KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon CloudNativeCon THE LINUX FOUNDATION OPEN SOURCE SUMMIT AI_dev Open Source GenAI & ML Summit

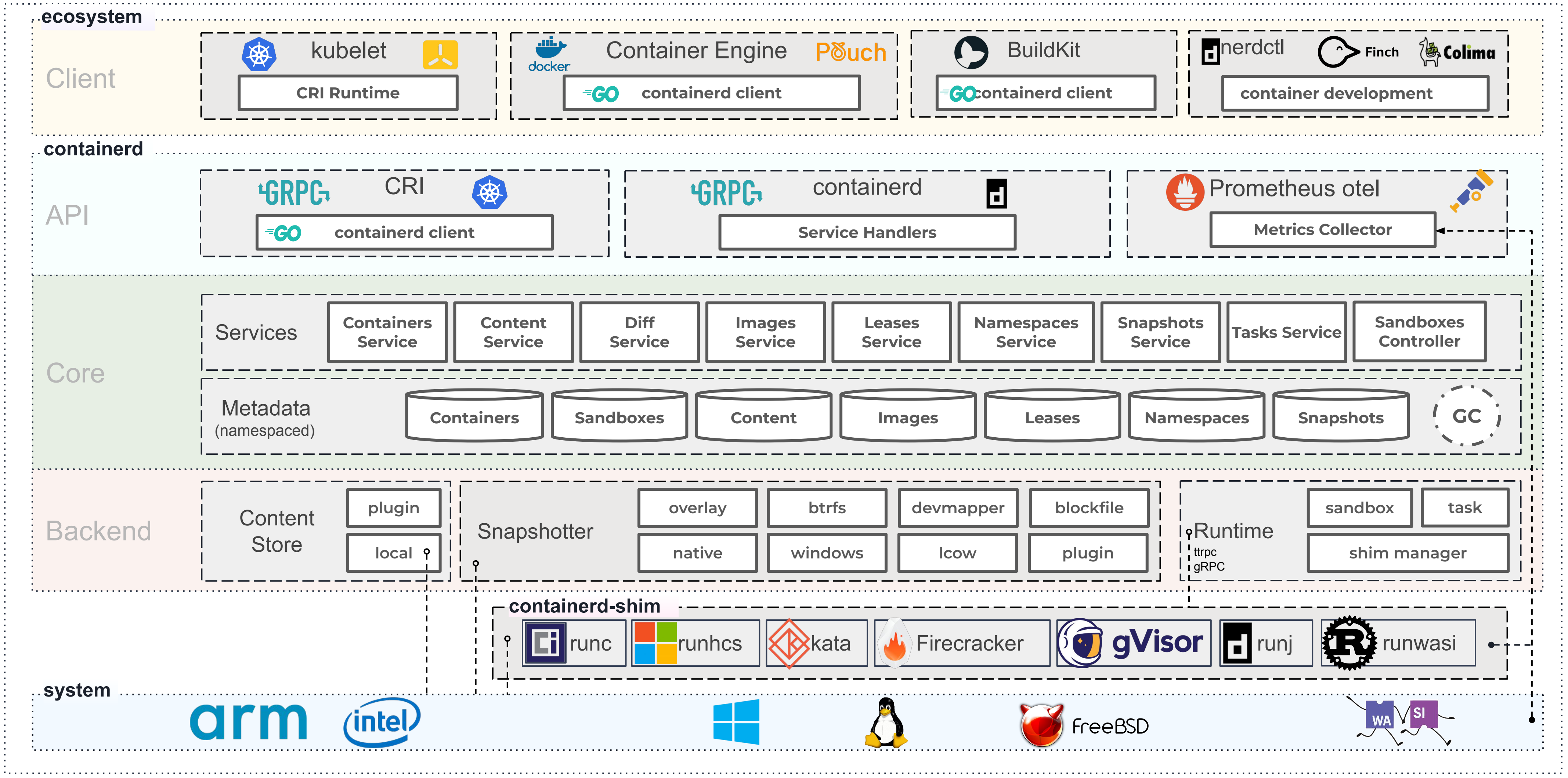
China 2024

Build Container Runtime Based on Sandbox API of Containerd

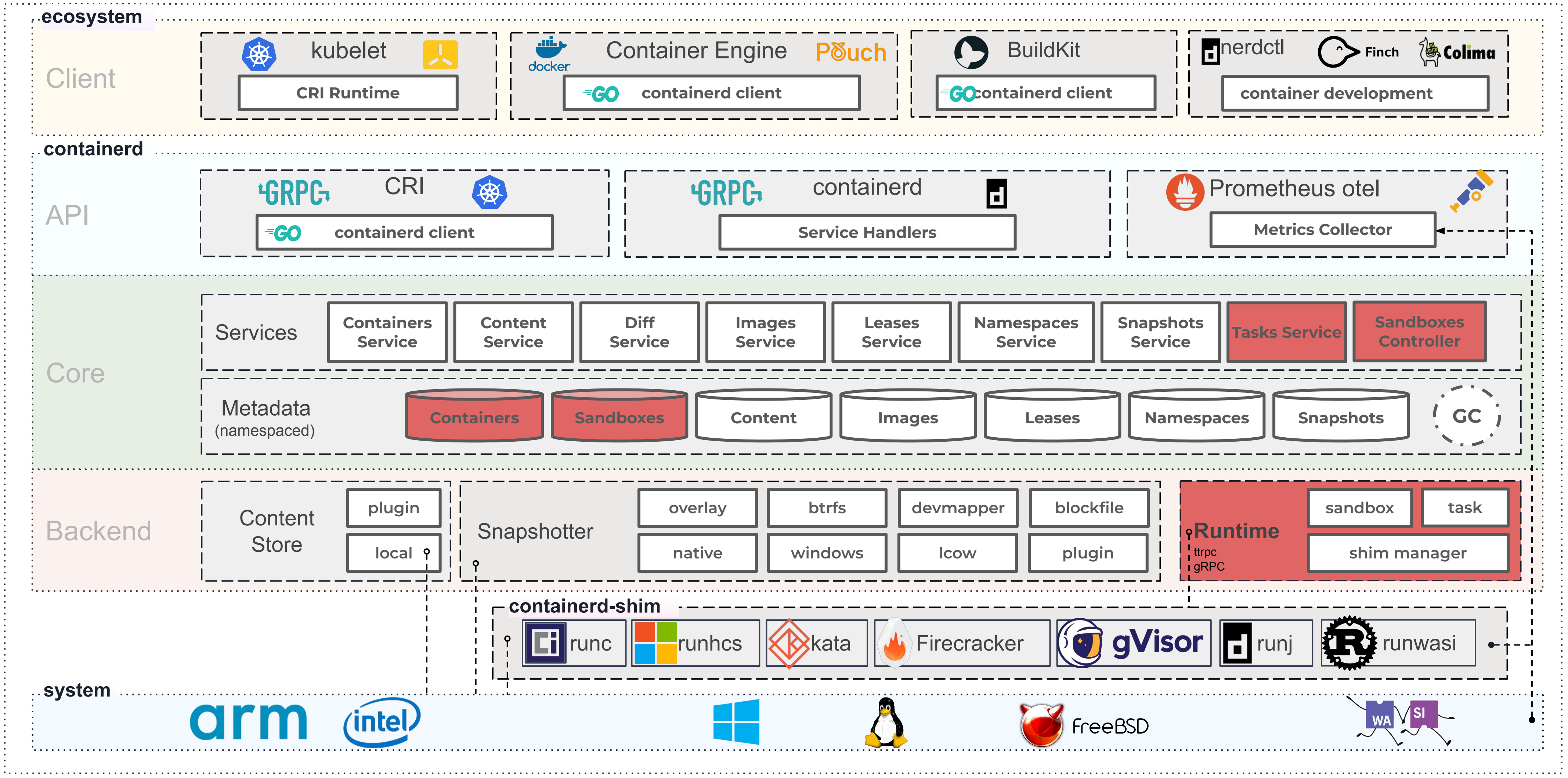
Cai Wei, DaoCloud
Shaobao Feng, HuaweiCloud

