Craft Providers

Release 1.6.2

Canonical Ltd.

Jan 11, 2023

PUBLIC APIS:

Here you will find all of the provider documentation…

CHAPTER

ONE

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 is_running()
Check if instance is running.

Return type bool

Returns True if instance is running.

abstract mount(*, *host_source*, *target*)
Mount host source directory to target mount point.

Parameters

- **host_source** (Path) –
- **target** (Path) –

Return type None

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.

The destination file is overwritten if it exists.

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.

The temporary file is stored in the home directory where Multipass has access.

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*)
Mount host source directory to target mount point.

Checks first to see if already mounted. The source will be mounted as a disk named "disk-{target.as_posix()}".

Parameters

- **host_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

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.

The destination file is overwritten if it exists.

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 a process in the 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.

The command is run as root via sudo. Running as root may be required even when the command itself does not require root permissions, because the instance's working directory may be a directory that the default *ubuntu* user does not have access to.

Parameters

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

Return type Popen

Returns Popen instance.

execute_run(*command*, *, *cwd=None*, *env=None*, ***kwargs*)
Execute a command in the instance using subprocess.run().

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

The command is run as root via sudo. Running as root may be required even when the command itself does not require root permissions, because the instance's working directory may be a directory that the default *ubuntu* user does not have access to.

Parameters

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

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.

The destination file is overwritten if it exists.

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

CHAPTER

TWO

BASES

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}.{apprevision}" 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}.{apprevision}" 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

CHAPTER

THREE

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.**get_host_snap_info**(*snap_name*)

Get info about a snap installed on the host.

Parameters **snap_name** (str) –

Return type Dict[str, Any]

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

Inject snap from host snap.

Parameters

- **executor** (*Executor*) – Executor for target
- **snap_name** (str) – Name of snap to inject
- **classic** (bool) – Install in classic mode

Raises *SnapInstallationError* – on failure to inject snap

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}.{apprevision}" 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

exception craft_providers.bases.errors.**NetworkError**

Bases: *craft_providers.errors.ProviderError*

Network error when configuring the base.

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"

  charmcraft: 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.**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}.{apprevision}" 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]) –

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.**BuilddBaseAlias**(*value*)

Bases: enum.Enum

Mappings for supported buildd 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=None*, *use_base_instance=False*, *project='default'*, *remote='local'*, *lxc=<craft_providers.lxd.lxc.LXC object>*)
Create, start, and configure an instance.

On the first run of an application, an instance will be launched from an image (i.e. an image from https://cloud-images.ubuntu.com). The instance is setup according to the Base configuration passed to this function.

After setup, a copy of this instance is saved (or cached) as a 'base instance'. This is done to reduce setup time on subsequent runs. When the application requests a new instance on a subsequent run, the base instance will be copied to create the new instance. This instance is run through a small subset of the setup, which is referred to as 'warmup'.

To keep build environments clean, consistent, and up-to-date, any base instance older than 3 months (90 days) is deleted and recreated.

Parameters

- **name** (str) – Name of instance.
- **base_configuration** (*Base*) – Base configuration to apply to the 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) – If true and the existing instance is incompatible, then the

instance will be deleted and rebuilt. If false and the existing instance is incompatible, then a BaseCompatibilityError is raised. :type auto_create_project: bool :param auto_create_project: Automatically create LXD project, if needed. :type ephemeral: bool :param ephemeral: After the instance is stopped, delete it. Non-ephemeral instances cannot be converted to ephemeral instances, so if the instance already exists, it will be deleted, then recreated as an ephemeral instance. :type map_user_uid: bool :param map_user_uid: Map host uid/gid to instance's root uid/gid. :type uid: Optional[int] :param uid: The uid to be mapped, if map_user_id is enabled. :type use_base_instance: Optional[bool] :param use_base_instance: Use the base instance mechanisms to reduce setup time. :type use_snapshots: Optional[bool] :param use_snapshots: Deprecated parameter replaced by *use_base_instance*. :type project: str :param project: LXD project to create instance in. :type remote: str :param remote: LXD remote to create instance on. :type lxc: *LXC* :param lxc: LXC client.

Return type *LXDInstance*

Returns LXD instance.

Raises

- ***BaseConfigurationError*** – on unexpected error configuration base.
- ***BaseCompatibilityError*** – if instance is incompatible with the base.
- ***LXDError*** – on unexpected LXD error.
- ***ProviderError*** – if name of instance collides with base instance name.

