Open Container Initiative Runtime Specification

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0.0-rc2	2016-11		T

Contents

The Open Container Initiative develops specifications for standards on Operating System process and application containers.¶

1 Use Cases

To provide context for users the following section gives example use cases for each part of the spec.¶

1.1 Application Bundle Builders

Application bundle builders can create a bundle directory that includes all of the files required for launching an application as a container. The bundle contains an OCI configuration where the builder can specify host-independent details such as which executable to launch and host-specific settings such as Section 6.3, Section 6.10, Section 6.7.2 and Section 6.7.5. Because the configuration includes host-specific settings, application bundle directories copied between two hosts may require configuration adjustments.¶

1.2 Hook Developers

Hook developers can extend the functionality of an OCI-compliant runtime by hooking into a container's lifecycle with an external application. Example use cases include sophisticated network configuration, volume garbage collection, etc.¶

1.3 Runtime Developers

Runtime developers can build runtime implementations that run OCI-compliant bundles and container configuration, containing low-level OS and host specific details, on a particular platform.¶

2 Definitions

In the specifications in the above table of contents, the keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997).¶

The keywords "unspecified", "undefined", and "implementation-defined" are to be interpreted as described in the rationale for the C99 standard.¶

An implementation is not compliant for a given CPU architecture if it fails to satisfy one or more of the MUST, REQUIRED, or SHALL requirements for the protocols it implements. An implementation is compliant for a given CPU architecture if it satisfies all the MUST, REQUIRED, and SHALL requirements for the protocols it implements.¶

bundle
: A directory structure that is written ahead of time, distributed, and used to seed the runtime for creating a container and launching a process within it.

configuration
: The config.json file in a bundle which defines the intended container and container process.

container
: An environment for executing processes with configurable isolation and resource limitations. For example, namespaces, resource limits, and mounts are all part of the container environment.

container namespace
: On Linux, a leaf in the namespace hierarchy in which the configured process executes.

JSON
: All configuration JSON MUST be encoded in UTF-8. JSON objects MUST NOT include duplicate names. The order of entries in JSON objects is not significant.

runtime
An implementation of this specification. It reads the configuration from a bundle, uses that information to create a container, launches a process inside the container, and performs other lifecycle actions.

runtime namespace
On Linux, a leaf in the namespace hierarchy from which the runtime process is executed. New container namespaces will be created as children of the runtime namespaces.

3 Protocols

Protocols defined by this specification are:¶

- Linux containers: Section 5, Section 5.2, Section 6, and Section 6.7.
- Solaris containers: Section 5, Section 6, and Section 6.8.
- Windows containers: Section 5, Section 6, and Section 6.9.

4 Filesystem Bundle

This section defines a format for encoding a container as a *filesystem bundle* - a set of files organized in a certain way, and containing all the necessary data and metadata for any compliant runtime to perform all standard operations against it. See also OS X application bundles for a similar use of the term *bundle*.¶

The definition of a bundle is only concerned with how a container, and its configuration data, are stored on a local filesystem so that it can be consumed by a compliant runtime.¶

A Standard Container bundle contains all the information needed to load and run a container. This MUST include the following artifacts:¶

1. `config.json`: contains configuration data. This REQUIRED file MUST reside in the root of the bundle directory and MUST be named `config.json`. See Section 6 for more details.
2. A directory representing the root filesystem of the container. While the name of this REQUIRED directory may be arbitrary, users should consider using a conventional name, such as `rootfs`. This directory MUST be referenced from within the `config.json` file.

While these artifacts MUST all be present in a single directory on the local filesystem, that directory itself is not part of the bundle. In other words, a tar archive of a *bundle* will have these artifacts at the root of the archive, not nested within a top-level directory.¶

5 Runtime and Lifecycle

5.1 Platform-independent runtime behavior

5.1.1 Scope of a Container

Barring access control concerns, the entity using a runtime to create a container MUST be able to use the operations defined in this specification against that same container. Whether other entities using the same, or other, instance of the runtime can see that container is out of scope of this specification.¶

State

The state of a container MUST include, at least, the following properties:¶

`ociVersion`
(string) is the OCI specification version used when creating the container.

`id`
(string) is the container's ID. This MUST be unique across all containers on this host. There is no requirement that it be unique across hosts.

`status`
(string) is the runtime state of the container. The value MAY be one of:

`created`
The container has been created but the user-specified code has not yet been executed.

`running`
The container has been created and the user-specified code is running.

`stopped`
The container has been created and the user-specified code has been executed but is no longer running.

Additional values MAY be defined by the runtime, however, they MUST be used to represent new runtime states not defined above.¶

`pid`
(int) is the ID of the container process, as seen by the host.

`bundlePath`
(string) is the absolute path to the container's bundle directory. This is provided so that consumers can find the container's configuration and root filesystem on the host.

`annotations`
(map) contains the list of annotations associated with the container. If no annotations were provided then this property MAY either be absent or an empty map.

When serialized in JSON, the format MUST adhere to the following pattern:¶

```
{
    "ociVersion": "0.2.0",
    "id": "oci-container1",
    "status": "running",
    "pid": 4422,
    "bundlePath": "/containers/redis",
    "annotations": {
        "myKey": "myValue"
    }
}
```

See Section 5.1.4 for information on retrieving the state of a container.¶

5.1.2 Lifecycle

The lifecycle describes the timeline of events that happen from when a container is created to when it ceases to exist.¶

1. OCI compliant runtime's create command is invoked with a reference to the location of the bundle and a unique identifier.
2. The container's runtime environment MUST be created according to the configuration in config.json. If the runtime is unable to create the environment specified in the configuration, it MUST generate an error. While the resources requested in the configuration MUST be created, the user-specified code (from Section 6.4) MUST NOT be run at this time. Any updates to the configuration after this step MUST NOT affect the container.

3. Once the container is created additional actions MAY be performed based on the features the runtime chooses to support. However, some actions might only be available based on the current state of the container (e.g. only available while it is started).
4. Runtime's start command is invoked with the unique identifier of the container. The runtime MUST run the user-specified code, as specified by Section 6.4.
5. The container's process is stopped. This MAY happen due to them erroring out, exiting, crashing or the runtime's kill operation being invoked.
6. Runtime's delete command is invoked with the unique identifier of the container. The container MUST be destroyed by undoing the steps performed during create phase (step 2).

5.1.3 Errors

In cases where the specified operation generates an error, this specification does not mandate how, or even if, that error is returned or exposed to the user of an implementation. Unless otherwise stated, generating an error MUST leave the state of the environment as if the operation were never attempted - modulo any possible trivial ancillary changes such as logging.¶

5.1.4 Operations

OCI compliant runtimes MUST support the following operations, unless the operation is not supported by the base operating system.¶

> **Note**
> These operations are not specifying any command line APIs, and the parameters are inputs for general operations.