Build **Container** Runtime based on **Sandbox** API of Containerd

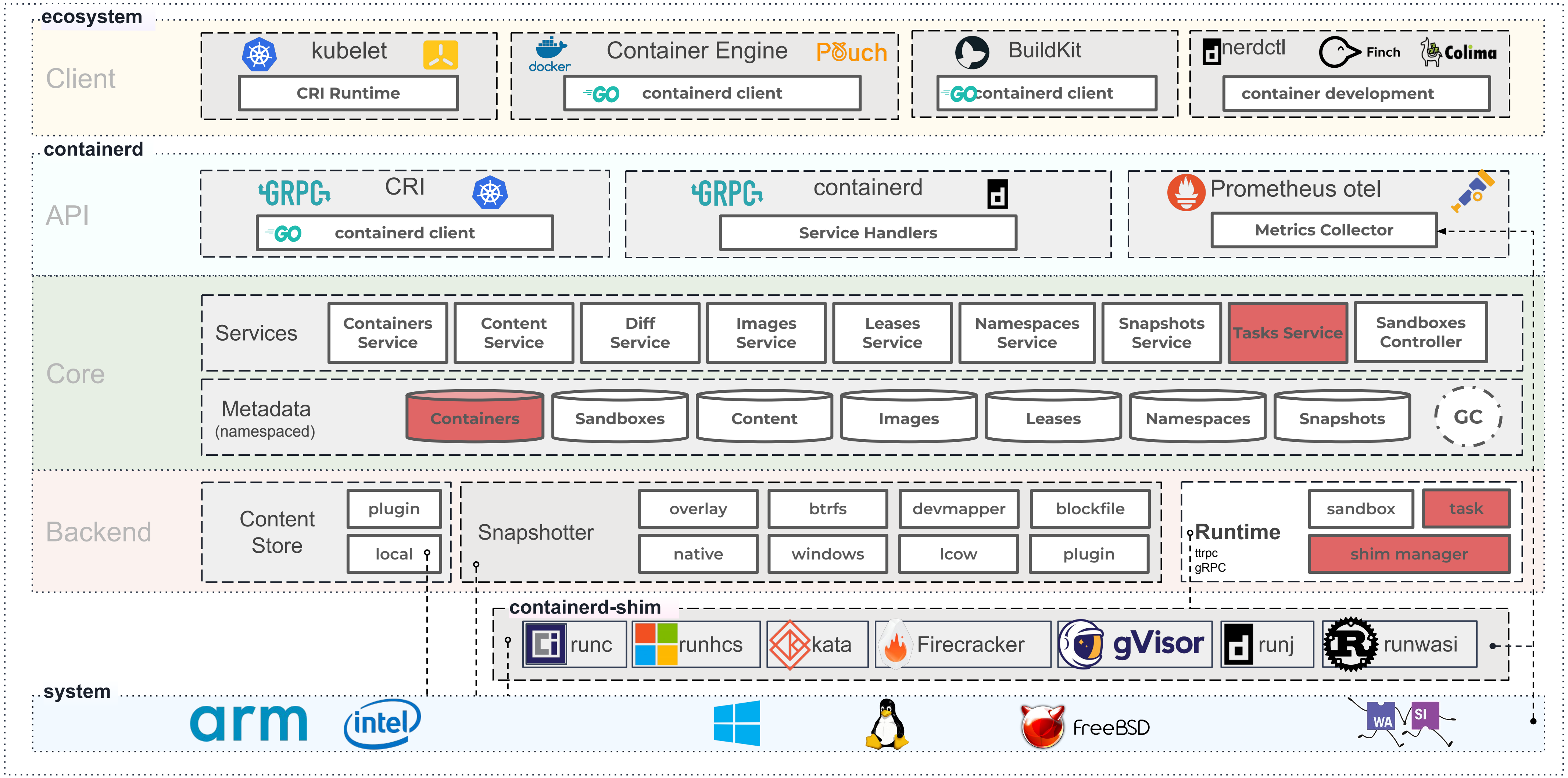
Contained 2.0



Sandbox & Container

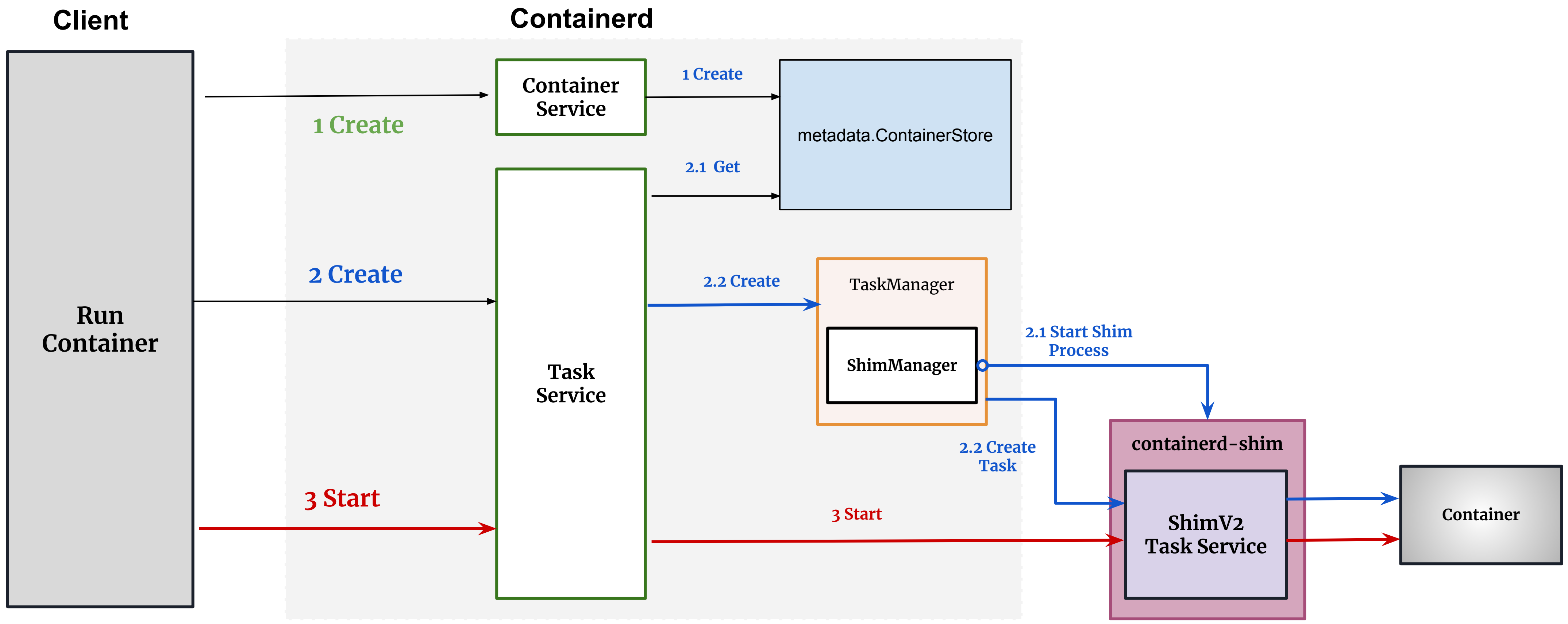


