIOBase) – An IO stream to write to.
- **chunk\_size** (int) – Number of bytes to transfer at a time. Defaults to 4096.

**Raises** *MultipassError* – On error.

**Return type** None

**transfer\_source\_io**(\*, *source*, *destination*, *chunk\_size*=4096)

Transfer to destination path with source IO.

Note that this can't use std{in,out}=open(...) due to LP #1849753.

**Parameters**

- **source** (BufferedIOBase) – An IO stream to read from.
- **destination** (str) – The destination path, prefixed with <name:> for a path inside the instance.
- **chunk\_size** (int) – Number of bytes to transfer at a time. Defaults to 4096.

**Raises** *MultipassError* – On error.

**Return type** None

**umount**(*\*, mount*)

Unmount target in VM.

**Parameters** **mount** (str) – Mount point in <name>[:<path>] format, where <name> are instance names, and optional <path> are mount points. If omitted, all mounts will be removed from the named instance.

**Raises** *MultipassError* – On error.

**Return type** None

**version**()

Get multipass and multipassd versions.

**Return type** Tuple[str, Optional[str]]

**Returns** Tuple of parsed versions (multipass, multipassd). multipassd may be None if Multipass is not yet ready.

**wait\_until\_ready**(*\*, retry\_wait=0.25, timeout=None*)

Wait until Multipass is ready (upon install/startup).

**Parameters**

- **retry\_wait** (float) – Time to sleep between retries.
- **timeout** (Optional[float]) – Timeout in seconds.

**Return type** Tuple[str, Optional[str]]

**Returns** Tuple of parsed versions (multipass, multipassd). multipassd may be None if Multipass is not ready and the timeout limit is reached.

## **craft\_providers.multipass.multipass\_instance module**

Multipass Instance.

**class** craft\_providers.multipass.multipass\_instance.**MultipassInstance**(*\*, name, multipass=None*)

Bases: *craft\_providers.executor.Executor*

Multipass Instance Lifecycle.

**Parameters**

- **name** (str) – Name of multipass instance.
- **multipass** (Optional[Multipass]) –

**delete**()

Delete instance and purge.

**Return type** None

**execute\_popen**(*command, \*, cwd=None, env=None, \*\*kwargs*)

Execute process in instance using subprocess.Popen().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via env parameter.

**Parameters**

- **command** (List[str]) – Command to execute.



- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments for subprocess.Popen().
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**execute\_run**(*command*, \*, *cwd*=None, *env*=None, *\*\*kwargs*)

Execute command using subprocess.run().

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via env parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to subprocess.run().
- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** **subprocess.CalledProcessError** – if command fails and check is True.

**exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**Raises** **MultipassError** – On unexpected failure.

**is\_mounted**(\*, *host\_source*, *target*)

Check if path is mounted at target.

**Parameters**

- **host\_source** (Path) – Host path to check.
- **target** (PurePath) – Instance path to check.

**Return type** bool

**Returns** True if host\_source is mounted at target.

**Raises** **MultipassError** – On unexpected failure.

**is\_running()**

Check if instance is running.

**Return type** bool

**Returns** True if instance is running.

**Raises** **MultipassError** – On unexpected failure.

**launch**(\*, *image*, *cpus*=2, *disk\_gb*=256, *mem\_gb*=2)

Launch instance.

**Parameters**

- **image** (str) – Name of image to create the instance with.
- **instance\_cpus** – Number of CPUs.
- **instance\_disk\_gb** – Disk allocation in gigabytes.
- **instance\_mem\_gb** – Memory allocation in gigabytes.
- **instance\_name** – Name of instance to use/create.
- **instance\_stop\_time\_mins** – Stop time delay in minutes.
- **cpus** (int) –
- **disk\_gb** (int) –
- **mem\_gb** (int) –

Raises **MultipassError** – On unexpected failure.

Return type None

**mount**(\* , host\_source, target)

Mount host host\_source directory to target mount point.

Checks first to see if already mounted.

Parameters

- **host\_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

Raises **MultipassError** – On unexpected failure.

Return type None

**pull\_file**(\* , source, destination)

Copy a file from the environment to host.

Parameters

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (destination.parent) must exist.

Raises

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **MultipassError** – On unexpected error copying file.

Return type None

**push\_file**(\* , source, destination)

Copy a file from the host into the environment.

Parameters

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (destination.parent) must exist.

Raises

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **MultipassError** – On unexpected error copying file.

**Return type** None

**push\_file\_io**(\*, *destination*, *content*, *file\_mode*, *group*='root', *user*='root')

Create or replace file with content and file mode.

Multipass transfers data as “ubuntu” user, forcing us to first copy a file to a temporary location before moving to a (possibly) root-owned location and with appropriate permissions.

**Parameters**

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File group owner/id.
- **user** (str) – File user owner/id.

**Return type** None

**start**()

Start instance.

**Raises** *MultipassError* – On unexpected failure.

**Return type** None

**stop**(\*, *delay\_mins*=0)

Stop instance.

**Parameters** **delay\_mins** (int) – Delay shutdown for specified minutes.

**Raises** *MultipassError* – On unexpected failure.

**Return type** None

**unmount**(*target*)

Unmount mount target shared with host.

**Parameters** **target** (Path) – Target shared with host to unmount.

**Raises** *MultipassError* – On failure to unmount target.

**Return type** None

**unmount\_all**()

Unmount all mounts shared with host.

**Raises** *MultipassError* – On failure to unmount target.

**Return type** None

## Module contents

Multipass provider support package.

**class** `craft_providers.multipass.Multipass`(\*, *multipass\_path*=*PosixPath('multipass')*)

Bases: object

Wrapper for multipass command.

**Parameters** **multipass\_path** (Path) – Path to multipass command to use.

**Variables** **minimum\_required\_version** – Minimum required version for compatibility.

**delete**(\*, *instance\_name*, *purge=True*)

Passthrough for running multipass delete.

**Parameters**

- **instance\_name** (str) – The name of the instance\_name to delete.
- **purge** – Flag to purge the instance\_name’s image after deleting.

**Raises** *MultipassError* – on error.

**Return type** None

**exec**(\*, *command*, *instance\_name*, *runner=<function run>*, *\*\*kwargs*)

Execute command in instance\_name with specified runner.

**Parameters**

- **command** (List[str]) – Command to execute in the instance.
- **instance\_name** (str) – Name of instance to execute in.
- **runner** (Callable) – Execution function to invoke, e.g. subprocess.run or Popen. First argument is finalized command with the attached kwargs.
- **kwargs** – Additional kwargs for runner.

**Returns** Runner’s instance.

**info**(\*, *instance\_name*)

Get information/state for instance.

**Return type** Dict[str, Any]

**Returns** Parsed json data from info command.

**Raises** *MultipassError* – On error.

**Parameters** **instance\_name** (str) –

**is\_supported\_version**()

Check if Multipass version is supported.

A helper to check if Multipass meets minimum supported version for craft-providers.

**Return type** bool

**Returns** True if installed version is supported.

**launch**(\*, *instance\_name*, *image*, *cpus=None*, *mem=None*, *disk=None*)

Launch multipass VM.

**Parameters**

- **instance\_name** (str) – The name the launched instance will have.
- **image** (str) – Name of image to create the instance with.
- **cpus** (Optional[str]) – Amount of virtual CPUs to assign to the launched instance.
- **mem** (Optional[str]) – Amount of RAM to assign to the launched instance.
- **disk** (Optional[str]) – Amount of disk space the launched instance will have.

**Raises** *MultipassError* – on error.

**Return type** None

**list()**

List names of VMs.