State

`state <container-id>`¶

This operation MUST generate an error if it is not provided the ID of a container. Attempting to query a container that does not exist MUST generate an error. This operation MUST return the state of a container as specified in Section 5.1.1.¶

Create

`create <container-id> <path-to-bundle>`¶

This operation MUST generate an error if it is not provided a path to the bundle and the container ID to associate with the container. If the ID provided is not unique across all containers within the scope of the runtime, or is not valid in any other way, the implementation MUST generate an error and a new container MUST NOT be created. Using the data in config.json, this operation MUST create a new container. This means that all of the resources associated with the container MUST be created, however, the user-specified code MUST NOT be run at this time. If the runtime cannot create the container as specified in the configuration, it MUST generate an error and a new container MUST NOT be created.¶

Upon successful completion of this operation the status property of this container MUST be created.¶

The runtime MAY validate `config.json` against this spec, either generically or with respect to the local system capabilities, before creating the container (step 2). Runtime callers who are interested in pre-create validation can run bundle-validation tools before invoking the create operation.¶

Any changes made to the configuration after this operation will not have an effect on the container.¶

Start

`start <container-id>`¶

This operation MUST generate an error if it is not provided the container ID. Attempting to start a container that does not exist MUST generate an error. Attempting to start an already started container MUST have no effect on the container and MUST generate an error. This operation MUST run the user-specified code as specified by Section 6.4.¶

Upon successful completion of this operation the status property of this container MUST be running.¶

Kill

`kill <container-id> <signal>`¶

This operation MUST generate an error if it is not provided the container ID. Attempting to send a signal to a container that is not running MUST have no effect on the container and MUST generate an error. This operation MUST send the specified signal to the process in the container.¶

When the process in the container is stopped, irrespective of it being as a result of a `kill` operation or any other reason, the status property of this container MUST be stopped.¶

Delete

`delete <container-id>`¶

This operation MUST generate an error if it is not provided the container ID. Attempting to delete a container that does not exist MUST generate an error. Attempting to delete a container whose process is still running MUST generate an error. Deleting a container MUST delete the resources that were created during step 2. Note that resources associated with the container, but not created by this container, MUST NOT be deleted. Once a container is deleted its ID MAY be used by a subsequent container.¶

Hooks

Many of the operations specified in this specification have "hooks" that allow for additional actions to be taken before or after each operation. See Section 6.10 for more information.¶

5.2 Linux Runtime

5.2.1 File descriptors

By default, only the `stdin`, `stdout` and `stderr` file descriptors are kept open for the application by the runtime. The runtime MAY pass additional file descriptors to the application to support features such as socket activation. Some of the file descriptors MAY be redirected to `/dev/null` even though they are open.¶

5.2.2 Dev symbolic links

After the container has `/proc` mounted, the following standard symlinks MUST be setup within `/dev/` for the IO.¶

Table 1: Required symbolic links

Source	Destination
`/proc/self/fd`	`/dev/fd`
`/proc/self/fd/0`	`/dev/stdin`
`/proc/self/fd/1`	`/dev/stdout`
`/proc/self/fd/2`	`/dev/stderr`

6 Container Configuration

The configuration contains metadata necessary to implement standard operations against the container. This includes the process to run, environment variables to inject, sandboxing features to use, etc.¶

The canonical schema is defined in this document, but there is a JSON Schema and Go bindings.¶

6.1 Version

`ociVersion`
(string, REQUIRED) MUST be in SemVer v2.0.0 format and specifies the version of this specification with which the configuration complies. This configuration format is semantic versioning and retains forward and backward compatibility within major versions. For example, if a configuration is compliant with version 1.1 of this specification, it is compatible with all runtimes that support any 1.1 or later release of this specification, but is not compatible with a runtime that supports 1.0 and not 1.1.

Example

```
{
  "ociVersion": "1.0.0-rc2",
  ...
}
```

6.2 Root filesystem

`root`
(object, REQUIRED) configures the container's root filesystem.

The following properties can be specified:¶

`path`
(string, REQUIRED) Specifies the path to the root filesystem for the container. The path can be an absolute path (starting with `/`) or a relative path (not starting with `/`), which is relative to the bundle. For example (Linux), with a bundle at `/to/bundle` and a root filesystem at `/to/bundle/rootfs`, the `path` value can be either `/to/bundle/rootfs` or `rootfs`. A directory MUST exist at the path declared by the field.

`readonly`
(bool, OPTIONAL) If true then the root filesystem MUST be read-only inside the container, defaults to false.

Example

```
{
  "root": {
    "path": "rootfs",
    "readonly": true
  },
  ...
}
```

6.3 Mounts

`mounts`
(array, OPTIONAL) configures additional mounts (on top of Section 6.2). The runtime MUST mount entries in the listed order. The parameters are similar to the ones in the Linux `mount(2)` system call. For Solaris, the mounts corresponds to fs resource in zonecfg(8).

Entries have the following properties:¶

`destination`
(string, REQUIRED) Destination of mount point: path inside container. This value MUST be an absolute path. For the Windows operating system, one mount destination MUST NOT be nested within another mount (e.g., `c:\\foo` and `c:\\foo\\bar`). For the Solaris operating system, this corresponds to "dir" of the fs resource in zonecfg(8).

`type`
(string, REQUIRED) The filesystem type of the filesystem to be mounted. Linux: **filesystemtype** argument supported by the kernel are listed in **/proc/filesystems** (e.g., "minix", "ext2", "ext3", "jfs", "xfs", "reiserfs", "msdos", "proc", "nfs", "iso9660"). Windows: ntfs. Solaris: corresponds to "type" of the fs resource in zonecfg(8).

`source`
(string, REQUIRED) A device name, but can also be a directory name or a dummy. Windows: the volume name that is the target of the mount point, `\\?\Volume\{GUID}\` (on Windows source is called target). Solaris: corresponds to "special" of the fs resource in zonecfg(8).

`options`
(array of strings, OPTIONAL) Mount options of the filesystem to be used. Linux: supported options are listed in `mount(8)`. Solaris: corresponds to "options" of the fs resource in zonecfg(8).

Linux Example

```
{
  "mounts": [
    {
      "destination": "/tmp",
      "type": "tmpfs",
      "source": "tmpfs",
      "options": ["nosuid","strictatime","mode=755","size=65536k"]
    },
    {
      "destination": "/data",
      "type": "bind",
      "source": "/volumes/testing",
      "options": ["rbind","rw"]
    }
  ],
  ...
}
```

Windows Example

```
{
  "mounts": [
    "myfancymountpoint": {
      "destination": "C:\\Users\\crosbymichael\\My Fancy Mount Point\\",
      "type": "ntfs",
      "source": "\\\\?\\Volume\\{2eca078d-5cbc-43d3-aff8-7e8511f60d0e}\\",
      "options": []
    }
  ],
  ...
}
```

See links for details about mountvol and SetVolumeMountPoint in Windows.¶

Solaris Example

```
{
  "mounts": [
    {
      "destination": "/opt/local",
      "type": "lofs",
      "source": "/usr/local",
      "options": ["ro","nodevices"]
    },
    {
      "destination": "/opt/sfw",
      "type": "lofs",
      "source": "/opt/sfw"
    }
  ],
  ...
}
```

6.4 Process

`process`
(object, REQUIRED) configures the container process.

The `process` schema has the following properties:¶

`terminal`
(bool, OPTIONAL) specifies whether a terminal is attached to that process, defaults to false. On Linux, a pseudoterminal pair is allocated for the container process and the pseudoterminal slave is duplicated on the container process's standard streams.