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

copy(*, *source_remote='local'*, *source_instance_name*, *destination_remote='local'*, *destination_instance_name*, *project='default'*)
Copy instances within or in between LXD servers.

Calls *lxc copy <source_remote>:<source_instance_name> <destination_remote>: destination_instance_name*. A running instance can be copied but the manpages state "This may cause data corruption or data loss depending on the used filesystem and applications. Use with care."

Parameters

- **source_remote** (str) – Name of source LXD remote.
- **source_instance_name** (str) – Name of instance to copy from.
- **destination_remote** (str) – Name of remote LXD destination.

- **destination_instance_name** (str) – Name of instance to copy to.
- **project** (str) – Name of LXD project.

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 >= 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*)
Mount host source directory to target mount point.

Checks first to see if already mounted. The source will be mounted as a disk named "disk-{target.as_posix()}".

Parameters

- **host_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

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.

The destination file is overwritten if it exists.

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.lxd_provider module

LXD Provider class.

class craft_providers.lxd.lxd_provider.**LXDProvider**(*, *lxc*=<*craft_providers.lxd.lxc.LXC object*>, *lxd_project*='default', *lxd_remote*='local')

Bases: *craft_providers.provider.Provider*

LXD build environment provider.

This class is not stable and is likely to change. This class will be stable and recommended for use in the release of craft-providers 2.0.

Parameters

- **lxc** (*LXC*) – Optional lxc client to use.
- **lxd_project** (str) – LXD project to use (default is default).
- **lxd_remote** (str) – LXD remote to use (default is local).

create_environment(*, *instance_name*)
Create a bare environment for specified base.

No initializing, launching, or cleaning up of the environment occurs.

Parameters **instance_name** (str) – Name of the instance.

Return type *Executor*

classmethod ensure_provider_is_available()
Ensure provider is available and ready, installing if required.

Raises

- *LXDInstallationError* – if LXD cannot be installed
- *LXDError* – if provider is not available

Return type None

classmethod is_provider_installed()
Check if provider is installed.

Return type bool

Returns True if installed.

launched_environment(*, *project_name*, *project_path*, *base_configuration*, *build_base*, *instance_name*)
Configure and launch environment for specified base.

When this method loses context, all directories are unmounted and the environment is stopped. For more control of environment setup and teardown, use *create_environment()* instead.

Parameters

- **project_name** (str) – Name of project.
- **project_path** (Path) – Path to project.

- **base_configuration** (*Base*) – Base configuration to apply to instance.
- **build_base** (str) – Base to build from.
- **instance_name** (str) – Name of the instance to launch.

Raises *LXDError* – if instance cannot be configured and launched

Return type Generator[*Executor*, None, 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_buildd_image_remote**(*lxc=<craft_providers.lxd.lxc.LXC object>*)

Configure buildd 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

copy(*, *source_remote='local'*, *source_instance_name*, *destination_remote='local'*, *destination_instance_name*, *project='default'*)

Copy instances within or in between LXD servers.

Calls *lxc copy <source_remote>:<source_instance_name> <destination_remote>: destination_instance_name*. A running instance can be copied but the manpages state "This may cause data corruption or data loss depending on the used filesystem and applications. Use with care."

Parameters

- **source_remote** (str) – Name of source LXD remote.
- **source_instance_name** (str) – Name of instance to copy from.
- **destination_remote** (str) – Name of remote LXD destination.
- **destination_instance_name** (str) – Name of instance to copy to.
- **project** (str) – Name of LXD project.

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 >= 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*)
Mount host source directory to target mount point.

Checks first to see if already mounted. The source will be mounted as a disk named "disk-{target.as_posix()}".

Parameters

- **host_source** (Path) – Host path to mount.
- **target** (PurePath) – Instance path to mount to.

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.

The destination file is overwritten if it exists.

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

class craft_providers.lxd.**LXDProvider**(*, *lxc=<craft_providers.lxd.lxc.LXC object>*, *lxd_project='default'*, *lxd_remote='local'*)
Bases: *craft_providers.provider.Provider*

LXD build environment provider.

This class is not stable and is likely to change. This class will be stable and recommended for use in the release of craft-providers 2.0.

Parameters

- **lxc** (*LXC*) – Optional lxc client to use.
- **lxd_project** (str) – LXD project to use (default is default).
- **lxd_remote** (str) – LXD remote to use (default is local).

create_environment(*, *instance_name*)
Create a bare environment for specified base.

No initializing, launching, or cleaning up of the environment occurs.