**Return type** List[str]

**Returns** Data from stdout if instance exists, else None.

**Raises** *MultipassError* – On error.

**minimum\_required\_version** = '1.7'

**mount**(\*, source, target, uid\_map=None, gid\_map=None)

Mount host source path to target.

**Parameters**

- **source** (Path) – Path of local directory to mount.
- **target** (str) – Target mount points, in <name>[:<path>] format, where <name> is an instance name, and optional <path> is the mount point. If omitted, the mount point will be the same as the source's absolute path.
- **uid\_map** (Optional[Dict[str, str]]) – A mapping of user IDs for use in the mount of the form <host-id> -> <instance-id>. File and folder ownership will be mapped from <host-id> to <instance-id> inside the instance.
- **gid\_map** (Optional[Dict[str, str]]) – A mapping of group IDs for use in the mount of the form <host-id> -> <instance-id>. File and folder ownership will be mapped from <host-id> to <instance-id> inside the instance.

**Return type** None

**start**(\*, instance\_name)

Start VM instance.

**Parameters** **instance\_name** (str) – the name of the instance to start.

**Raises** *MultipassError* – on error.

**Return type** None

**stop**(\*, instance\_name, delay\_mins=0)

Stop VM instance.

**Parameters**

- **instance\_name** (str) – the name of the instance\_name to stop.
- **delay\_mins** (int) – Delay shutdown for specified number of minutes.

**Raises** *MultipassError* – on error.

**Return type** None

**transfer**(\*, source, destination)

Transfer to destination path with source IO.

**Parameters**

- **source** (str) – The source path, prefixed with <name:> for a path inside the instance.
- **destination** (str) – The destination path, prefixed with <name:> for a path inside the instance.

**Raises** *MultipassError* – On error.

**Return type** None

**transfer\_destination\_io**(\*, *source*, *destination*, *chunk\_size*=4096)

Transfer from source file to destination IO.

Note that this can't use `std{in,out}=open(...)` due to LP #1849753.

**Parameters**

- **source** (str) – The source path, prefixed with <name:> for a path inside the instance.
- **destination** (BufferedIOBase) – An IO stream to write to.
- **chunk\_size** (int) – Number of bytes to transfer at a time. Defaults to 4096.

**Raises** *MultipassError* – On error.

**Return type** None

**transfer\_source\_io**(\*, *source*, *destination*, *chunk\_size*=4096)

Transfer to destination path with source IO.

Note that this can't use `std{in,out}=open(...)` due to LP #1849753.

**Parameters**

- **source** (BufferedIOBase) – An IO stream to read from.
- **destination** (str) – The destination path, prefixed with <name:> for a path inside the instance.
- **chunk\_size** (int) – Number of bytes to transfer at a time. Defaults to 4096.

**Raises** *MultipassError* – On error.

**Return type** None

**umount**(\*, *mount*)

Unmount target in VM.

**Parameters** **mount** (str) – Mount point in <name>[:<path>] format, where <name> are instance names, and optional <path> are mount points. If omitted, all mounts will be removed from the named instance.

**Raises** *MultipassError* – On error.

**Return type** None

**version**()

Get multipass and multipassd versions.

**Return type** Tuple[str, Optional[str]]

**Returns** Tuple of parsed versions (multipass, multipassd). multipassd may be None if Multipass is not yet ready.

**wait\_until\_ready**(\*, *retry\_wait*=0.25, *timeout*=None)

Wait until Multipass is ready (upon install/startup).

**Parameters**

- **retry\_wait** (float) – Time to sleep between retries.
- **timeout** (Optional[float]) – Timeout in seconds.

**Return type** Tuple[str, Optional[str]]

**Returns** Tuple of parsed versions (multipass, multipassd). multipassd may be None if Multipass is not ready and the timeout limit is reached.

**exception** `craft_providers.multipass.MultipassError`(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: `craft_providers.errors.ProviderError`

Unexpected Multipass error.

#### Parameters

- **brief** (str) –
- **details** (Optional[str]) –
- **resolution** (Optional[str]) –

**brief:** str

**exception** `craft_providers.multipass.MultipassInstallationError`(*reason, \*, details=None*)

Bases: `craft_providers.multipass.errors.MultipassError`

Multipass Installation Error.

#### Parameters

- **reason** (str) – Reason for install failure.
- **details** (Optional[str]) – Optional details to include.

**brief:** str

**class** `craft_providers.multipass.MultipassInstance`(*\*, name, multipass=None*)

Bases: `craft_providers.executor.Executor`

Multipass Instance Lifecycle.

#### Parameters

- **name** (str) – Name of multipass instance.
- **multipass** (Optional[`Multipass`]) –

**delete()**

Delete instance and purge.

**Return type** None

**execute\_popen**(*command, \*, cwd=None, env=None, \*\*kwargs*)

Execute process in instance using `subprocess.Popen()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

#### Parameters

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments for `subprocess.Popen()`.
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**execute\_run**(*command, \*, cwd=None, env=None, \*\*kwargs*)

Execute command using `subprocess.run()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to `subprocess.run()`.
- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** `subprocess.CalledProcessError` – if command fails and `check` is True.

**exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**Raises** `MultipassError` – On unexpected failure.

**is\_mounted(\*, host\_source, target)**

Check if path is mounted at target.

**Parameters**

- **host\_source** (Path) – Host path to check.
- **target** (PurePath) – Instance path to check.

**Return type** bool

**Returns** True if `host_source` is mounted at target.

**Raises** `MultipassError` – On unexpected failure.

**is\_running()**

Check if instance is running.

**Return type** bool

**Returns** True if instance is running.

**Raises** `MultipassError` – On unexpected failure.

**launch(\*, image, cpus=2, disk\_gb=256, mem\_gb=2)**

Launch instance.

**Parameters**

- **image** (str) – Name of image to create the instance with.
- **instance\_cpus** – Number of CPUs.
- **instance\_disk\_gb** – Disk allocation in gigabytes.
- **instance\_mem\_gb** – Memory allocation in gigabytes.
- **instance\_name** – Name of instance to use/create.
- **instance\_stop\_time\_mins** – Stop time delay in minutes.
- **cpus** (int) –



- **disk\_gb** (int) –

- **mem\_gb** (int) –

Raises **MultipassError** – On unexpected failure.

Return type None

**mount**(\* , *host\_source*, *target*)

Mount host *host\_source* directory to target mount point.

Checks first to see if already mounted.

Parameters

- **host\_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

Raises **MultipassError** – On unexpected failure.

Return type None

**pull\_file**(\* , *source*, *destination*)

Copy a file from the environment to host.

Parameters

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (*destination.parent*) must exist.

Raises

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **MultipassError** – On unexpected error copying file.

Return type None

**push\_file**(\* , *source*, *destination*)

Copy a file from the host into the environment.

Parameters

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (*destination.parent*) must exist.

Raises

- **FileNotFoundError** – If source file or destination’s parent directory does not exist.
- **MultipassError** – On unexpected error copying file.

Return type None

**push\_file\_io**(\* , *destination*, *content*, *file\_mode*, *group*='root', *user*='root')

Create or replace file with content and file mode.

Multipass transfers data as “ubuntu” user, forcing us to first copy a file to a temporary location before moving to a (possibly) root-owned location and with appropriate permissions.

Parameters

- **destination** (PurePath) – Path to file.

- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File group owner/id.
- **user** (str) – File user owner/id.

**Return type** None

**start()**

Start instance.

**Raises** *MultipassError* – On unexpected failure.

**Return type** None

**stop**(\*, *delay\_mins=0*)

Stop instance.

**Parameters** **delay\_mins** (int) – Delay shutdown for specified minutes.

**Raises** *MultipassError* – On unexpected failure.

**Return type** None

**unmount**(*target*)

Unmount mount target shared with host.