`consoleSize`
(object, OPTIONAL) specifies the console size of the terminal if attached, containing the following properties:

`height`
(uint, REQUIRED)

`width`
(uint, REQUIRED)

`cwd`
(string, REQUIRED) is the working directory that will be set for the executable. This value MUST be an absolute path.

`env`
(array of strings, OPTIONAL) contains a list of variables that will be set in the process's environment prior to execution. Elements in the array are specified as Strings in the form "KEY=value". The left hand side MUST consist solely of letters, digits, and underscores _ as outlined in IEEE Std 1003.1-2001.

`args`
(array of strings, REQUIRED) executable to launch and any flags as an array. The executable is the first element and MUST be available at the given path inside of the rootfs. If the executable path is not an absolute path then the search `$PATH` is interpreted to find the executable.

For Linux-based systems the process structure supports the following process specific fields:¶

`capabilities`
(array of strings, OPTIONAL) capabilities is an array that specifies Linux capabilities that can be provided to the process inside the container. Valid values are the strings for capabilities defined in the man page.

`rlimits`
(array of rlimits, OPTIONAL) rlimits is an array of rlimits that allows setting resource limits for a process inside the container. The kernel enforces the `soft` limit for a resource while the `hard` limit acts as a ceiling for that value that could be set by an unprivileged process. Valid values for the `type` field are the resources defined in the man page.

`apparmorProfile`
(string, OPTIONAL) apparmor profile specifies the name of the apparmor profile that will be used for the container. For more information about Apparmor, see Apparmor documentation.

`selinuxLabel`
(string, OPTIONAL) SELinux process label specifies the label with which the processes in a container are run. For more information about SELinux, see Selinux documentation.

`noNewPrivileges`
(bool, OPTIONAL) setting `noNewPrivileges` to true prevents the processes in the container from gaining additional privileges. The kernel doc has more information on how this is achieved using a `prctl` system call.

6.4.1 User

The user for the process is a platform-specific structure that allows specific control over which user the process runs as.¶

Linux and Solaris User

For Linux and Solaris based systems the user structure has the following fields:¶

`uid`
(int, REQUIRED) specifies the user ID in the container namespace.

`gid`
(int, REQUIRED) specifies the group ID in the container namespace.

`additionalGids`
(array of ints, OPTIONAL) specifies additional group IDs (in the container namespace) to be added to the process.

Note
Symbolic name for `uid` and `gid`, such as `uname` and `gname` respectively, are left to upper levels to derive (i.e. `/etc/passwd` parsing, NSS, etc.).

Note
For Solaris, `uid` and `gid` specify the `uid` and `gid` of the process inside the container and need not be same as in the host.

Linux Example

```
{
  "process": {
    "terminal": true,
    "consoleSize": {
      "height": 25,
      "width": 80
    },
    "user": {
      "uid": 1,
      "gid": 1,
      "additionalGids": [5, 6]
    },
```

```
    "env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "TERM=xterm"
    ],
    "cwd": "/root",
    "args": [
      "sh"
    ],
    "apparmorProfile": "acme_secure_profile",
    "selinuxLabel": "system_u:system_r:svirt_lxc_net_t:s0:c124,c675",
    "noNewPrivileges": true,
    "capabilities": [
      "CAP_AUDIT_WRITE",
      "CAP_KILL",
      "CAP_NET_BIND_SERVICE"
    ],
    "rlimits": [
      {
        "type": "RLIMIT_NOFILE",
        "hard": 1024,
        "soft": 1024
      }
    ]
  },
  ...
}
```

Solaris Example

```
{
  "process": {
    "terminal": true,
    "consoleSize": {
      "height": 25,
      "width": 80
    },
    "user": {
      "uid": 1,
      "gid": 1,
      "additionalGids": [2, 8]
    },
    "env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "TERM=xterm"
    ],
    "cwd": "/root",
    "args": [
      "/usr/bin/bash"
    ]
  },
  ...
}
```

Windows User

For Windows based systems the user structure has the following fields:¶

`username`
(string, OPTIONAL) specifies the user name for the process.

Windows Example

```
{
  "process": {
    "terminal": true,
    "user": {
      "username": "containeradministrator"
    },
    "env": [
      "VARIABLE=1"
    ],
    "cwd": "c:\\foo",
    "args": [
      "someapp.exe",
    ]
  },
  ...
}
```

6.5 Hostname

`hostname`
(string, OPTIONAL) configures the container's hostname as seen by processes running inside the container. On Linux, you can only set this if your bundle creates a new UTS namespace.

Example

```
{
  "hostname": "mrsdalloway",
  ...
}
```

6.6 Platform

`platform`
(object, REQUIRED) specifies the configuration's target platform.

The following properties can be specified:¶

`os`
(string, REQUIRED) specifies the operating system family this image targets. The runtime MUST generate an error if it does not support the configured `os`. Bundles SHOULD use, and runtimes SHOULD understand, `os` entries listed in the Go Language document for `$GOOS`. If an operating system is not included in the `$GOOS` documentation, it SHOULD be submitted to this specification for standardization.

`arch`
(string, REQUIRED) specifies the instruction set for which the binaries in the image have been compiled. The runtime MUST generate an error if it does not support the configured `arch`. Values for **`arch`** SHOULD use, and runtimes SHOULD understand, `arch` entries listed in the Go Language document for `$GOARCH`. If an architecture is not included in the `$GOARCH` documentation, it SHOULD be submitted to this specification for standardization.

Example

```
{
  "platform": {
    "os": "linux",
    "arch": "amd64"
  },
  ...
}
```

platform.os is used to lookup further platform-specific configuration.¶

`linux`
: (object, OPTIONAL) Section 6.7. This SHOULD only be set if `platform.os` is `linux`.

`solaris`
: (object, OPTIONAL) Section 6.8. This SHOULD only be set if `platform.os` is `solaris`.

`windows`
: (object, OPTIONAL) <<config-windows>. This SHOULD only be set if `platform.os` is `windows`.

Linux Example

```
{
  "platform": {
    "os": "linux",
    "arch": "amd64"
  },
  "linux": {
    "namespaces": [
      {
        "type": "pid"
      }
    ]
  },
  ...
}
```

6.7 Linux-specific Container Configuration

The Linux container specification uses various kernel features like namespaces, cgroups, capabilities, LSM, and filesystem jails to fulfill the spec.¶

6.7.1 Default Filesystems

The Linux ABI includes both syscalls and several special file paths. Applications expecting a Linux environment will very likely expect these file paths to be setup correctly.¶

The following filesystems MUST be made available in each application's filesystem:¶

Table 2: Required filesystems

Path	Type
`/proc`	`procfs`
`/sys`	`sysfs`
`/dev/pts`	`devpts`
`/dev/shm`	`tmpfs`

6.7.2 Namespaces

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. For more information, see the man page.¶

`namespaces`
(array, OPTIONAL) configures the container's namespaces.

Entries have the following properties:¶

`type`
(string, REQUIRED) - namespace type. The following namespace types are supported:

`pid`
processes inside the container will only be able to see other processes inside the same container.

`network`
the container will have its own network stack.

`mount`
the container will have an isolated mount table.

`ipc`
processes inside the container will only be able to communicate to other processes inside the same container via system level IPC.

`uts`
the container will be able to have its own hostname and domain name.