Parameters **instance_name** (str) – Name of the instance.

Return type *Executor*

classmethod ensure_provider_is_available()
Ensure provider is available and ready, installing if required.

Raises

- ***LXDInstallationError*** – if LXD cannot be installed
- ***LXDError*** – if provider is not available

Return type None

classmethod is_provider_installed()
Check if provider is installed.

Return type bool

Returns True if installed.

launched_environment(*, *project_name*, *project_path*, *base_configuration*, *build_base*, *instance_name*)
Configure and launch environment for specified base.

When this method loses context, all directories are unmounted and the environment is stopped. For more control of environment setup and teardown, use *create_environment()* instead.

Parameters

- **project_name** (str) – Name of project.
- **project_path** (Path) – Path to project.
- **base_configuration** (*Base*) – Base configuration to apply to instance.
- **build_base** (str) – Base to build from.
- **instance_name** (str) – Name of the instance to launch.

Raises ***LXDError*** – if instance cannot be configured and launched

Return type Generator[*Executor*, None, None]

craft_providers.lxd.**configure_buildd_image_remote**(*lxc=<craft_providers.lxd.lxc.LXC object>*)
Configure buildd 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=None*, *use_base_instance=False*, *project='default'*, *remote='local'*, *lxc=<craft_providers.lxd.lxc.LXC object>*)
Create, start, and configure an instance.

On the first run of an application, an instance will be launched from an image (i.e. an image from https://cloud-images.ubuntu.com). The instance is setup according to the Base configuration passed to this function.

After setup, a copy of this instance is saved (or cached) as a 'base instance'. This is done to reduce setup time on subsequent runs. When the application requests a new instance on a subsequent run, the base instance will be copied to create the new instance. This instance is run through a small subset of the setup, which is referred to as 'warmup'.

To keep build environments clean, consistent, and up-to-date, any base instance older than 3 months (90 days) is deleted and recreated.

Parameters

- **name** (str) – Name of instance.
- **base_configuration** (*Base*) – Base configuration to apply to the 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) – If true and the existing instance is incompatible, then the

instance will be deleted and rebuilt. If false and the existing instance is incompatible, then a BaseCompatibilityError is raised. :type auto_create_project: bool :param auto_create_project: Automatically create LXD project, if needed. :type ephemeral: bool :param ephemeral: After the instance is stopped, delete it. Non-ephemeral instances cannot be converted to ephemeral instances, so if the instance already exists, it will be deleted, then recreated as an ephemeral instance. :type map_user_uid: bool :param map_user_uid: Map host uid/gid to instance's root uid/gid. :type uid: Optional[int] :param uid: The uid to be mapped, if map_user_id is enabled. :type use_base_instance: Optional[bool] :param use_base_instance: Use the base instance mechanisms to reduce setup time. :type use_snapshots: Optional[bool] :param use_snapshots: Deprecated parameter replaced by *use_base_instance*. :type project: str :param project: LXD project to create instance in. :type remote: str :param remote: LXD remote to create instance on. :type lxc: *LXC* :param lxc: LXC client.

Return type *LXDInstance*

Returns LXD instance.

Raises

- ***BaseConfigurationError*** – on unexpected error configuration base.
- ***BaseCompatibilityError*** – if instance is incompatible with the base.
- ***LXDError*** – on unexpected LXD error.
- ***ProviderError*** – if name of instance collides with base instance name.

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.

The working directory the command is executed from inside the instance depends on the host's cwd. From the Multipass documentation: "In case we are executing the alias on the host from a directory which is mounted on the instance, the command will be executed on the instance from there. If the working directory is not mounted on the instance, the command will be executed on the default directory on the instance."

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.

Multipass transfer uses sftp. By default, only the user *ubuntu* can transfer files. Therefore, the path inside the instance should be accessible by the *ubuntu* user.

By default, Multipass only has access to the host's home directory. The host's path should be inside the home directory.

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 a process in the 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.

The command is run as root via sudo. Running as root may be required even when the command itself does not require root permissions, because the instance's working directory may be a directory that the default *ubuntu* user does not have access to.

Parameters

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

Return type Popen

Returns Popen instance.

execute_run(*command*, *, *cwd=None*, *env=None*, ***kwargs*)
Execute a command in the instance using subprocess.run().

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

The command is run as root via sudo. Running as root may be required even when the command itself does not require root permissions, because the instance's working directory may be a directory that the default *ubuntu* user does not have access to.

Parameters

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

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.

The destination file is overwritten if it exists.