**Parameters** **target** (Path) – Target shared with host to unmount.

**Raises** *MultipassError* – On failure to unmount target.

**Return type** None

**unmount\_all()**

Unmount all mounts shared with host.

**Raises** *MultipassError* – On failure to unmount target.

**Return type** None

**craft\_providers.multipass.ensure\_multipass\_is\_ready**(\*, *multipass=<craft\_providers.multipass.multipass.Multipass object>*)

Ensure Multipass is ready for use.

**Raises** *MultipassError* – on error.

**Parameters** **multipass** (*Multipass*) –

**Return type** None

**craft\_providers.multipass.install()**

Install Multipass.

**Return type** str

**Returns** Multipass version.

**Raises** *MultipassInstallationError* – on error.

**craft\_providers.multipass.is\_installed()**

Check if Multipass is installed (and found on PATH).

**Return type** bool

**Returns** True if multipass is installed.

`craft_providers.multipass.launch(name, *, base_configuration, image_name, cpus=2, disk_gb=64, mem_gb=2, auto_clean=False)`

Create, start, and configure instance.

If `auto_clean` is enabled, automatically delete an existing instance that is deemed to be incompatible, rebuilding it with the specified environment.

#### Parameters

- **name** (str) – Name of instance.
- **base\_configuration** (*Base*) – Base configuration to apply to instance.
- **image\_name** (str) – Multipass image to use, e.g. `snapcraft:core20`.
- **cpus** (int) – Number of CPUs.
- **disk\_gb** (int) – Disk allocation in gigabytes.
- **mem\_gb** (int) – Memory allocation in gigabytes.
- **auto\_clean** (bool) – Automatically clean instance, if incompatible.

**Return type** *MultipassInstance*

**Returns** Multipass instance.

#### Raises

- *BaseConfigurationError* – on unexpected error configuration base.
- *MultipassError* – on unexpected Multipass error.

### 3.1.5 craft\_providers.util package

#### Submodules

##### craft\_providers.util.env\_cmd module

Helper(s) for env command.

`craft_providers.util.env_cmd.formulate_command(env=None, *, chdir=None, ignore_environment=False)`

Create an env command with the specified environment.

For each key-value, the env command will include the `key=value` argument to the env command.

If a variable is `None`, then the env `-u` parameter will be used to unset it.

An empty environment will simply yield the env command.

NOTE: not all versions of *env* support `-chdir`, it is up to the caller to ensure compatibility.

#### Parameters

- **env** (Optional[Dict[str, Optional[str]]]) – Environment flags to set/unset.
- **chdir** (Optional[Path]) – Optional directory to run in.
- **ignore\_environment** (bool) – Start with an empty environment.

**Return type** List[str]

**Returns** List of env command strings.

### craft\_providers.util.os\_release module

Parser for /etc/os-release.

`craft_providers.util.os_release.parse_os_release(content)`

Parser for /etc/os-release.

Format documentation at:

<https://www.freedesktop.org/software/systemd/man/os-release.html>

Example os-release contents:

```
NAME="Ubuntu"
VERSION="20.10 (Groovy Gorilla)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.10"
VERSION_ID="20.10"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=groovy
UBUNTU_CODENAME=groovy
```

**Parameters** `content` (str) – String contents of os-release file.

**Return type** Dict[str, str]

**Returns** Dictionary of key-mappings found in os-release. Values are stripped of encapsulating quotes.

### craft\_providers.util.snap\_cmd module

Helper(s) for snap command.

`craft_providers.util.snap_cmd.formulate_local_install_command(classic, dangerous, snap_path)`

Formulate snap install command.

**Parameters**

- **classic** (bool) – Flag to enable installation of classic snap.
- **dangerous** (bool) – Flag to enable installation of snap without ack.
- **snap\_path** (Path) –

**Return type** List[str]

**Returns** List of command parts.

`craft_providers.util.snap_cmd.formulate_refresh_command(snap_name, channel)`

Formulate snap refresh command.

**Parameters**

- **snap\_name** (str) – The name of the channel.
- **channel** (str) – The channel to install the snap from.

**Return type** List[str]

**Returns** List of command parts.

`craft_providers.util.snap_cmd.formulate_remote_install_command(snap_name, channel, classic)`  
Formulate the command to snap install from Store.

**Parameters**

- **snap\_name** (str) – The name of the channel.
- **channel** (str) – The channel to install the snap from.
- **classic** (bool) – Flag to enable installation of classic snap.
- **dangerous** – Flag to enable installation of snap without ack.

**Return type** List[str]

**Returns** List of command parts.

`craft_providers.util.snap_cmd.formulate_remove_command(snap_name)`  
Formulate snap remove command.

**Parameters** **snap\_name** (str) – The name of the channel.

**Return type** List[str]

**Returns** List of command parts.

## craft\_providers.util.temp\_paths module

Helpers for temporary files.

`craft_providers.util.temp_paths.home_temporary_directory()`  
Create temporary directory in home directory where Multipass has access.

**Return type** Iterator[Path]

`craft_providers.util.temp_paths.home_temporary_file()`  
Create a temporary directory in the home directory where Multipass has access.

**Return type** Iterator[Path]

## Module contents

Utility modules.

## 3.2 Submodules

### 3.2.1 craft\_providers.base module

Base configuration module.

**class** `craft_providers.base.Base`  
Bases: `abc.ABC`

Interface for providers to configure instantiated environments.

Defines how to setup/configure an environment that has been instantiated by a provider and prepare it for some operation, e.g. execute build. It must account for:

- (1) the OS type and version.
- (2) the provided image that was launched, e.g. bootstrapping a minimal image versus a more fully featured one.
- (3) any dependencies that are required for the operation to complete, e.g. installed applications, networking configuration, etc. This includes any environment configuration that the application will assume is available.

**Variables** `compatibility_tag` – Tag/Version for variant of build configuration and setup. Any change to this version would indicate that prior [versioned] instances are incompatible and must be cleaned. As such, any new value should be unique to old values (e.g. incrementing). It is suggested to extend this tag, not overwrite it, e.g.: `compatibility_tag = f'{appname}-{Base.compatibility_tag}.{apprevision}'` to ensure base compatibility levels are maintained.

**compatibility\_tag:** `str = 'base-v0'`

**abstract** `get_command_environment()`

Get command environment to use when executing commands.

**Return type** `Dict[str, Optional[str]]`

**Returns** Dictionary of environment, allowing None as a value to indicate that a value should be unset.

**abstract** `setup(*, executor, retry_wait=0.25, timeout=None)`

Prepare base instance for use by the application.

Wait for environment to become ready and configure it. At completion of setup, the executor environment should have networking up and have all of the installed dependencies required for subsequent use by the application.

Setup should not be called more than once in a given instance to refresh/update the environment, use *warmup* for that.

If timeout is specified, abort operation if time has been exceeded.

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

#### Raises

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

**Return type** `None`

**abstract** `wait_until_ready(*, executor, retry_wait=0.25, timeout=None)`

Wait until base instance is ready.

Ensure minimum-required boot services are running. This would be used when starting an environment's container/VM after already [recently] running setup(), e.g. rebooting the instance. Allows the environment to be used without the cost incurred by re-executing the steps unnecessarily.

If timeout is specified, abort operation if time has been exceeded.

#### Parameters

- **executor** (*Executor*) – Executor for target container.

- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- **BaseCompatibilityError** – if instance is incompatible.
- **BaseConfigurationError** – on other unexpected error.

**Return type** None

**abstract** **warmup**(\*, executor, retry\_wait=0.25, timeout=None)

Prepare a previously created and setup instance for use by the application.