`user`
the container will be able to remap user and group IDs from the host to local users and groups within the container.

`cgroup`
the container will have an isolated view of the cgroup hierarchy.

`path`
(string, OPTIONAL) - path to namespace file in the runtime mount namespace.

If a path is specified, that particular file is used to join that type of namespace. If a namespace type is not specified in the `namespaces` array, the container MUST inherit the runtime namespace of that type. If a new namespace is not created (because the namespace type is not listed, or because it is listed with a `path`), runtimes MUST assume that the setup for that namespace has already been done and error out if the config specifies anything else related to that namespace. If a `namespaces` field contains duplicated namespaces with same `type`, the runtime MUST error out.¶

Example

```
{
  "linux": {
    "namespaces": [
      {
        "type": "pid",
        "path": "/proc/1234/ns/pid"
      },
      {
        "type": "network",
        "path": "/var/run/netns/neta"
      },
      {
        "type": "mount"
      },
```

```
      {
        "type": "ipc"
      },
      {
        "type": "uts"
      },
      {
        "type": "user"
      },
      {
        "type": "cgroup"
      }
    ],
  },
  ...
}
```

6.7.3 User namespace mappings

`uidMappings`
(array of objects, OPTIONAL) describes the user namespace uid mappings from the host to the container.

`gidMappings`
(array of objects, OPTIONAL) describes the user namespace gid mappings from the host to the container.

Each entry has the following structure:¶

`hostID`
(uint32, REQUIRED) - is the starting uid/gid on the host to be mapped to containerID.

`containerID`
(uint32, REQUIRED) - is the starting uid/gid in the container.

`size`
(uint32, REQUIRED) - is the number of ids to be mapped.

The runtime SHOULD NOT modify the ownership of referenced filesystems to realize the mapping. There is a limit of 5 mappings which is the Linux kernel hard limit.¶

Example

```
{
  "linux": {
    "namespaces": [
      {
        "type": "user"
      }
    ],
    "uidMappings": [
      {
        "hostID": 1000,
        "containerID": 0,
        "size": 32000
      }
    ],
    "gidMappings": [
      {
        "hostID": 1000,
        "containerID": 0,
```

```
        "size": 32000
      }
    ]
  },
  ...
}
```

6.7.4 Devices

`devices`
(array of objects, OPTIONAL) lists devices that MUST be available in the container. The runtime may supply them however it likes (with mknod, by bind mounting from the runtime mount namespace, etc.).

Each entry has the following structure:¶

`type`
(string, REQUIRED) - type of device: `c`, `b`, `u` or `p`. More info in mknod(1).

`path`
(string, REQUIRED) - full path to device inside container.

`major, minor`
(int64, REQUIRED unless type is `p`) - major, minor numbers for the device.

`fileMode`
(uint32, OPTIONAL) - file mode for the device. You can also control access to devices with cgroups.

`uid`
(uint32, OPTIONAL) - id of device owner.

`gid`
(uint32, OPTIONAL) - id of device group.

Example

```
{
  "linux": {
    "namespaces": [
      {
        "type": "mount"
      },
    ],
    "devices": [
      {
        "path": "/dev/fuse",
        "type": "c",
        "major": 10,
        "minor": 229,
        "fileMode": 438,
        "uid": 0,
        "gid": 0
      },
      {
        "path": "/dev/sda",
        "type": "b",
        "major": 8,
        "minor": 0,
        "fileMode": 432,
        "uid": 0,
```

```
        "gid": 0
      }
    ],
  }
  ...
}
```

Default Devices

In addition to any devices configured with this setting, the runtime MUST also supply:¶

- `/dev/null`.
- `/dev/zero`.
- `/dev/full`.
- `/dev/random`.
- `/dev/urandom`.
- `/dev/tty`.
- `/dev/console` is setup if terminal is enabled in the config by bind mounting the pseudoterminal slave to `/dev/console`.
- `/dev/ptmx`. A bind-mount or symlink of the container's `/dev/pts/ptmx`.

6.7.5 Control groups

Also known as cgroups, they are used to restrict resource usage for a container and handle device access. cgroups provide controls (through controllers) to restrict cpu, memory, IO, pids and network for the container. For more information, see the kernel cgroups documentation.¶

The path to the cgroups can be specified in the Spec via `cgroupsPath`. `cgroupsPath` can be used to either control the cgroup hierarchy for containers or to run a new process in an existing container. If `cgroupsPath` is:¶

- ... an absolute path (starting with `/`), the runtime MUST take the path to be relative to the cgroup mount point.
- ... a relative path (not starting with `/`), the runtime MAY interpret the path relative to a runtime-determined location in the cgroup hierarchy.
- ... not specified, the runtime MAY define the default cgroup path.

Runtimes MAY consider certain `cgroupsPath` values to be invalid, and MUST generate an error if this is the case. If a `cgroupsPath` value is specified, the runtime MUST consistently attach to the same place in the cgroup hierarchy given the same value of `cgroupsPath`.¶

Implementations of the Spec can choose to name cgroups in any manner. The Spec does not include naming schema for cgroups. The Spec does not support per-controller paths for the reasons discussed in the cgroupv2 documentation. The cgroups will be created if they don't exist.¶

You can configure a container's cgroups via the `resources` field of the Linux configuration. Do not specify `resources` unless limits have to be updated. For example, to run a new process in an existing container without updating limits, `resources` need not be specified.¶

A runtime MUST at least use the minimum set of cgroup controllers required to fulfill the `resources` settings. However, a runtime MAY attach the container process to additional cgroup controllers supported by the system.¶

Example

```
{
  "linux": {
    "cgroupsPath": "/myRuntime/myContainer",
    "resources": {
      "memory": {
        "limit": 100000,
        "reservation": 200000
      },
      "devices": [
        {
          "allow": false,
          "access": "rwm"
        }
      ]
    }
  },
  ...
}
```

Device whitelist

`devices`
(array of objects, OPTIONAL) configures the device whitelist. The runtime MUST apply entries in the listed order.

Each entry has the following structure:¶

`allow`
(boolean, REQUIRED) - whether the entry is allowed or denied.

`type`
(string, OPTIONAL) - type of device: `a` (all), `c` (char), or `b` (block). `null` or unset values mean "all", mapping to `a`.

`major`, `minor`
(int64, OPTIONAL) - major, minor numbers for the device. `null` or unset values mean "all", mapping to `*` in the filesystem API.

`access`
(string, OPTIONAL) - cgroup permissions for device. A composition of `r` (read), `w` (write), and `m` (mknod).