Container in Contained

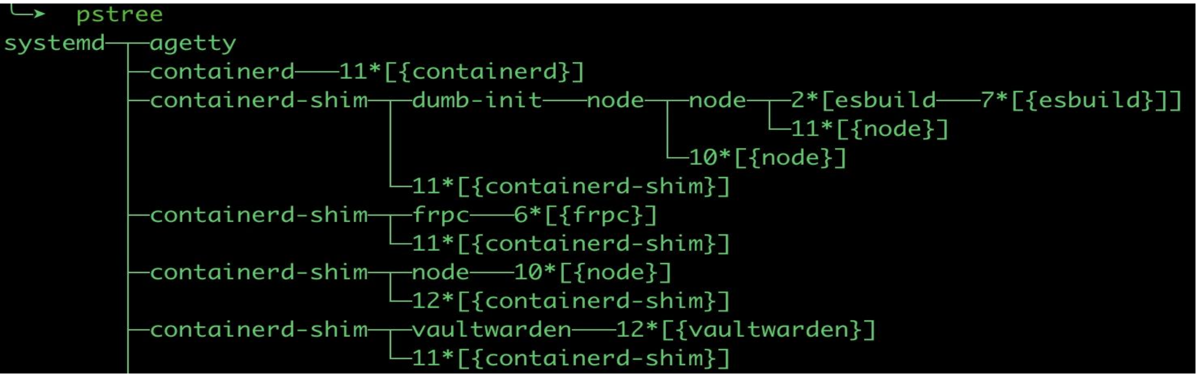


Container & Task

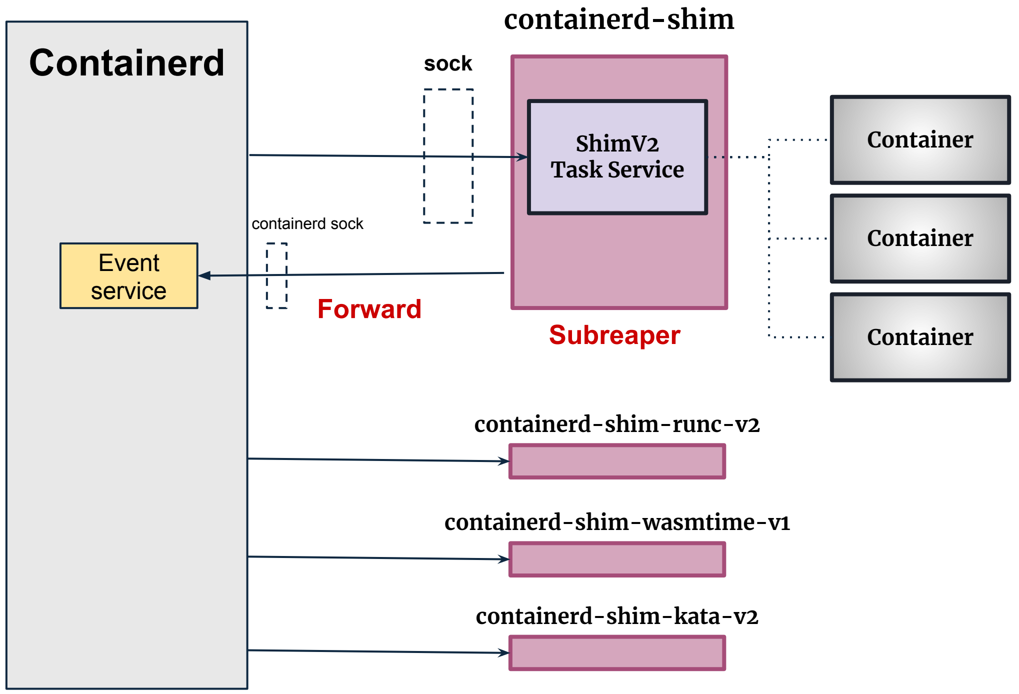
Run a Container = Create Container + Create Task + Start Task



