Tasks subsystem

The goal of this proposal is to simplify the overall responsibilities of domain manager by making it only aware of running local (non-hypervisor based) containers AKA Tasks. More complex executors (hypervisor based or emulation based) will then be packaged as simple Tasks themselves.

Definitions/APIs

Any running entity on EVE (be it a group of Linux processes or an active domain consuming memory and CPU) will be managed through the **Task subsystem**. Every task it defined by An OCI spec structure. The OCI spec structure may refer to the native representations of the following EVE Objects:

- A designated Volume with a POSIX FS access interface providing a root filesystem for the task (note that it will be possible to have tasks with root filesystem of the host EVE by either specifying 0 designated volumes or a designated volume of a special kind)
- A number of additional volumes with a POSIX FS access interface mapping into the mountpoints
- A number of network interfaces

A minimum viable OCI spec structure for launching a task must contain:

- spec.Root (pointing at a root filesystem and declaring it either R/W or R/O)
- spec.Process.Args (exec style specification of a task's Anchor Process)
- spec.Linux.LinuxResources.Memory
- spec.Linux.LinuxResources.CPU

Ultimately, it becomes the job of the domainmanager (likely to be renamed taskmanager as part of this refactoring) to fill out OCI spec structure based on the EVE app config entries, create the required task and then drive the lifecycle of each task through the following methods:

- Start (starts a stopped or just created task)
- Stop (stops a running task)
- Delete (deletes a stoped task)

It must be noted, that once this proposal gets implemented no other methods of creating long running entities (os.Exec, etc.) will be allowed on EVE – everything will have to go through the tasks interface.

Implementation

Each running task will consist of a group of Linux processes executing within a task-specific set of namespaces constrained by a task-specific cgroup. The group of processes doesn't have to be a tree, but a single process will be designated as an **Anchor process**. A PID of an Anchor process is the ID of a task. All communications with the task are done through communicating with its Anchor process (this is limited to sending it signals and reading its various metadata fields from a procfs for now, but can later to expanded into making requiring Anchor process to communicate through EVE's message bus mechanism directly).

It is very much a supported configuration for a task to have an anchor process that simply controls the execution of an outside entity and doesn't have any meaningful useful function of its own. The primary use case for this type of task configuration is to have a **Device model** anchor process (either qemu or xl) controlling the execution of an isolated hypervisor domain. However, even in this configuration all resources accounting (CPU, memory, etc.) is done at the Task level and thus may require certain "anchor process tax" to be introduced in addition to whatever the outside entity will be consuming. E.g: in order to run a hypervisor domain with X Mb or RAM domainmanager will have to create a Task with X + delta RAM allocated to account for qemu anchor process.

Initially, we plan to provide a very thin wrapper over native containerd TaskClient interface to represent a handle that domainmanager will be using for task management.

Discussion

This proposal provides a pleasant unification property between running things natively on EVE vs having EVE drive the lifecycle of hypervisor domains. In fact, it completely eliminates the difference between the two by simply focusing on where to pick up the appropriate seed of an OCI spec to create the task:

1. From a rootfs OCI container spec (in case of a native EVE container running without any kind of hypervisor support)
2. From a "built-in" designated "hypervisor tools" container (in case of a workload that requires a hypervisor domain)

This further offers additional opportunities for optimization down the road:

- We can eliminate this #1 vs #2 logic completely from the domain manager and have the controller fill out the appropriate data structure telling EVE where to pick up OCI spec for the task
- We can provide dynamic "hypervisor tools" containers thus opening up a plethora of possibilities such as:
    - Be able to run cross architecture code (e.g. running ARM binaries on x86 and vice versa)
    - Be able to run VM based binary artifacts (e.g. having Tasks defined by Volumes derived from JAR files)

Note that the dynamic "hypervisor tools" containers become completely transparent to EVE itself – they simply will be specified in the EVE config and downloaded like any other container images