Example

```
{
  "linux": {
    "resources": {
      "devices": [
        {
          "allow": false,
          "access": "rwm"
        },
        {
          "allow": true,
          "type": "c",
          "major": 10,
          "minor": 229,
          "access": "rw"
        },
        {
          "allow": true,
```

```
          "type": "b",
          "major": 8,
          "minor": 0,
          "access": "r"
        }
      ]
    }
  },
  ...
}
```

Disable out-of-memory killer

`disableOOMKiller` contains a boolean (`true` or `false`) that enables or disables the Out of Memory killer for a cgroup. If enabled (`false`), tasks that attempt to consume more memory than they are allowed are immediately killed by the OOM killer. The OOM killer is enabled by default in every cgroup using the `memory` subsystem. To disable it, specify a value of `true`. For more information, see the memory cgroup man page.¶

`disableOOMKiller`
: (bool, OPTIONAL) - enables or disables the OOM killer

Example

```
{
  "linux": {
    "resources": {
      "disableOOMKiller": false
    }
  },
  ...
}
```

Set `oom_score_adj`

`oomScoreAdj` sets heuristic regarding how the process is evaluated by the kernel during memory pressure. For more information, see the proc filesystem documentation section 3.1. This is a kernel/system level setting, where as `disableOOMKiller` is scoped for a memory cgroup. For more information on how these two settings work together, see the memory cgroup documentation section 10. OOM Contol.¶

`oomScoreAdj`
: (int, OPTIONAL) - adjust the oom-killer score

Example

```
{
  "linux": {
    "resources": {
      "oomScoreAdj": 100
    },
  },
  ...
}
```

Memory

`memory`
(object, OPTIONAL) represents the cgroup subsystem `memory` and it's used to set limits on the container's memory usage. For more information, see the memory cgroup man page.

The following parameters can be specified to setup the controller:¶

`limit`
(uint64, OPTIONAL) - sets limit of memory usage in bytes

`reservation`
(uint64, OPTIONAL) - sets soft limit of memory usage in bytes

`swap`
(uint64, OPTIONAL) - sets limit of memory+Swap usage

`kernel`
(uint64, OPTIONAL) - sets hard limit for kernel memory

`kernelTCP`
(uint64, OPTIONAL) - sets hard limit in bytes for kernel TCP buffer memory

`swappiness`
(uint64, OPTIONAL) - sets swappiness parameter of vmscan (See sysctl's vm.swappiness)

Example

```
{
  "linux": {
    "resources": {
      "memory": {
        "limit": 536870912,
        "reservation": 536870912,
        "swap": 536870912,
        "kernel": 0,
        "kernelTCP": 0,
        "swappiness": 0
      }
    }
  },
  ...
}
```

CPU

`cpu`
(object, OPTIONAL) represents the cgroup subsystems `cpu` and `cpusets`. For more information, see the cpusets cgroup man page.

The following parameters can be specified to setup the controller:¶

`shares`
(uint64, OPTIONAL) - specifies a relative share of CPU time available to the tasks in a cgroup

`quota`
(uint64, OPTIONAL) - specifies the total amount of time in microseconds for which all tasks in a cgroup can run during one period (as defined by period below)

`period`
(uint64, OPTIONAL) - specifies a period of time in microseconds for how regularly a cgroup's access to CPU resources should be reallocated (CFS scheduler only)

`realtimeRuntime`
(uint64, OPTIONAL) - specifies a period of time in microseconds for the longest continuous period in which the tasks in a cgroup have access to CPU resources

`realtimePeriod`
(uint64, OPTIONAL) - same as period but applies to realtime scheduler only

`cpus`
(string, OPTIONAL) - list of CPUs the container will run in

`mems`
(string, OPTIONAL) - list of Memory Nodes the container will run in

Example

```
{
  "linux": {
    "resources": {
      "cpu": {
        "shares": 1024,
        "quota": 1000000,
        "period": 500000,
        "realtimeRuntime": 950000,
        "realtimePeriod": 1000000,
        "cpus": "2-3",
        "mems": "0-7"
      }
    }
  },
  ...
}
```

Block IO Controller

`blockIO`
(object, OPTIONAL) represents the cgroup subsystem `blkio` which implements the block IO controller. For more information, see the kernel cgroups documentation about blkio.

The following parameters can be specified to setup the controller:¶

`blkioWeight`
(uint16, OPTIONAL) - specifies per-cgroup weight. This is default weight of the group on all devices until and unless overridden by per-device rules. The range is from 10 to 1000.

`blkioLeafWeight`
(uint16, OPTIONAL)* - equivalents of `blkioWeight` for the purpose of deciding how much weight tasks in the given cgroup has while competing with the cgroup's child cgroups. The range is from 10 to 1000.

`blkioWeightDevice`
(array, OPTIONAL) - specifies the list of devices which will be bandwidth rate limited. The following parameters can be specified per-device:

`major,minor`
(int64, REQUIRED) - major, minor numbers for device. More info in mknod(1).

`weight`
(uint16, OPTIONAL) - bandwidth rate for the device, range is from 10 to 1000

`leafWeight`
(uint16, OPTIONAL) - bandwidth rate for the device while competing with the cgroup's child cgroups, range is from 10 to 1000, CFQ scheduler only

You must specify at least one of `weight` or `leafWeight` in a given entry, and can specify both.¶

`blkioThrottleReadBpsDevice`, `blkioThrottleWriteBpsDevice`, `blkioThrottleReadIOPSDevice`, `blkioT`

(array, OPTIONAL) - specify the list of devices which will be IO rate limited. The following parameters can be specified per-device:

`major`, `minor`
(int64, REQUIRED) - major, minor numbers for device. More info in mknod(1).

`rate`
(uint64, REQUIRED) - IO rate limit for the device

Example

```
{
  "linux": {
    "resources": {
      "blockIO": {
        "blkioWeight": 10,
        "blkioLeafWeight": 10,
        "blkioWeightDevice": [
          {
            "major": 8,
            "minor": 0,
            "weight": 500,
            "leafWeight": 300
          },
          {
            "major": 8,
            "minor": 16,
            "weight": 500
          }
        ],
        "blkioThrottleReadBpsDevice": [
          {
            "major": 8,
            "minor": 0,
            "rate": 600
          }
        ],
        "blkioThrottleWriteIOPSDevice": [
          {
            "major": 8,
            "minor": 16,
            "rate": 300
          }
        ]
      }
    }
  },
  ...
}
```

Huge page limits

`hugepageLimits`
(array of objects, OPTIONAL) represents the `hugetlb` controller which allows to limit the HugeTLB usage per control group and enforces the controller limit during page fault. For more information, see the kernel cgroups documentation about HugeTLB.

Each entry has the following structure:¶

`pageSize`
(string, REQUIRED) - hugepage size

`limit`
(uint64, REQUIRED) - limit in bytes of *hugepagesize* HugeTLB usage

Example

```
{
  "linux": {
    "resources": {
      "hugepageLimits": [
        {
          "pageSize": "2MB",
          "limit": 9223372036854771712
        }
      ]
    }
  },
  ...
}
```

Network

`network`
(object, OPTIONAL) represents the cgroup subsystems `net_cls` and `net_prio`. For more information, see the `net_cls` cgroup man page and the `net_prio` cgroup man page.

The following parameters can be specified to setup the controller:¶

`classID`
(uint32, OPTIONAL) - is the network class identifier the cgroup's network packets will be tagged with

`priorities`
(array, OPTIONAL) - specifies a list of objects of the priorities assigned to traffic originating from processes in the group and egressing the system on various interfaces. The following parameters can be specified per-priority:

`name`
(string, REQUIRED) - interface name

`priority`
(uint32, REQUIRED) - priority applied to the interface

Example

```
{
  "linux": {
    "resources": {
      "network": {
        "classID": 1048577,
        "priorities": [
          {
            "name": "eth0",
            "priority": 500
          },
          {
            "name": "eth1",
            "priority": 1000
          }
        ]
      }
    }
  },
  ...
}
```

PIDs

`pids`
(object, OPTIONAL) represents the cgroup subsystem `pids`. For more information, see the pids cgroup man page.