Task & Shim



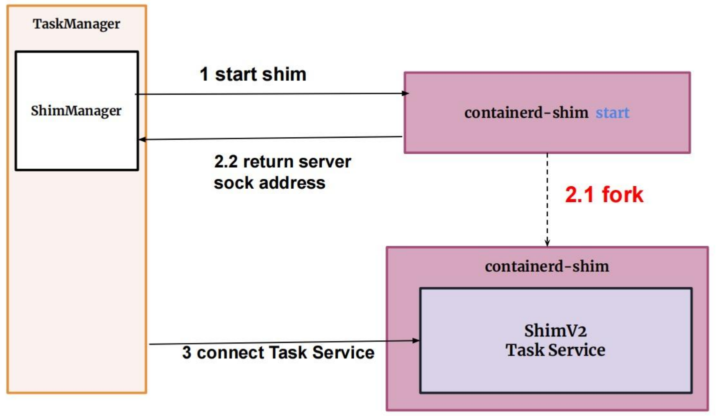
```
└─➤ pstree
systemd─┬─agetty
        ├─containerd───11*[{containerd}]
        ├─containerd-shim─┬─dumb-init───node─┬─node─┬─2*[esbuild───7*[{esbuild}]]
        │                 │                  │      └─11*[{node}]
        │                 │                  └─10*[{node}]
        │                 └─11*[{containerd-shim}]
        ├─containerd-shim─┬─frpc───6*[{frpc}]
        │                 └─11*[{containerd-shim}]
        ├─containerd-shim─┬─node───10*[{node}]
        │                 └─12*[{containerd-shim}]
        ├─containerd-shim─┬─vaultwarden───12*[{vaultwarden}]
        │                 └─11*[{containerd-shim}]
```



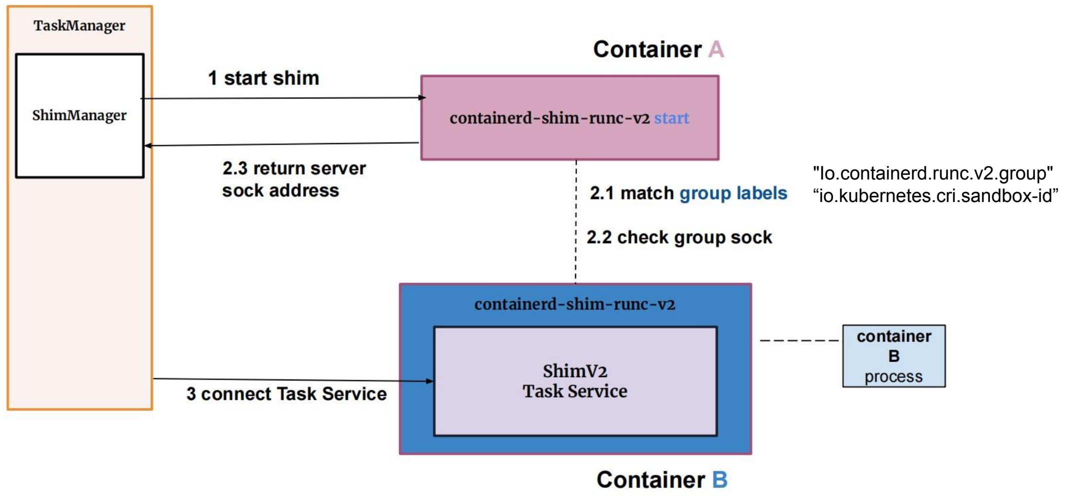
```
service Task {
        rpc State(StateRequest) returns (StateResponse);
        rpc Create(CreateTaskRequest) returns (CreateTaskResponse);
        rpc Start(StartRequest) returns (StartResponse);
        rpc Delete(DeleteRequest) returns (DeleteResponse);
        rpc Pids(PidsRequest) returns (PidsResponse);
        rpc Pause(PauseRequest) returns (google.protobuf.Empty);
        rpc Resume(ResumeRequest) returns (google.protobuf.Empty);
        rpc Checkpoint(CheckpointTaskRequest) returns
(google.protobuf.Empty);
        rpc Kill(KillRequest) returns (google.protobuf.Empty);
        rpc Exec(ExecProcessRequest) returns (google.protobuf.Empty);
        rpc ResizePty(ResizePtyRequest) returns (google.protobuf.Empty);
        rpc CloseIO(CloseIORequest) returns (google.protobuf.Empty);
        rpc Update(UpdateTaskRequest) returns (google.protobuf.Empty);
        rpc Wait(WaitRequest) returns (WaitResponse);
        rpc Stats(StatsRequest) returns (StatsResponse);
        rpc Connect(ConnectRequest) returns (ConnectResponse);
        rpc Shutdown(ShutdownRequest) returns (google.protobuf.Empty);
}
```