Ensure the instance is still valid and wait for environment to become ready.

If timeout is specified, abort operation if time has been exceeded.

**Parameters**

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

**Raises**

- **BaseCompatibilityError** – if instance is incompatible.
- **BaseConfigurationError** – on other unexpected error.

**Return type** None

### 3.2.2 craft\_providers.errors module

Craft provider errors.

**exception** **craft\_providers.errors.ProviderError**(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: Exception

Unexpected error.

**Parameters**

- **brief** (str) – Brief description of error.
- **details** (Optional[str]) – Detailed information.
- **resolution** (Optional[str]) – Recommendation, if any.

**brief:** str

**details:** Optional[str] = None

**resolution:** Optional[str] = None

**craft\_providers.errors.details\_from\_called\_process\_error**(*error*)

Create a consistent ProviderError from command errors.

**Parameters** **error** (CalledProcessError) – CalledProcessError.

**Return type** str

**Returns** Details string.

`craft_providers.errors.details_from_command_error(*, cmd, returncode, stdout=None, stderr=None)`

Create a consistent ProviderError from command errors.

stdout and stderr, if provided, will be stringified using its object representation. This method does not decode byte strings.

**Parameters**

- **cmd** (List[str]) – Command executed.
- **returncode** (int) – Command exit code.
- **stdout** (Union[str, bytes, None]) – Optional stdout to include.
- **stderr** (Union[str, bytes, None]) – Optional stderr to include.

**Return type** str

**Returns** Details string.

### 3.2.3 `craft_providers.executor` module

Executor module.

**class** `craft_providers.executor.Executor`

Bases: `abc.ABC`

Interfaces to execute commands and move data in/out of an environment.

**abstract** `delete()`

Delete instance.

**Return type** None

**abstract** `execute_popen(command, *, cwd=None, env=None, **kwargs)`

Execute a command in instance, using `subprocess.Popen()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments to pass.
- **cwd** (Optional[Path]) –

**Return type** Popen

**Returns** Popen instance.

**abstract** `execute_run(command, *, cwd=None, env=None, **kwargs)`

Execute a command using `subprocess.run()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (List[str]) – Command to execute.
- **env** (Optional[Dict[str, Optional[str]]]) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to `subprocess.run()`.



- **cwd** (Optional[Path]) –

**Return type** CompletedProcess

**Returns** Completed process.

**Raises** **subprocess.CalledProcessError** – if command fails and check is True.

**abstract exists()**

Check if instance exists.

**Return type** bool

**Returns** True if instance exists.

**abstract pull\_file(\*, source, destination)**

Copy a file from the environment to host.

**Parameters**

- **source** (PurePath) – Environment file to copy.
- **destination** (Path) – Host file path to copy to. Parent directory (destination.parent) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **ProviderError** – On error copying file.

**Return type** None

**abstract push\_file(\*, source, destination)**

Copy a file from the host into the environment.

**Parameters**

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (destination.parent) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **ProviderError** – On error copying file.

**Return type** None

**abstract push\_file\_io(\*, destination, content, file\_mode, group='root', user='root')**

Create or replace a file with specified content and file mode.

**Parameters**

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File owner group.
- **user** (str) – File owner user.

**Return type** None

**temporarily\_pull\_file**(\*, *source*, *missing\_ok=False*)

Copy a file from the environment to a temporary file in the host.

This is mainly a layer above *pull\_file* that pulls the file into a temporary path which is cleaned later.

Works as a context manager, provides the file path in the host as target.

#### Parameters

- **source** (Path) – Environment file to copy.
- **missing\_ok** (bool) – Do not raise an error if the file does not exist in the environment; in this case the target will be None.

#### Raises

- **FileNotFoundError** – If source file or destination's parent directory does not exist (and *missing\_ok* is False).
- **ProviderError** – On error copying file content.

**Return type** Generator[Optional[Path], None, None]

## 3.3 Module contents

Craft Providers base package.

**class** `craft_providers.Base`

Bases: `abc.ABC`

Interface for providers to configure instantiated environments.

Defines how to setup/configure an environment that has been instantiated by a provider and prepare it for some operation, e.g. `execute build`. It must account for:

- (1) the OS type and version.
- (2) the provided image that was launched, e.g. bootstrapping a minimal image versus a more fully featured one.
- (3) any dependencies that are required for the operation to complete, e.g. installed applications, networking configuration, etc. This includes any environment configuration that the application will assume is available.

**Variables** `compatibility_tag` – Tag/Version for variant of build configuration and setup. Any change to this version would indicate that prior [versioned] instances are incompatible and must be cleaned. As such, any new value should be unique to old values (e.g. incrementing). It is suggested to extend this tag, not overwrite it, e.g.: `compatibility_tag = f'{appname}-{Base.compatibility_tag}.{appversion}'` to ensure base compatibility levels are maintained.

**compatibility\_tag:** `str = 'base-v0'`

**abstract** `get_command_environment()`

Get command environment to use when executing commands.

**Return type** `Dict[str, Optional[str]]`

**Returns** Dictionary of environment, allowing None as a value to indicate that a value should be unset.

**abstract** `setup`(\*, *executor*, *retry\_wait=0.25*, *timeout=None*)

Prepare base instance for use by the application.

Wait for environment to become ready and configure it. At completion of setup, the executor environment should have networking up and have all of the installed dependencies required for subsequent use by the application.

Setup should not be called more than once in a given instance to refresh/update the environment, use *warmup* for that.

If timeout is specified, abort operation if time has been exceeded.

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

#### Raises

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

#### Return type None

**abstract wait\_until\_ready**(\* , executor, retry\_wait=0.25, timeout=None)

Wait until base instance is ready.

Ensure minimum-required boot services are running. This would be used when starting an environment's container/VM after already [recently] running setup(), e.g. rebooting the instance. Allows the environment to be used without the cost incurred by re-executing the steps unnecessarily.

If timeout is specified, abort operation if time has been exceeded.

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

#### Raises

- *BaseCompatibilityError* – if instance is incompatible.
- *BaseConfigurationError* – on other unexpected error.

#### Return type None

**abstract warmup**(\* , executor, retry\_wait=0.25, timeout=None)

Prepare a previously created and setup instance for use by the application.

Ensure the instance is still valid and wait for environment to become ready.

If timeout is specified, abort operation if time has been exceeded.

#### Parameters

- **executor** (*Executor*) – Executor for target container.
- **retry\_wait** (float) – Duration to sleep() between status checks (if required).
- **timeout** (Optional[float]) – Timeout in seconds.

#### Raises

- *BaseCompatibilityError* – if instance is incompatible.

- **BaseConfigurationError** – on other unexpected error.

**Return type** None

**class** `craft_providers.Executor`

Bases: `abc.ABC`

Interfaces to execute commands and move data in/out of an environment.

**abstract** `delete()`

Delete instance.

**Return type** None

**abstract** `execute_popen(command, *, cwd=None, env=None, **kwargs)`

Execute a command in instance, using `subprocess.Popen()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (`List[str]`) – Command to execute.
- **env** (`Optional[Dict[str, Optional[str]]]`) – Additional environment to set for process.
- **kwargs** – Additional keyword arguments to pass.
- **cwd** (`Optional[Path]`) –

**Return type** `Popen`

**Returns** `Popen` instance.

**abstract** `execute_run(command, *, cwd=None, env=None, **kwargs)`

Execute a command using `subprocess.run()`.

The process' environment will inherit the execution environment's default environment (PATH, etc.), but can be additionally configured via `env` parameter.