The following parameters can be specified to setup the controller:¶

`limit`
(int64, REQUIRED) - specifies the maximum number of tasks in the cgroup

Example

```
{
  "linux": {
    "resources": {
      "pids": {
        "limit": 32771
      }
    }
  },
  ...
}
```

6.7.6 Sysctl

`sysctl`
(object, OPTIONAL) allows kernel parameters to be modified at runtime for the container. For more information, see the man page.

Example

```
{
  "linux": {
    "sysctl": {
```

```
      "net.ipv4.ip_forward": "1",
      "net.core.somaxconn": "256"
    }
  },
  ...
}
```

6.7.7 Seccomp

Seccomp provides application sandboxing mechanism in the Linux kernel. Seccomp configuration allows one to configure actions to take for matched syscalls and furthermore also allows matching on values passed as arguments to syscalls. For more information about Seccomp, see Seccomp kernel documentation. The actions, architectures, and operators are strings that match the definitions in `seccomp.h` from libseccomp and are translated to corresponding values. A valid list of constants as of libseccomp v2.3.0 is shown below.¶

Architecture Constants:¶

- `SCMP_ARCH_X86`
- `SCMP_ARCH_X86_64`
- `SCMP_ARCH_X32`
- `SCMP_ARCH_ARM`
- `SCMP_ARCH_AARCH64`
- `SCMP_ARCH_MIPS`
- `SCMP_ARCH_MIPS64`
- `SCMP_ARCH_MIPS64N32`
- `SCMP_ARCH_MIPSEL`
- `SCMP_ARCH_MIPSEL64`
- `SCMP_ARCH_MIPSEL64N32`
- `SCMP_ARCH_PPC`
- `SCMP_ARCH_PPC64`
- `SCMP_ARCH_PPC64LE`
- `SCMP_ARCH_S390`
- `SCMP_ARCH_S390X`

Action Constants:¶

- `SCMP_ACT_KILL`
- `SCMP_ACT_TRAP`
- `SCMP_ACT_ERRNO`
- `SCMP_ACT_TRACE`
- `SCMP_ACT_ALLOW`

Operator Constants:¶

- SCMP_CMP_NE
- SCMP_CMP_LT
- SCMP_CMP_LE
- SCMP_CMP_EQ
- SCMP_CMP_GE
- SCMP_CMP_GT
- SCMP_CMP_MASKED_EQ

Example

```
{
  "linux": {
    "seccomp": {
      "defaultAction": "SCMP_ACT_ALLOW",
      "architectures": [
        "SCMP_ARCH_X86"
      ],
      "syscalls": [
        {
          "name": "getcwd",
          "action": "SCMP_ACT_ERRNO"
        }
      ]
    }
  },
  ...
}
```

6.7.8 Rootfs Mount Propagation

`rootfsPropagation`
(string, OPTIONAL) sets the rootfs's mount propagation. Its value is either `slave`, `private`, or `shared`. The kernel doc has more information about mount propagation.

Example

```
{
  "linux": {
    "rootfsPropagation": "slave"
  },
  ...
}
```

6.7.9 Masked Paths

`maskedPaths`
(array of strings, OPTIONAL) will mask over the provided paths inside the container so that they cannot be read. The values MUST be absolute paths in the container namespace.

Example

```
{
  "linux": {
    "maskedPaths": [
      "/proc/kcore"
    ]
  },
  ...
}
```

6.7.10 Readonly Paths

`readonlyPaths`
(array of strings, OPTIONAL) will set the provided paths as readonly inside the container. The values MUST be absolute paths in the container namespace.

Example

```
{
  "linux": {
    "readonlyPaths": [
      "/proc/sys"
    ]
  },
  ...
}
```

6.7.11 Mount Label

`mountLabel`
(string, OPTIONAL) will set the Selinux context for the mounts in the container.

Example

```
{
  "linux": {
    "mountLabel": "system_u:object_r:svirt_sandbox_file_t:s0:c715,c811"
  },
  ...
}
```

6.8 Solaris-specific Container Configuration

Solaris application containers can be configured using the following properties, all of the below properties have mappings to properties specified under zonecfg(8) man page, except milestone.¶

6.8.1 milestone

The SMF(Service Management Facility) FMRI which should go to "online" state before we start the desired process within the container.¶

`milestone`
(string, OPTIONAL)

Example

```
{
  "solaris": {
    "milestone": "svc:/milestone/container:default"
  },
  ...
}
```

6.8.2 limitpriv

The maximum set of privileges any process in this container can obtain. The property should consist of a comma-separated privilege set specification as described in `priv_str_to_set(3C)` man page for the respective release of Solaris.¶

`limitpriv`
(string, OPTIONAL)

Example

```
{
  "solaris": {
    "limitpriv": "default"
  },
  ...
}
```

6.8.3 maxShmMemory

The maximum amount of shared memory allowed for this application container. A scale (K, M, G, T) can be applied to the value for each of these numbers (for example, 1M is one megabyte). Mapped to max-shm-memory in zonecfg(8) man page.¶

`maxShmMemory`
(string, OPTIONAL)

Example

```
{
  "solaris": {
    "maxShmMemory": "512m"
  },
  ...
}
```

6.8.4 cappedCPU

Sets a limit on the amount of CPU time that can be used by a container. The unit used translates to the percentage of a single CPU that can be used by all user threads in a container, expressed as a fraction (for example, .75) or a mixed number (whole number and fraction, for example, 1.25). An ncpu value of 1 means 100% of a CPU, a value of 1.25 means 125%, .75 mean 75%, and so forth. When projects within a capped container have their own caps, the minimum value takes precedence. cappedCPU is mapped to capped-cpu in zonecfg(8) man page.¶

`ncpus`
(string, OPTIONAL)

Example

```
{
  "solaris": {
    "cappedCPU": {
      "ncpus": "8"
    }
  },
  ...
}
```

6.8.5 cappedMemory

The physical and swap caps on the memory that can be used by this application container. A scale (K, M, G, T) can be applied to the value for each of these numbers (for example, 1M is one megabyte). cappedMemory is mapped to capped-memory in zonecfg(8) man page.¶

`physical`
(string, OPTIONAL)

`swap`
(string, OPTIONAL)

Example

```
{
  "solaris": {
    "cappedMemory": {
      "physical": "512m",
      "swap": "512m"
    }
  },
  ...
}
```

6.8.6 Network

Automatic Network (anet)

anet is specified as an array that is used to setup networking for Solaris application containers. The anet resource represents the automatic creation of a network resource for an application container. The zones administration daemon, zoneadmd, is the primary process for managing the container's virtual platform. One of the daemons is responsibilities is creation and teardown of the networks for the container. For more information on the daemon check the zoneadmd(1M) man page. When such a container is started, a temporary VNIC(Virtual NIC) is automatically created for the container. The VNIC is deleted when the container is torn down. The following properties can be used to setup automatic networks. For additional information on properties check zonecfg(8) man page for the respective release of Solaris.¶

`linkname`
(string, OPTIONAL) Specify a name for the automatically created VNIC datalink.