Shim & Container Group

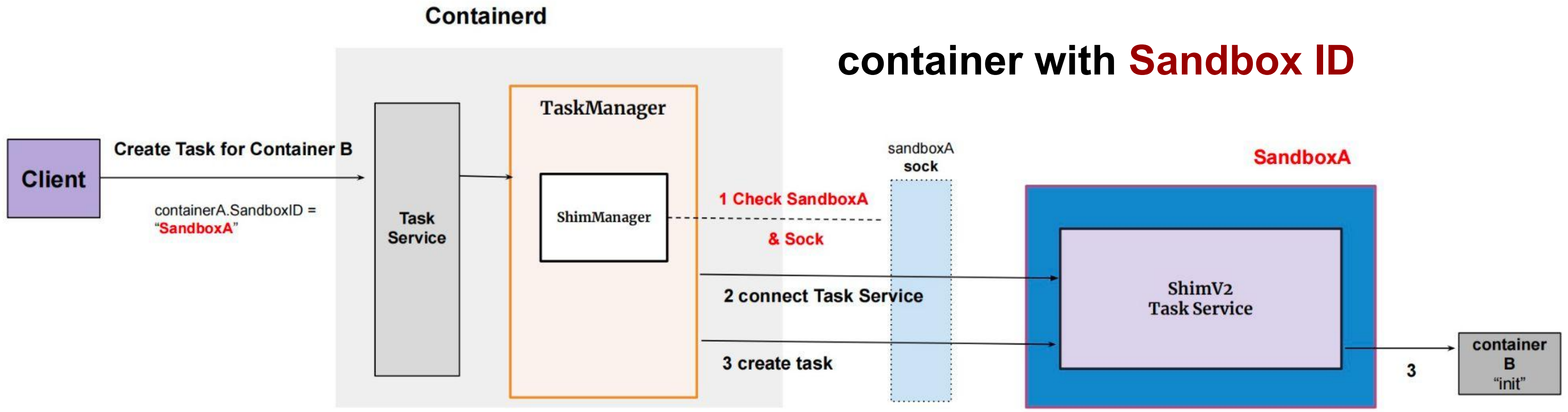
Shim Start



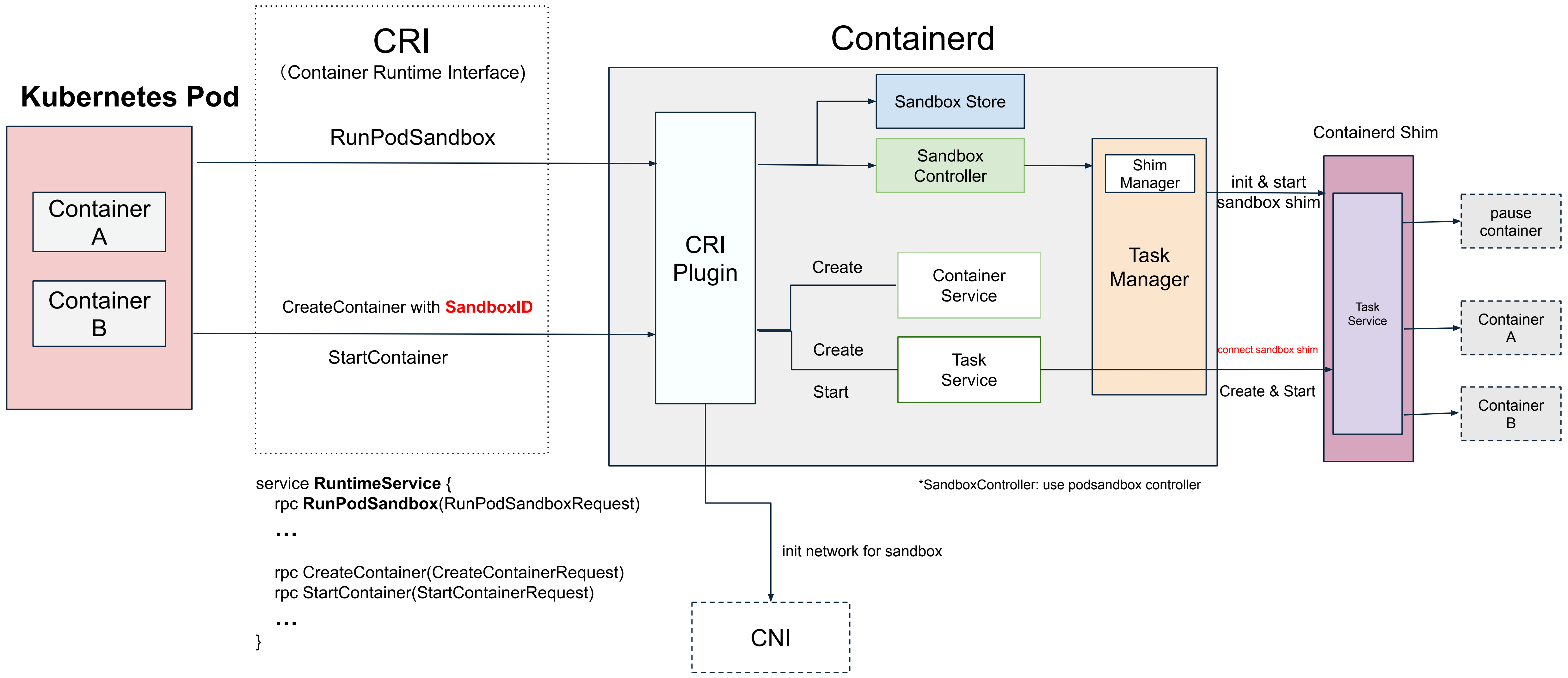
container with group labels



container with Sandbox ID



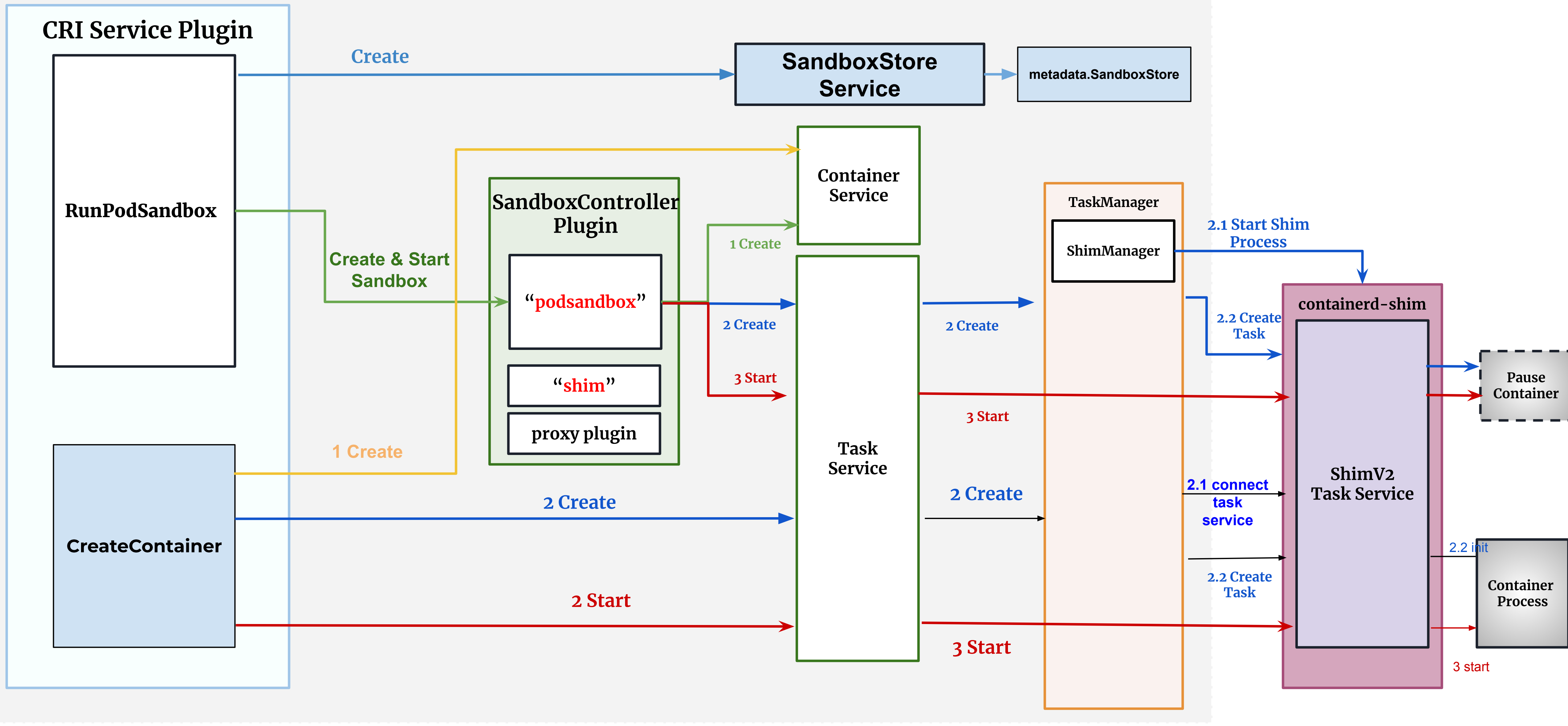
Container Group & Sandbox



*SandboxController: use podsandbox controller

```
service **RuntimeService** {
  rpc **RunPodSandbox**(RunPodSandboxRequest)
  …

  rpc CreateContainer(CreateContainerRequest)
  rpc StartContainer(StartContainerRequest)
  …
}
```

Sandbox & Sandbox API



Sandbox API

Sandbox Controller

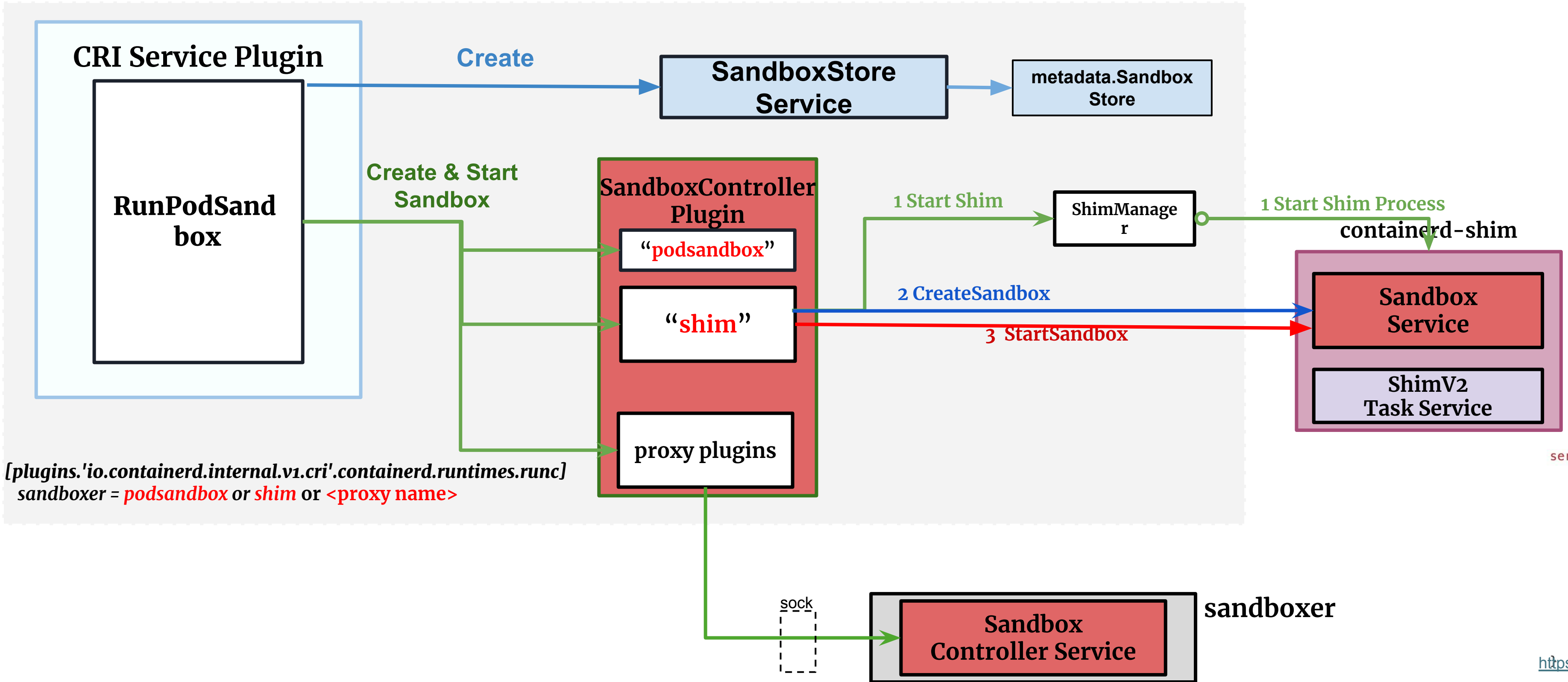
https://github.com/containerd/containerd/blob/main/core/sandbox/controller.go

```go
// Controller is an interface to manage sandboxes at runtime.
// When running in sandbox mode, shim expected to implement `SandboxService`.
// Shim lifetimes are now managed manually via sandbox API by the containerd's client.
type Controller interface {
        // Create is used to initialize sandbox environment. (mounts, any)
        Create(ctx context.Context, sandboxInfo Sandbox, opts ...CreateOpt) error
        // Start will start previously created sandbox.
        Start(ctx context.Context, sandboxID string) (ControllerInstance, error)
        // Platform returns target sandbox OS that will be used by Controller.
        // containerd will rely on this to generate proper OCI spec.
        Platform(_ctx context.Context, _sandboxID string) (imagespec.Platform, error)
        // Stop will stop sandbox instance
        Stop(ctx context.Context, sandboxID string, opts ...StopOpt) error
        // Wait blocks until sandbox process exits.
        Wait(ctx context.Context, sandboxID string) (ExitStatus, error)
        // Status will query sandbox process status. It is heavier than Ping call and must be used whenever you need to
        // gather metadata about current sandbox state (status, uptime, resource use, etc).
        Status(ctx context.Context, sandboxID string, verbose bool) (ControllerStatus, error)
        // Shutdown deletes and cleans all tasks and sandbox instance.
        Shutdown(ctx context.Context, sandboxID string) error
        // Metrics queries the sandbox for metrics.
        Metrics(ctx context.Context, sandboxID string) (*types.Metric, error)
        // Update changes a part of sandbox, such as extensions/annotations/labels/spec of
        // Sandbox object, controllers may have to update the running sandbox according to the changes.
        Update(ctx context.Context, sandboxID string, sandbox Sandbox, fields ...string) error
}
```