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.multipass_provider module

Multipass Provider class.

class craft_providers.multipass.multipass_provider.**MultipassProvider**(*instance=<craft_providers.multipass.multipa object>*)

Bases: *craft_providers.provider.Provider*

Multipass build environment provider.

This class is not stable and is likely to change. This class will be stable and recommended for use in the release of craft-providers 2.0.

Parameters

- **multipass** – Optional Multipass client to use.
- **instance** (*Multipass*) –

create_environment(*, *instance_name*)
Create a bare environment for specified base.

No initializing, launching, or cleaning up of the environment occurs.

Parameters

- **name** – Name of the instance.
- **instance_name** (str) –

Return type *Executor*

classmethod ensure_provider_is_available()
Ensure provider is available, prompting the user to install it if required.

Raises *MultipassError* – if provider is not available.

Return type None

classmethod is_provider_installed()
Check if provider is installed.

Return type bool

Returns True if installed.

launched_environment(*, *project_name*, *project_path*, *base_configuration*, *build_base*, *instance_name*)
Configure and launch environment for specified base.

When this method loses context, all directories are unmounted and the environment is stopped. For more control of environment setup and teardown, use *create_environment()* instead.

Parameters

- **project_name** (str) – Name of the project.
- **project_path** (Path) – Path to project.
- **base_configuration** (*Base*) – Base configuration to apply to instance.
- **build_base** (str) – Base to build from.
- **instance_name** (str) – Name of the instance to launch.

Return type Generator[*Executor*, None, 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.

The working directory the command is executed from inside the instance depends on the host's cwd. From the Multipass documentation: "In case we are executing the alias on the host from a directory which is mounted on the instance, the command will be executed on the instance from there. If the working directory is not mounted on the instance, the command will be executed on the default directory on the instance."

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.

Multipass transfer uses sftp. By default, only the user *ubuntu* can transfer files. Therefore, the path inside the instance should be accessible by the *ubuntu* user.

By default, Multipass only has access to the host's home directory. The host's path should be inside the home directory.

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 a process in the 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.

The command is run as root via sudo. Running as root may be required even when the command itself does not require root permissions, because the instance's working directory may be a directory that the default *ubuntu* user does not have access to.

Parameters

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

Return type Popen

Returns Popen instance.

execute_run(*command*, *, *cwd=None*, *env=None*, ***kwargs*)
Execute a command in the instance using subprocess.run().

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

The command is run as root via sudo. Running as root may be required even when the command itself does not require root permissions, because the instance's working directory may be a directory that the default *ubuntu* user does not have access to.

Parameters

- **command** (List[str]) – Command to execute.
- **cwd** (Optional[Path]) – working directory to execute the command

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

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.

The destination file is overwritten if it exists.

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

class craft_providers.multipass.**MultipassProvider**(*instance=<craft_providers.multipass.multipass.Multipass object>*)
Bases: *craft_providers.provider.Provider*

Multipass build environment provider.

This class is not stable and is likely to change. This class will be stable and recommended for use in the release of craft-providers 2.0.

Parameters

- **multipass** – Optional Multipass client to use.
- **instance** (*Multipass*) –

create_environment(*, *instance_name*)
Create a bare environment for specified base.

No initializing, launching, or cleaning up of the environment occurs.

Parameters

- **name** – Name of the instance.
- **instance_name** (str) –

Return type *Executor*

classmethod ensure_provider_is_available()
Ensure provider is available, prompting the user to install it if required.

Raises *MultipassError* – if provider is not available.

Return type None

classmethod is_provider_installed()
Check if provider is installed.

Return type bool

Returns True if installed.

launched_environment(*, *project_name*, *project_path*, *base_configuration*, *build_base*, *instance_name*)
Configure and launch environment for specified base.

When this method loses context, all directories are unmounted and the environment is stopped. For more control of environment setup and teardown, use *create_environment()* instead.

Parameters

- **project_name** (str) – Name of the project.
- **project_path** (Path) – Path to project.
- **base_configuration** (*Base*) – Base configuration to apply to instance.
- **build_base** (str) – Base to build from.
- **instance_name** (str) – Name of the instance to launch.

Return type Generator[*Executor*, None, 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

Helpers for snap command.

craft_providers.util.snap_cmd.**formulate_ack_command**(*snap_assert_path*)
Formulate snap ack command to add assertions.

Return type List[str]

Returns List of ack command parts.

Parameters **snap_assert_path** (Path) –

craft_providers.util.snap_cmd.**formulate_known_command**(*query*)
Formulate snap known command to retrieve assertions.