`lowerLink`
(string, OPTIONAL) Specify the link over which the VNIC will be created. Mapped to lower-link in the zonecfg(8) man page.

`allowedAddress`
(string, OPTIONAL) The set of IP addresses that the container can use might be constrained by specifying the allowedAddress property. If allowedAddress has not been specified, then they can use any IP address on the associated physical interface for the network resource. Otherwise, when allowedAddress is specified, the container cannot use IP addresses that are not in the allowedAddress list for the physical address. Mapped to allowed-address in the zonecfg(8) man page.

`configureAllowedAddress`
(string, OPTIONAL) If configureAllowedAddress is set to true, the addresses specified by allowedAddress are automatically configured on the interface each time the container starts. When it is set to false, the allowedAddress will not be configured on container start. Mapped to configure-allowed-address in the zonecfg(8) man page.

`defrouter`
(string, OPTIONAL) The value for the OPTIONAL default router.

`macAddress`
(string, OPTIONAL) Set the VNIC's MAC addresses based on the specified value or keyword. If not a keyword, it is interpreted as a unicast MAC address. For a list of the supported keywords please refer to the zonecfg(8) man page of the respective Solaris release. Mapped to mac-address in the zonecfg(8) man page.

`linkProtection`
(string, OPTIONAL) Enables one or more types of link protection using comma-separated values. See the protection property in dladm(8) for supported values in respective release of Solaris. Mapped to link-protection in the zonecfg(8) man page.

Example

```
{
  "solaris": {
    "anet": [
      {
        "allowedAddress": "172.17.0.2/16",
        "configureAllowedAddress": "true",
        "defrouter": "172.17.0.1/16",
        "linkProtection": "mac-nospoof, ip-nospoof",
        "linkname": "net0",
        "lowerLink": "net2",
        "macAddress": "02:42:f8:52:c7:16"
      }
    ]
  },
  ...
}
```

6.9 Windows-specific Container Configuration

The Windows container specification uses APIs provided by the Windows Host Compute Service (HCS) to fulfill the spec.¶

6.9.1 Resources

You can configure a container's resource limits via the OPTIONAL `resources` field of the Windows configuration.¶

Memory

`memory` is an OPTIONAL configuration for the container's memory usage.¶

The following parameters can be specified:¶

`limit`
(uint64, OPTIONAL) - sets limit of memory usage in bytes.

`reservation`
(uint64, OPTIONAL) - sets the guaranteed minimum amount of memory for a container in bytes.

Example

```
{
  "windows": {
    "resources": {
      "memory": {
        "limit": 2097152,
        "reservation": 524288
      }
    }
  },
  ...
}
```

CPU

`cpu` is an OPTIONAL configuration for the container's CPU usage.¶

The following parameters can be specified:¶

`count`
: (uint64, OPTIONAL) - specifies the number of CPUs available to the container.

`shares`
: (uint16, OPTIONAL) - specifies the relative weight to other containers with CPU shares. The range is from 1 to 10000.

`percent`
: (uint, OPTIONAL) - specifies the percentage of available CPUs usable by the container.

Example

```
{
  "windows": {
    "resources": {
      "cpu": {
        "percent": 50
      }
    }
  },
  ...
}
```

Storage

`storage` is an OPTIONAL configuration for the container's storage usage.¶

The following parameters can be specified:¶

`iops`
: (uint64, OPTIONAL) - specifies the maximum IO operations per second for the system drive of the container.

`bps`
: (uint64, OPTIONAL) - specifies the maximum bytes per second for the system drive of the container.

`sandboxSize`
: (uint64, OPTIONAL) - specifies the minimum size of the system drive in bytes.

Example

```
{
  "windows": {
    "resources": {
      "storage": {
        "iops": 50
      }
    }
  },
  ...
}
```

Network

`network` is an OPTIONAL configuration for the container's network usage.¶

The following parameters can be specified:¶

`egressBandwidth`
: (uint64, OPTIONAL) - specified the maximum egress bandwidth in bytes per second for the container.

Example

```
{
  "windows": {
    "resources": {
      "network": {
        "egressBandwidth": 1048577
      }
    }
  },
  ...
}
```

6.10 Hooks

`hooks`
: (object, OPTIONAL) configures callbacks for container lifecycle events. Lifecycle hooks allow custom events for different points in a container's runtime. Presently there are `Prestart`, `Poststart` and `Poststop`.

The following properties can be specified:¶

- prestart is a list of hooks to be run before the container process is executed.
- poststart is a list of hooks to be run immediately after the container process is started.
- poststop is a list of hooks to be run after the container process exits.

Hooks allow one to run code before/after various lifecycle events of the container. Hooks MUST be called in the listed order. The state of the container is passed to the hooks over stdin, so the hooks could get the information they need to do their work.¶

Hook paths are absolute and are executed from the host's filesystem in the runtime namespace.¶

6.10.1 Prestart

The pre-start hooks are called after the container process is spawned, but before the user supplied command is executed. They are called after the container namespaces are created on Linux, so they provide an opportunity to customize the container. In Linux, for e.g., the network namespace could be configured in this hook.¶

If a hook returns a non-zero exit code, then an error including the exit code and the stderr is returned to the caller and the container is torn down.¶

6.10.2 Poststart

The post-start hooks are called after the user process is started. For example this hook can notify user that real process is spawned.¶

If a hook returns a non-zero exit code, then an error is logged and the remaining hooks are executed.¶

6.10.3 Poststop

The post-stop hooks are called after the container process is stopped. Cleanup or debugging could be performed in such a hook. If a hook returns a non-zero exit code, then an error is logged and the remaining hooks are executed.¶

Example

```
{
  "hooks": {
    "prestart": [
      {
        "path": "/usr/bin/fix-mounts",
        "args": ["fix-mounts", "arg1", "arg2"],
        "env":  [ "key1=value1"]
      },
      {
        "path": "/usr/bin/setup-network"
      }
    ],
    "poststart": [
      {
        "path": "/usr/bin/notify-start",
        "timeout": 5
      }
    ],
    "poststop": [
      {
        "path": "/usr/sbin/cleanup.sh",
        "args": ["cleanup.sh", "-f"]
      }
    ]
  },
  ...
}
```

`path` is REQUIRED for a hook. `args` and `env` are OPTIONAL. `timeout` is the number of seconds before aborting the hook. The semantics are the same as `Path`, `Args` and `Env` in golang Cmd.¶

6.11 Annotations

`annotations`
(object, OPTIONAL) contains arbitrary metadata for the container. This information MAY be structured or unstructured. Annotations MUST be a key-value map where both the key and value MUST be strings. While the value MUST be present, it MAY be an empty string. Keys MUST be unique within this map, and best practice is to namespace the keys. Keys SHOULD be named using a reverse domain notation - e.g. `com.example.myKey`. Keys using the `org.opencontainers` namespace are reserved and MUST NOT be used by subsequent specifications. If there are no annotations then this property MAY either be absent or an empty map. Implementations that are reading/processing this configuration file MUST NOT generate an error if they encounter an unknown annotation key.

```
{
  "annotations": {
```

```
    "com.example.gpu-cores": "2"
  },
  ...
}
```