Runtime Sandbox Service

```protobuf
service Sandbox {
        // CreateSandbox will be called right after sandbox shim instance launched.
        // It is a good place to initialize sandbox environment.
        rpc CreateSandbox(CreateSandboxRequest) returns (CreateSandboxResponse);

        // StartSandbox will start a previously created sandbox.
        rpc StartSandbox(StartSandboxRequest) returns (StartSandboxResponse);

        // Platform queries the platform the sandbox is going to run containers on.
        // containerd will use this to generate a proper OCI spec.
        rpc Platform(PlatformRequest) returns (PlatformResponse);

        // StopSandbox will stop existing sandbox instance
        rpc StopSandbox(StopSandboxRequest) returns (StopSandboxResponse);

        // WaitSandbox blocks until sandbox exits.
        rpc WaitSandbox(WaitSandboxRequest) returns (WaitSandboxResponse);

        // SandboxStatus will return current status of the running sandbox instance
        rpc SandboxStatus(SandboxStatusRequest) returns (SandboxStatusResponse);

        // PingSandbox is a lightweight API call to check whether sandbox alive.
        rpc PingSandbox(PingRequest) returns (PingResponse);

        // ShutdownSandbox must shutdown shim instance.
        rpc ShutdownSandbox(ShutdownSandboxRequest) returns (ShutdownSandboxResponse);

        // SandboxMetrics retrieves metrics about a sandbox instance.
        rpc SandboxMetrics(SandboxMetricsRequest) returns (SandboxMetricsResponse);
}
```

https://github.com/containerd/containerd/blob/main/api/runtime/sandbox/v1/sandbox.proto