Return type List[str]

Returns List of ack command parts.

Parameters **query** (List[str]) –

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_pack_command**(*snap_name*, *output_file_path*)
Formulate the command to pack a snap from a directory.

Parameters

- **snap_name** (str) – The name of the channel.
- **output_file_path** – File path to the packed snap.

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 is_running()

Check if instance is running.

Return type bool

Returns True if instance is running.

abstract mount(*, *host_source*, *target*)
Mount host source directory to target mount point.

Parameters

- **host_source** (Path) –
- **target** (Path) –

Return type None

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.

The destination file is overwritten if it exists.

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.

The temporary file is stored in the home directory where Multipass has access.

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]

3.2.4 craft_providers.provider module

Abstract base class for Providers.

The Provider, LXDProvider, and MultipassProvider classes are not stable and are likely to change. These classes will be stable and recommended for use in the release of craft-providers 2.0.

class craft_providers.provider.**Provider**
Bases: abc.ABC

Build environment provider.

clean_project_environments(*, *instance_name*)
Clean the provider environment.

Parameters **instance_name** (str) – name of the instance to clean

Return type None

abstract create_environment(*, *instance_name*)
Create a bare environment for specified base.

No initializing, launching, or cleaning up of the environment occurs.

Parameters **instance_name** (str) – name of the instance to create

Return type *Executor*

abstract classmethod ensure_provider_is_available()
Ensure provider is available, prompting the user to install it if required.

Raises ***ProviderError*** – if provider is not available.

Return type None

abstract classmethod is_provider_installed()
Check if provider is installed.

Return type bool

Returns True if installed.

abstract launched_environment(*, *project_name*, *project_path*, *base_configuration*, *build_base*, *instance_name*)

Configure and launch environment for specified base.

When this method loses context, all directories are unmounted and the environment is stopped. For more control of environment setup and teardown, use *create_environment()* instead.

Parameters

- **project_name** (str) – Name of project.
- **project_path** (Path) – Path to project.
- **base_configuration** (*Base*) – Base configuration to apply to instance.
- **build_base** (str) – Base to build from.
- **instance_name** (str) – Name of the instance to launch.

Return type Generator[*Executor*, 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}.{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

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 is_running()

Check if instance is running.

Return type bool

Returns True if instance is running.

abstract mount(*, *host_source*, *target*)

Mount host source directory to target mount point.

Parameters

- **host_source** (Path) –
- **target** (Path) –

Return type None

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.

The destination file is overwritten if it exists.

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.

The temporary file is stored in the home directory where Multipass has access.

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]

class craft_providers.**Provider**

Bases: abc.ABC

Build environment provider.

clean_project_environments(*, *instance_name*)

Clean the provider environment.

Parameters **instance_name** (str) – name of the instance to clean

Return type None

abstract create_environment(*, *instance_name*)

Create a bare environment for specified base.

No initializing, launching, or cleaning up of the environment occurs.

Parameters **instance_name** (str) – name of the instance to create

Return type *Executor*

abstract classmethod ensure_provider_is_available()

Ensure provider is available, prompting the user to install it if required.

Raises *ProviderError* – if provider is not available.

Return type None

abstract classmethod is_provider_installed()

Check if provider is installed.

Return type bool

Returns True if installed.

abstract launched_environment(*, *project_name*, *project_path*, *base_configuration*, *build_base*, *instance_name*)

Configure and launch environment for specified base.

When this method loses context, all directories are unmounted and the environment is stopped. For more control of environment setup and teardown, use *create_environment()* instead.

Parameters

- **project_name** (str) – Name of project.
- **project_path** (Path) – Path to project.
- **base_configuration** (*Base*) – Base configuration to apply to instance.

- **build_base** (str) – Base to build from.
- **instance_name** (str) – Name of the instance to launch.

Return type Generator[*Executor*, 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

CHAPTER

FOUR

EXPLANATIONS

4.1 Failure to properly execute commands that depend on network access

A common problem that can occur when running the setup or warmup of an instance is a failure associated with different processes (e.g. `apt`) not being able to access the network properly.

While this can happen because of a myriad of situations (network outage in the host, a malfunctioning proxy, etc), there is a very common case where Docker changes the *iptables* of the host in a particular way that conflicts with LXD instances.

If you're getting network errors in an LXD instance and have Docker installed, please refer to this section in the Linux Containers documentation for more information and ways to solve the situation.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

D

E

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W

X