**Parameters**

- **command** (`List[str]`) – Command to execute.
- **env** (`Optional[Dict[str, Optional[str]]]`) – Additional environment to set for process.
- **kwargs** – Keyword args to pass to `subprocess.run()`.
- **cwd** (`Optional[Path]`) –

**Return type** `CompletedProcess`

**Returns** Completed process.

**Raises** `subprocess.CalledProcessError` – if command fails and `check` is `True`.

**abstract** `exists()`

Check if instance exists.

**Return type** `bool`

**Returns** `True` if instance exists.

**abstract** `pull_file(*, source, destination)`

Copy a file from the environment to host.

**Parameters**

- **source** (`PurePath`) – Environment file to copy.

- **destination** (Path) – Host file path to copy to. Parent directory (destination.parent) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **ProviderError** – On error copying file.

**Return type** None

**abstract push\_file**(\*, *source*, *destination*)

Copy a file from the host into the environment.

**Parameters**

- **source** (Path) – Host file to copy.
- **destination** (PurePath) – Target environment file path to copy to. Parent directory (destination.parent) must exist.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist.
- **ProviderError** – On error copying file.

**Return type** None

**abstract push\_file\_io**(\*, *destination*, *content*, *file\_mode*, *group*='root', *user*='root')

Create or replace a file with specified content and file mode.

**Parameters**

- **destination** (PurePath) – Path to file.
- **content** (BytesIO) – Contents of file.
- **file\_mode** (str) – File mode string (e.g. '0644').
- **group** (str) – File owner group.
- **user** (str) – File owner user.

**Return type** None

**temporarily\_pull\_file**(\*, *source*, *missing\_ok*=False)

Copy a file from the environment to a temporary file in the host.

This is mainly a layer above *pull\_file* that pulls the file into a temporary path which is cleaned later.

Works as a context manager, provides the file path in the host as target.

**Parameters**

- **source** (Path) – Environment file to copy.
- **missing\_ok** (bool) – Do not raise an error if the file does not exist in the environment; in this case the target will be None.

**Raises**

- **FileNotFoundError** – If source file or destination's parent directory does not exist (and *missing\_ok* is False).
- **ProviderError** – On error copying file content.

**Return type** Generator[Optional[Path], None, None]

**exception** `craft_providers.ProviderError`(*brief: str, details: Optional[str] = None, resolution: Optional[str] = None*)

Bases: `Exception`

Unexpected error.

**Parameters**

- **brief** (`str`) – Brief description of error.
- **details** (`Optional[str]`) – Detailed information.
- **resolution** (`Optional[str]`) – Recommendation, if any.

**brief:** `str`

**details:** `Optional[str] = None`

**resolution:** `Optional[str] = None`

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### C

- `craft_providers`, 78
- `craft_providers.actions`, 20
- `craft_providers.actions.snap_installer`, 19
- `craft_providers.base`, 73
- `craft_providers.bases`, 25
- `craft_providers.bases.bulld`, 20
- `craft_providers.bases.errors`, 23
- `craft_providers.bases.instance_config`, 24
- `craft_providers.errors`, 75
- `craft_providers.executor`, 76
- `craft_providers.lxd`, 43
- `craft_providers.lxd.errors`, 28
- `craft_providers.lxd.installer`, 29
- `craft_providers.lxd.launcher`, 30
- `craft_providers.lxd.lxc`, 31
- `craft_providers.lxd.lxd`, 38
- `craft_providers.lxd.lxd_instance`, 39
- `craft_providers.lxd.project`, 42
- `craft_providers.lxd.remotes`, 43
- `craft_providers.multipass`, 63
- `craft_providers.multipass.errors`, 56
- `craft_providers.multipass.installer`, 57
- `craft_providers.multipass.multipass`, 57
- `craft_providers.multipass.multipass_instance`, 60
- `craft_providers.util`, 73
- `craft_providers.util.env_cmd`, 71
- `craft_providers.util.os_release`, 72
- `craft_providers.util.snap_cmd`, 72
- `craft_providers.util.temp_paths`, 73



## A

alias (craft\_providers.bases.buildd.BuilddBase attribute), 21

## B

Base (class in craft\_providers), 13, 78

Base (class in craft\_providers.base), 73

BaseCompatibilityError, 23, 25

BaseConfigurationError, 23, 26

BIONIC (craft\_providers.bases.buildd.BuilddBaseAlias attribute), 22

BIONIC (craft\_providers.bases.BuilddBaseAlias attribute), 28

brief (craft\_providers.actions.snap\_installer.SnapInstallationError attribute), 19

brief (craft\_providers.bases.BaseCompatibilityError attribute), 25

brief (craft\_providers.bases.BaseConfigurationError attribute), 26

brief (craft\_providers.bases.errors.BaseCompatibilityError attribute), 23

brief (craft\_providers.bases.errors.BaseConfigurationError attribute), 24

brief (craft\_providers.errors.ProviderError attribute), 75

brief (craft\_providers.lxd.errors.LXDError attribute), 29

brief (craft\_providers.lxd.errors.LXDInstallationError attribute), 29

brief (craft\_providers.lxd.LXDError attribute), 50

brief (craft\_providers.lxd.LXDInstallationError attribute), 51

brief (craft\_providers.multipass.errors.MultipassError attribute), 56

brief (craft\_providers.multipass.errors.MultipassInstallationError attribute), 56

brief (craft\_providers.multipass.MultipassError attribute), 67

brief (craft\_providers.multipass.MultipassInstallationError attribute), 67

brief (craft\_providers.ProviderError attribute), 82

BuilddBase (class in craft\_providers.bases), 15, 26

BuilddBase (class in craft\_providers.bases.buildd), 20

BuilddBaseAlias (class in craft\_providers.bases), 28

BuilddBaseAlias (class in craft\_providers.bases.buildd), 22

## C

channel (craft\_providers.bases.buildd.Snap attribute), 23

classic (craft\_providers.bases.buildd.Snap attribute), 23

compatibility\_tag (craft\_providers.Base attribute), 78

compatibility\_tag (craft\_providers.base.Base attribute), 74

compatibility\_tag (craft\_providers.bases.buildd.BuilddBase attribute), 21

compatibility\_tag (craft\_providers.bases.BuilddBase attribute), 26

compatibility\_tag (craft\_providers.bases.instance\_config.InstanceConfig attribute), 24