6.12 Extensibility

Implementations that are reading/processing this configuration file MUST NOT generate an error if they encounter an unknown property. Instead they MUST ignore unknown properties.¶

6.13 Example

Here is a full example `config.json` for reference.¶

```
{
  "ociVersion": "0.5.0-dev",
  "platform": {
    "os": "linux",
    "arch": "amd64"
  },
  "process": {
    "terminal": true,
    "user": {
      "uid": 1,
      "gid": 1,
      "additionalGids": [
        5,
        6
      ]
    },
    "args": [
      "sh"
    ],
    "env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "TERM=xterm"
    ],
    "cwd": "/",
    "capabilities": [
      "CAP_AUDIT_WRITE",
      "CAP_KILL",
      "CAP_NET_BIND_SERVICE"
    ],
    "rlimits": [
      {
        "type": "RLIMIT_CORE",
        "hard": 1024,
        "soft": 1024
      },
      {
        "type": "RLIMIT_NOFILE",
        "hard": 1024,
        "soft": 1024
      }
    ],
    "apparmorProfile": "acme_secure_profile",
    "selinuxLabel": "system_u:system_r:svirt_lxc_net_t:s0:c124,c675",
    "noNewPrivileges": true
  },
```

```
  "root": {
    "path": "rootfs",
    "readonly": true
  },
  "hostname": "slartibartfast",
  "mounts": [
    {
      "destination": "/proc",
      "type": "proc",
      "source": "proc"
    },
    {
      "destination": "/dev",
      "type": "tmpfs",
      "source": "tmpfs",
      "options": [
        "nosuid",
        "strictatime",
        "mode=755",
        "size=65536k"
      ]
    },
    {
      "destination": "/dev/pts",
      "type": "devpts",
      "source": "devpts",
      "options": [
        "nosuid",
        "noexec",
        "newinstance",
        "ptmxmode=0666",
        "mode=0620",
        "gid=5"
      ]
    },
    {
      "destination": "/dev/shm",
      "type": "tmpfs",
      "source": "shm",
      "options": [
        "nosuid",
        "noexec",
        "nodev",
        "mode=1777",
        "size=65536k"
      ]
    },
    {
      "destination": "/dev/mqueue",
      "type": "mqueue",
      "source": "mqueue",
      "options": [
        "nosuid",
        "noexec",
        "nodev"
      ]
    },
    {
      "destination": "/sys",
      "type": "sysfs",
      "source": "sysfs",
      "options": [
```

```
        "nosuid",
        "noexec",
        "nodev"
      ]
    },
    {
      "destination": "/sys/fs/cgroup",
      "type": "cgroup",
      "source": "cgroup",
      "options": [
        "nosuid",
        "noexec",
        "nodev",
        "relatime",
        "ro"
      ]
    }
  ],
  "hooks": {
    "prestart": [
      {
        "path": "/usr/bin/fix-mounts",
        "args": [
          "fix-mounts",
          "arg1",
          "arg2"
        ],
        "env": [
          "key1=value1"
        ]
      },
      {
        "path": "/usr/bin/setup-network"
      }
    ],
    "poststart": [
      {
        "path": "/usr/bin/notify-start",
        "timeout": 5
      }
    ],
    "poststop": [
      {
        "path": "/usr/sbin/cleanup.sh",
        "args": [
          "cleanup.sh",
          "-f"
        ]
      }
    ]
  },
  "linux": {
    "devices": [
      {
        "path": "/dev/fuse",
        "type": "c",
        "major": 10,
        "minor": 229,
        "fileMode": 438,
        "uid": 0,
        "gid": 0
      },
```

```
      {
        "path": "/dev/sda",
        "type": "b",
        "major": 8,
        "minor": 0,
        "fileMode": 432,
        "uid": 0,
        "gid": 0
      }
    ],
    "uidMappings": [
      {
        "hostID": 1000,
        "containerID": 0,
        "size": 32000
      }
    ],
    "gidMappings": [
      {
        "hostID": 1000,
        "containerID": 0,
        "size": 32000
      }
    ],
    "sysctl": {
      "net.ipv4.ip_forward": "1",
      "net.core.somaxconn": "256"
    },
    "cgroupsPath": "/myRuntime/myContainer",
    "resources": {
      "network": {
        "classID": 1048577,
        "priorities": [
          {
            "name": "eth0",
            "priority": 500
          },
          {
            "name": "eth1",
            "priority": 1000
          }
        ]
      },
      "pids": {
        "limit": 32771
      },
      "hugepageLimits": [
        {
          "pageSize": "2MB",
          "limit": 9223372036854772000
        }
      ],
      "oomScoreAdj": 100,
      "memory": {
        "limit": 536870912,
        "reservation": 536870912,
        "swap": 536870912,
        "kernel": 0,
        "kernelTCP": 0,
        "swappiness": 0
      },
      "cpu": {
```

```
        "shares": 1024,
        "quota": 1000000,
        "period": 500000,
        "realtimeRuntime": 950000,
        "realtimePeriod": 1000000,
        "cpus": "2-3",
        "mems": "0-7"
      },
      "disableOOMKiller": false,
      "devices": [
        {
          "allow": false,
          "access": "rwm"
        },
        {
          "allow": true,
          "type": "c",
          "major": 10,
          "minor": 229,
          "access": "rw"
        },
        {
          "allow": true,
          "type": "b",
          "major": 8,
          "minor": 0,
          "access": "r"
        }
      ],
      "blockIO": {
        "blkioWeight": 10,
        "blkioLeafWeight": 10,
        "blkioWeightDevice": [
          {
            "major": 8,
            "minor": 0,
            "weight": 500,
            "leafWeight": 300
          },
          {
            "major": 8,
            "minor": 16,
            "weight": 500
          }
        ],
        "blkioThrottleReadBpsDevice": [
          {
            "major": 8,
            "minor": 0,
            "rate": 600
          }
        ],
        "blkioThrottleWriteIOPSDevice": [
          {
            "major": 8,
            "minor": 16,
            "rate": 300
          }
        ]
      }
    },
    "rootfsPropagation": "slave",
```

```
    "seccomp": {
      "defaultAction": "SCMP_ACT_ALLOW",
      "architectures": [
        "SCMP_ARCH_X86"
      ],
      "syscalls": [
        {
          "name": "getcwd",
          "action": "SCMP_ACT_ERRNO"
        }
      ]
    },
    "namespaces": [
      {
        "type": "pid"
      },
      {
        "type": "network"
      },
      {
        "type": "ipc"
      },
      {
        "type": "uts"
      },
      {
        "type": "mount"
      },
      {
        "type": "user"
      },
      {
        "type": "cgroup"
      }
    ],
    "maskedPaths": [
      "/proc/kcore",
      "/proc/latency_stats",
      "/proc/timer_stats",
      "/proc/sched_debug"
    ],
    "readonlyPaths": [
      "/proc/asound",
      "/proc/bus",
      "/proc/fs",
      "/proc/irq",
      "/proc/sys",
      "/proc/sysrq-trigger"
    ],
    "mountLabel": "system_u:object_r:svirt_sandbox_file_t:s0:c715,c811"
  },
  "annotations": {
    "com.example.key1": "value1",
    "com.example.key2": "value2"
  }
}
```