config\_device\_add\_disk() (craft\_providers.lxd.LXC method), 43

config\_device\_add\_disk() (craft\_providers.lxd.lxc.LXC method), 31

config\_device\_remove() (craft\_providers.lxd.LXC method), 43

config\_device\_remove() (craft\_providers.lxd.lxc.LXC method), 31

config\_device\_show() (craft\_providers.lxd.LXC method), 44

config\_device\_show() (craft\_providers.lxd.lxc.LXC method), 31

config\_set() (craft\_providers.lxd.LXC method), 44

config\_set() (craft\_providers.lxd.lxc.LXC method), 32

configure\_buildd\_image\_remote() (in module craft\_providers.lxd), 54

configure\_buildd\_image\_remote() (in module craft\_providers.lxd.remotes), 43

craft\_providers

module, 78

craft\_providers.actions

module, 20

[craft\\_providers.actions.snap\\_installer](#)  
     module, 19  
[craft\\_providers.base](#)  
     module, 73  
[craft\\_providers.bases](#)  
     module, 25  
[craft\\_providers.bases.bulld](#)  
     module, 20  
[craft\\_providers.bases.errors](#)  
     module, 23  
[craft\\_providers.bases.instance\\_config](#)  
     module, 24  
[craft\\_providers.errors](#)  
     module, 75  
[craft\\_providers.executor](#)  
     module, 76  
[craft\\_providers.lxd](#)  
     module, 43  
[craft\\_providers.lxd.errors](#)  
     module, 28  
[craft\\_providers.lxd.installer](#)  
     module, 29  
[craft\\_providers.lxd.launcher](#)  
     module, 30  
[craft\\_providers.lxd.lxc](#)  
     module, 31  
[craft\\_providers.lxd.lxd](#)  
     module, 38  
[craft\\_providers.lxd.lxd\\_instance](#)  
     module, 39  
[craft\\_providers.lxd.project](#)  
     module, 42  
[craft\\_providers.lxd.remotes](#)  
     module, 43  
[craft\\_providers.multipass](#)  
     module, 63  
[craft\\_providers.multipass.errors](#)  
     module, 56  
[craft\\_providers.multipass.installer](#)  
     module, 57  
[craft\\_providers.multipass.multipass](#)  
     module, 57  
[craft\\_providers.multipass.multipass\\_instance](#)  
     module, 60  
[craft\\_providers.util](#)  
     module, 73  
[craft\\_providers.util.env\\_cmd](#)  
     module, 71  
[craft\\_providers.util.os\\_release](#)  
     module, 72  
[craft\\_providers.util.snap\\_cmd](#)  
     module, 72  
[craft\\_providers.util.temp\\_paths](#)  
     module, 73

[create\\_with\\_default\\_profile\(\)](#) (in module [craft\\_providers.lxd.project](#)), 42

## D

[default\\_command\\_environment\(\)](#) (in module [craft\\_providers.bases.bulld](#)), 23  
[delete\(\)](#) ([craft\\_providers.Executor](#) method), 3, 80  
[delete\(\)](#) ([craft\\_providers.executor.Executor](#) method), 76  
[delete\(\)](#) ([craft\\_providers.lxd.LXC](#) method), 44  
[delete\(\)](#) ([craft\\_providers.lxd.lxc.LXC](#) method), 32  
[delete\(\)](#) ([craft\\_providers.lxd.lxd\\_instance.LXDInstance](#) method), 39  
[delete\(\)](#) ([craft\\_providers.lxd.LXDInstance](#) method), 5, 51  
[delete\(\)](#) ([craft\\_providers.multipass.Multipass](#) method), 63  
[delete\(\)](#) ([craft\\_providers.multipass.multipass.Multipass](#) method), 57  
[delete\(\)](#) ([craft\\_providers.multipass.multipass\\_instance.MultipassInstance](#) method), 60  
[delete\(\)](#) ([craft\\_providers.multipass.MultipassInstance](#) method), 8, 67  
[details](#) ([craft\\_providers.errors.ProviderError](#) attribute), 75  
[details](#) ([craft\\_providers.ProviderError](#) attribute), 82  
[details\\_from\\_called\\_process\\_error\(\)](#) (in module [craft\\_providers.errors](#)), 75  
[details\\_from\\_command\\_error\(\)](#) (in module [craft\\_providers.errors](#)), 75

## E

[ensure\\_lxd\\_is\\_ready\(\)](#) (in module [craft\\_providers.lxd](#)), 54  
[ensure\\_lxd\\_is\\_ready\(\)](#) (in module [craft\\_providers.lxd.installer](#)), 29  
[ensure\\_multipass\\_is\\_ready\(\)](#) (in module [craft\\_providers.multipass](#)), 70  
[exec\(\)](#) ([craft\\_providers.lxd.LXC](#) method), 44  
[exec\(\)](#) ([craft\\_providers.lxd.lxc.LXC](#) method), 32  
[exec\(\)](#) ([craft\\_providers.multipass.Multipass](#) method), 64  
[exec\(\)](#) ([craft\\_providers.multipass.multipass.Multipass](#) method), 57  
[execute\\_popen\(\)](#) ([craft\\_providers.Executor](#) method), 3, 80  
[execute\\_popen\(\)](#) ([craft\\_providers.executor.Executor](#) method), 76  
[execute\\_popen\(\)](#) ([craft\\_providers.lxd.lxd\\_instance.LXDInstance](#) method), 39  
[execute\\_popen\(\)](#) ([craft\\_providers.lxd.LXDInstance](#) method), 5, 51  
[execute\\_popen\(\)](#) ([craft\\_providers.multipass.multipass\\_instance.MultipassInstance](#) method), 60

`execute_popen()` (*craft\_providers.multipass.MultipassInstance* method), 8, 67

`execute_run()` (*craft\_providers.Executor* method), 3, 80

`execute_run()` (*craft\_providers.executor.Executor* method), 76

`execute_run()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 39

`execute_run()` (*craft\_providers.lxd.LXDInstance* method), 5, 51

`execute_run()` (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 61

`execute_run()` (*craft\_providers.multipass.MultipassInstance* method), 9, 67

**Executor** (class in *craft\_providers*), 3, 80

**Executor** (class in *craft\_providers.executor*), 76

`exists()` (*craft\_providers.Executor* method), 4, 80

`exists()` (*craft\_providers.executor.Executor* method), 77

`exists()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 40

`exists()` (*craft\_providers.lxd.LXDInstance* method), 6, 52

`exists()` (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 61

`exists()` (*craft\_providers.multipass.MultipassInstance* method), 9, 68

**F**

`file_pull()` (*craft\_providers.lxd.LXC* method), 45

`file_pull()` (*craft\_providers.lxd.lxc.LXC* method), 32

`file_push()` (*craft\_providers.lxd.LXC* method), 45

`file_push()` (*craft\_providers.lxd.lxc.LXC* method), 33

**FOCAL** (*craft\_providers.bases.buildd.BuilddBaseAlias* attribute), 22

**FOCAL** (*craft\_providers.bases.BuilddBaseAlias* attribute), 28

`formulate_command()` (in module *craft\_providers.util.env\_cmd*), 71

`formulate_local_install_command()` (in module *craft\_providers.util.snap\_cmd*), 72

`formulate_refresh_command()` (in module *craft\_providers.util.snap\_cmd*), 72

`formulate_remote_install_command()` (in module *craft\_providers.util.snap\_cmd*), 73

`formulate_remove_command()` (in module *craft\_providers.util.snap\_cmd*), 73

**G**

`get_command_environment()` (*craft\_providers.Base* method), 13, 78

`get_command_environment()` (*craft\_providers.base.Base* method), 74

`get_command_environment()` (*craft\_providers.bases.buildd.BuilddBase* method), 21

`get_command_environment()` (*craft\_providers.bases.BuilddBase* method), 15, 26

**H**

`has_image()` (*craft\_providers.lxd.LXC* method), 45

`has_image()` (*craft\_providers.lxd.lxc.LXC* method), 33

`home_temporary_directory()` (in module *craft\_providers.util.temp\_paths*), 73

`home_temporary_file()` (in module *craft\_providers.util.temp\_paths*), 73

**I**

`image_copy()` (*craft\_providers.lxd.LXC* method), 46

`image_copy()` (*craft\_providers.lxd.lxc.LXC* method), 33

`image_delete()` (*craft\_providers.lxd.LXC* method), 46

`image_delete()` (*craft\_providers.lxd.lxc.LXC* method), 34

`image_list()` (*craft\_providers.lxd.LXC* method), 46

`image_list()` (*craft\_providers.lxd.lxc.LXC* method), 34

`info()` (*craft\_providers.lxd.LXC* method), 46

`info()` (*craft\_providers.lxd.lxc.LXC* method), 34

`info()` (*craft\_providers.multipass.Multipass* method), 64

`info()` (*craft\_providers.multipass.multipass.Multipass* method), 57

`init()` (*craft\_providers.lxd.LXD* method), 50

`init()` (*craft\_providers.lxd.lxd.LXD* method), 38

`inject_from_host()` (in module *craft\_providers.actions.snap\_installer*), 19

`install()` (in module *craft\_providers.lxd*), 54

`install()` (in module *craft\_providers.lxd.installer*), 29

`install()` (in module *craft\_providers.multipass*), 70

`install()` (in module *craft\_providers.multipass.installer*), 57

`install_from_store()` (in module *craft\_providers.actions.snap\_installer*), 19

**instance\_config\_class** (*craft\_providers.bases.buildd.BuilddBase* attribute), 21

**instance\_config\_class** (*craft\_providers.bases.BuilddBase* attribute), 15, 26

**instance\_config\_path** (*craft\_providers.bases.buildd.BuilddBase* attribute), 21

**instance\_config\_path** (*craft\_providers.bases.BuilddBase* attribute), 27

**InstanceConfiguration** (class in *craft\_providers.bases.instance\_config*), 24

- INTERACTIVE (*craft\_providers.lxd.lxc.StdinType* attribute), 37
- is\_initialized() (in module *craft\_providers.lxd*), 55
- is\_initialized() (in module *craft\_providers.lxd.installer*), 29
- is\_installed() (in module *craft\_providers.lxd*), 55
- is\_installed() (in module *craft\_providers.lxd.installer*), 30
- is\_installed() (in module *craft\_providers.multipass*), 70
- is\_installed() (in module *craft\_providers.multipass.installer*), 57
- is\_mounted() (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 40
- is\_mounted() (*craft\_providers.lxd.LXDInstance* method), 6, 52
- is\_mounted() (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 61
- is\_mounted() (*craft\_providers.multipass.MultipassInstance* method), 9, 68
- is\_running() (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 40
- is\_running() (*craft\_providers.lxd.LXDInstance* method), 6, 52
- is\_running() (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 61
- is\_running() (*craft\_providers.multipass.MultipassInstance* method), 9, 68
- is\_supported\_version() (*craft\_providers.lxd.LXD* method), 50
- is\_supported\_version() (*craft\_providers.lxd.lxd.LXD* method), 38
- is\_supported\_version() (*craft\_providers.multipass.Multipass* method), 64
- is\_supported\_version() (*craft\_providers.multipass.multipass.Multipass* method), 58
- is\_user\_permitted() (in module *craft\_providers.lxd*), 55
- is\_user\_permitted() (in module *craft\_providers.lxd.installer*), 30
- J**
- JAMMY (*craft\_providers.bases.buildd.BuilddBaseAlias* attribute), 22
- JAMMY (*craft\_providers.bases.BuilddBaseAlias* attribute), 28
- L**
- launch() (*craft\_providers.lxd.LXC* method), 47
- launch() (*craft\_providers.lxd.lxc.LXC* method), 34
- launch() (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 40
- launch() (*craft\_providers.lxd.LXDInstance* method), 6, 52
- launch() (*craft\_providers.multipass.Multipass* method), 64
- launch() (*craft\_providers.multipass.multipass.Multipass* method), 58
- launch() (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 61
- launch() (*craft\_providers.multipass.MultipassInstance* method), 10, 68
- launch() (in module *craft\_providers.lxd*), 55
- launch() (in module *craft\_providers.lxd.launcher*), 30
- launch() (in module *craft\_providers.multipass*), 70
- list() (*craft\_providers.lxd.LXC* method), 47
- list() (*craft\_providers.lxd.lxc.LXC* method), 35
- list() (*craft\_providers.multipass.Multipass* method), 65
- list() (*craft\_providers.multipass.multipass.Multipass* method), 58
- list\_names() (*craft\_providers.lxd.LXC* method), 47
- list\_names() (*craft\_providers.lxd.lxc.LXC* method), 35
- load() (*craft\_providers.bases.instance\_config.InstanceConfiguration* class method), 24
- load\_yaml() (in module *craft\_providers.lxd.lxc*), 37
- LXC**
- LXC* (*craft\_providers.lxd*), 43
- LXC (class in *craft\_providers.lxd.lxc*), 31
- LXD**
- LXD (class in *craft\_providers.lxd*), 49
- LXD (class in *craft\_providers.lxd.lxd*), 38
- LXDError, 28, 50
- LXDInstallationError, 29, 50
- LXDInstance (class in *craft\_providers.lxd*), 5, 51
- LXDInstance (class in *craft\_providers.lxd.lxd\_instance*), 39
- M**
- marshal() (*craft\_providers.bases.instance\_config.InstanceConfiguration* method), 24
- minimum\_required\_version (*craft\_providers.lxd.LXD* attribute), 50
- minimum\_required\_version (*craft\_providers.lxd.lxd.LXD* attribute), 38
- minimum\_required\_version (*craft\_providers.multipass.Multipass* attribute), 65
- minimum\_required\_version (*craft\_providers.multipass.multipass.Multipass* attribute), 58
- module
- craft\_providers*, 78
  - craft\_providers.actions*, 20
  - craft\_providers.actions.snap\_installer*, 19
  - craft\_providers.base*, 73
  - craft\_providers.bases*, 25



craft\_providers.bases.builddd, 20  
 craft\_providers.bases.errors, 23  
 craft\_providers.bases.instance\_config, 24  
 craft\_providers.errors, 75  
 craft\_providers.executor, 76  
 craft\_providers.lxd, 43  
 craft\_providers.lxd.errors, 28  
 craft\_providers.lxd.installer, 29  
 craft\_providers.lxd.launcher, 30  
 craft\_providers.lxd.lxc, 31  
 craft\_providers.lxd.lxd, 38  
 craft\_providers.lxd.lxd\_instance, 39  
 craft\_providers.lxd.project, 42  
 craft\_providers.lxd.remotes, 43  
 craft\_providers.multipass, 63  
 craft\_providers.multipass.errors, 56  
 craft\_providers.multipass.installer, 57  
 craft\_providers.multipass.multipass, 57  
 craft\_providers.multipass.multipass\_instance, 60  
 craft\_providers.util, 73  
 craft\_providers.util.env\_cmd, 71  
 craft\_providers.util.os\_release, 72  
 craft\_providers.util.snap\_cmd, 72  
 craft\_providers.util.temp\_paths, 73  
 mount() (craft\_providers.lxd.lxd\_instance.LXDInstance method), 40  
 mount() (craft\_providers.lxd.LXDInstance method), 7, 52  
 mount() (craft\_providers.multipass.Multipass method), 65  
 mount() (craft\_providers.multipass.multipass.Multipass method), 58  
 mount() (craft\_providers.multipass.multipass\_instance.MultipassInstance method), 62  
 mount() (craft\_providers.multipass.MultipassInstance method), 10, 69  
 Multipass (class in craft\_providers.multipass), 63  
 Multipass (class in craft\_providers.multipass.multipass), 57  
 MultipassError, 56, 66  
 MultipassInstallationError, 56, 67  
 MultipassInstance (class in craft\_providers.multipass), 8, 67  
 MultipassInstance (class in craft\_providers.multipass.multipass\_instance), 60  
  
**N**  
 name (craft\_providers.bases.builddd.Snap attribute), 23  
 NULL (craft\_providers.lxd.lxc.StdinType attribute), 37  
  
**P**  
 parse\_os\_release() (in module
   
 craft\_providers.util.os\_release), 72  
 profile\_edit() (craft\_providers.lxd.LXC method), 47  
 profile\_edit() (craft\_providers.lxd.lxc.LXC method), 35  
 profile\_show() (craft\_providers.lxd.LXC method), 48  
 profile\_show() (craft\_providers.lxd.lxc.LXC method), 35  
 project\_create() (craft\_providers.lxd.LXC method), 48  
 project\_create() (craft\_providers.lxd.lxc.LXC method), 36  
 project\_delete() (craft\_providers.lxd.LXC method), 48  
 project\_delete() (craft\_providers.lxd.lxc.LXC method), 36  
 project\_list() (craft\_providers.lxd.LXC method), 48  
 project\_list() (craft\_providers.lxd.lxc.LXC method), 36  
 ProviderError, 75, 81  
 publish() (craft\_providers.lxd.LXC method), 48  
 publish() (craft\_providers.lxd.lxc.LXC method), 36  
 pull\_file() (craft\_providers.Executor method), 4, 80  
 pull\_file() (craft\_providers.executor.Executor method), 77  
 pull\_file() (craft\_providers.lxd.lxd\_instance.LXDInstance method), 41  
 pull\_file() (craft\_providers.lxd.LXDInstance method), 7, 53  
 pull\_file() (craft\_providers.multipass.multipass\_instance.MultipassInstance method), 62  
 pull\_file() (craft\_providers.multipass.MultipassInstance method), 10, 69  
 purge() (in module craft\_providers.lxd.project), 42  
 push\_file() (craft\_providers.Executor method), 4, 81  
 push\_file() (craft\_providers.executor.Executor method), 77  
 push\_file() (craft\_providers.lxd.lxd\_instance.LXDInstance method), 41  
 push\_file() (craft\_providers.lxd.LXDInstance method), 7, 53  
 push\_file() (craft\_providers.multipass.multipass\_instance.MultipassInstance method), 62  
 push\_file() (craft\_providers.multipass.MultipassInstance method), 10, 69  
 push\_file\_io() (craft\_providers.Executor method), 4, 81  
 push\_file\_io() (craft\_providers.executor.Executor method), 77  
 push\_file\_io() (craft\_providers.lxd.lxd\_instance.LXDInstance method), 41  
 push\_file\_io() (craft\_providers.lxd.LXDInstance method), 7, 53  
 push\_file\_io() (craft\_providers.multipass.multipass\_instance.MultipassInstance method), 63

`push_file_io()` (*craft\_providers.multipass.MultipassInstance* method), 11, 69

## R

`remote_add()` (*craft\_providers.lxd.LXC* method), 49  
`remote_add()` (*craft\_providers.lxd.lxc.LXC* method), 36  
`remote_list()` (*craft\_providers.lxd.LXC* method), 49  
`remote_list()` (*craft\_providers.lxd.lxc.LXC* method), 37  
`resolution` (*craft\_providers.errors.ProviderError* attribute), 75  
`resolution` (*craft\_providers.ProviderError* attribute), 82

## S

`save()` (*craft\_providers.bases.instance\_config.InstanceConfiguration* method), 24  
`setup()` (*craft\_providers.Base* method), 13, 78  
`setup()` (*craft\_providers.base.Base* method), 74  
`setup()` (*craft\_providers.bases.buildd.BuilddBase* method), 21  
`setup()` (*craft\_providers.bases.BuilddBase* method), 15, 27  
`Snap` (class in *craft\_providers.bases.buildd*), 23  
`SnapInstallationError`, 19  
`snaps` (*craft\_providers.bases.instance\_config.InstanceConfiguration* attribute), 25  
`start()` (*craft\_providers.lxd.LXC* method), 49  
`start()` (*craft\_providers.lxd.lxc.LXC* method), 37  
`start()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 41  
`start()` (*craft\_providers.lxd.LXDInstance* method), 8, 53  
`start()` (*craft\_providers.multipass.Multipass* method), 65  
`start()` (*craft\_providers.multipass.multipass.Multipass* method), 58  
`start()` (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 63  
`start()` (*craft\_providers.multipass.MultipassInstance* method), 11, 70  
`StdinType` (class in *craft\_providers.lxd.lxc*), 37  
`stop()` (*craft\_providers.lxd.LXC* method), 49  
`stop()` (*craft\_providers.lxd.lxc.LXC* method), 37  
`stop()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 41  
`stop()` (*craft\_providers.lxd.LXDInstance* method), 8, 54  
`stop()` (*craft\_providers.multipass.Multipass* method), 65  
`stop()` (*craft\_providers.multipass.multipass.Multipass* method), 59  
`stop()` (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 63

`stop()` (*craft\_providers.multipass.MultipassInstance* method), 11, 70  
`supports_mount()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 42  
`supports_mount()` (*craft\_providers.lxd.LXDInstance* method), 8, 54

## T

`temporarily_pull_file()` (*craft\_providers.Executor* method), 4, 81  
`temporarily_pull_file()` (*craft\_providers.executor.Executor* method), 77  
`transfer()` (*craft\_providers.multipass.Multipass* method), 65  
`transfer()` (*craft\_providers.multipass.multipass.Multipass* method), 59  
`transfer_destination_io()` (*craft\_providers.multipass.Multipass* method), 65  
`transfer_destination_io()` (*craft\_providers.multipass.multipass.Multipass* method), 59  
`transfer_source_io()` (*craft\_providers.multipass.Multipass* method), 66  
`transfer_source_io()` (*craft\_providers.multipass.multipass.Multipass* method), 59

## U

`umount()` (*craft\_providers.multipass.Multipass* method), 66  
`umount()` (*craft\_providers.multipass.multipass.Multipass* method), 60  
`unmarshal()` (*craft\_providers.bases.instance\_config.InstanceConfiguration* class method), 25  
`unmount()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 42  
`unmount()` (*craft\_providers.lxd.LXDInstance* method), 8, 54  
`unmount()` (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 63  
`unmount()` (*craft\_providers.multipass.MultipassInstance* method), 11, 70  
`unmount_all()` (*craft\_providers.lxd.lxd\_instance.LXDInstance* method), 42  
`unmount_all()` (*craft\_providers.lxd.LXDInstance* method), 8, 54  
`unmount_all()` (*craft\_providers.multipass.multipass\_instance.MultipassInstance* method), 63  
`unmount_all()` (*craft\_providers.multipass.MultipassInstance* method), 11, 70  
`update()` (*craft\_providers.bases.instance\_config.InstanceConfiguration* class method), 25



`update_nested_dictionaries()` (in module `craft_providers.bases.instance_config`), 25

## V

`validate_channel()` (`craft_providers.bases.buildd.Snap` class method), 23

`version()` (`craft_providers.lxd.LXD` method), 50

`version()` (`craft_providers.lxd.lxd.LXD` method), 38

`version()` (`craft_providers.multipass.Multipass` method), 66

`version()` (`craft_providers.multipass.multipass.Multipass` method), 60

## W

`wait_ready()` (`craft_providers.lxd.LXD` method), 50

`wait_ready()` (`craft_providers.lxd.lxd.LXD` method), 38

`wait_until_ready()` (`craft_providers.Base` method), 14, 79

`wait_until_ready()` (`craft_providers.base.Base` method), 74

`wait_until_ready()` (`craft_providers.bases.buildd.BuilddBase` method), 21

`wait_until_ready()` (`craft_providers.bases.BuilddBase` method), 16, 27

`wait_until_ready()` (`craft_providers.multipass.Multipass` method), 66

`wait_until_ready()` (`craft_providers.multipass.multipass.Multipass` method), 60

`warmup()` (`craft_providers.Base` method), 14, 79

`warmup()` (`craft_providers.base.Base` method), 75

`warmup()` (`craft_providers.bases.buildd.BuilddBase` method), 22

`warmup()` (`craft_providers.bases.BuilddBase` method), 16, 28

## X

`XENIAL` (`craft_providers.bases.buildd.BuilddBaseAlias` attribute), 22

`XENIAL` (`craft_providers.bases.BuilddBaseAlias` attribute), 28