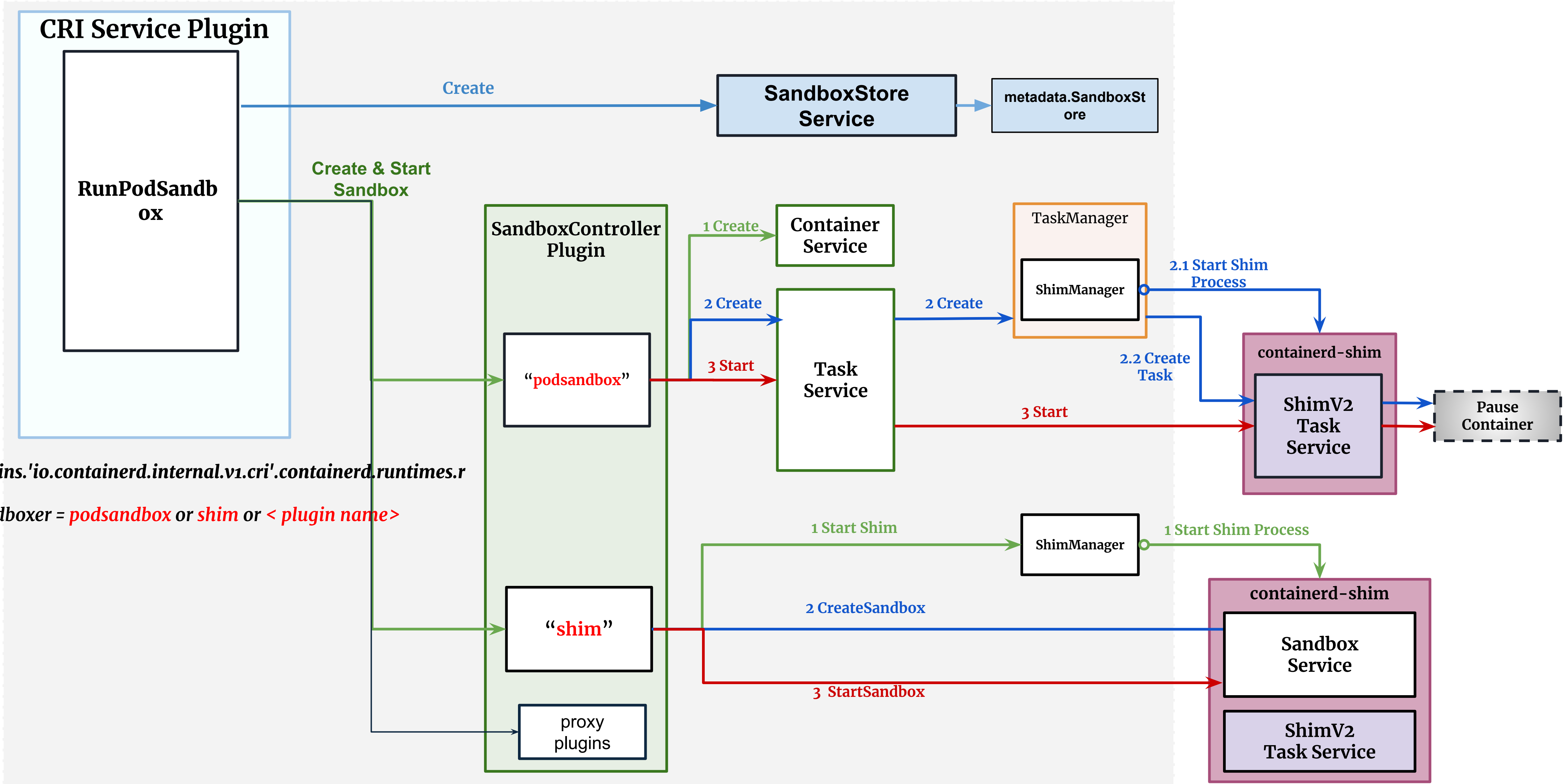


Sandbox Controller Service

```protobuf
service Controller {
        rpc Create(ControllerCreateRequest) returns (ControllerCreateResponse);
        rpc Start(ControllerStartRequest) returns (ControllerStartResponse);
        rpc Platform(ControllerPlatformRequest) returns (ControllerPlatformResponse);
        rpc Stop(ControllerStopRequest) returns (ControllerStopResponse);
        rpc Wait(ControllerWaitRequest) returns (ControllerWaitResponse);
        rpc Status(ControllerStatusRequest) returns (ControllerStatusResponse);
        rpc Shutdown(ControllerShutdownRequest) returns (ControllerShutdownResponse);
        rpc Metrics(ControllerMetricsRequest) returns (ControllerMetricsResponse);
        rpc Update(ControllerUpdateRequest) returns (ControllerUpdateResponse);
```

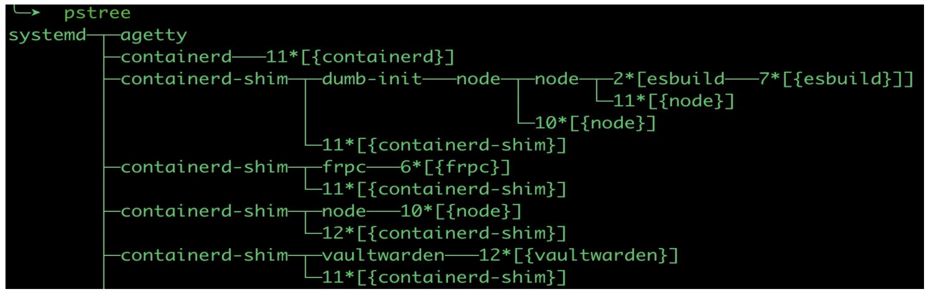
https://github.com/containerd/containerd/blob/main/api/services/sandbox/v1/sandbox.proto

Sandbox API & Shim

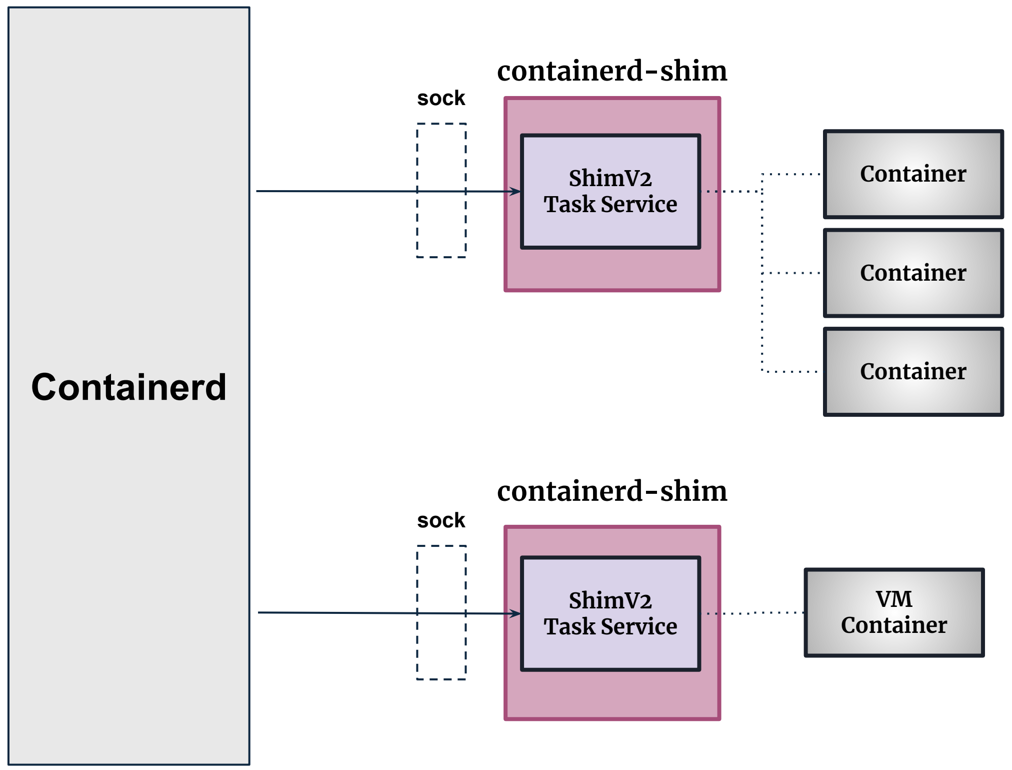


[plugins.'io.containerd.internal.v1.cri'.containerd.runtimes.runc]
sandboxer = podsandbox or shim or < plugin name>

Do we have to have Shim?

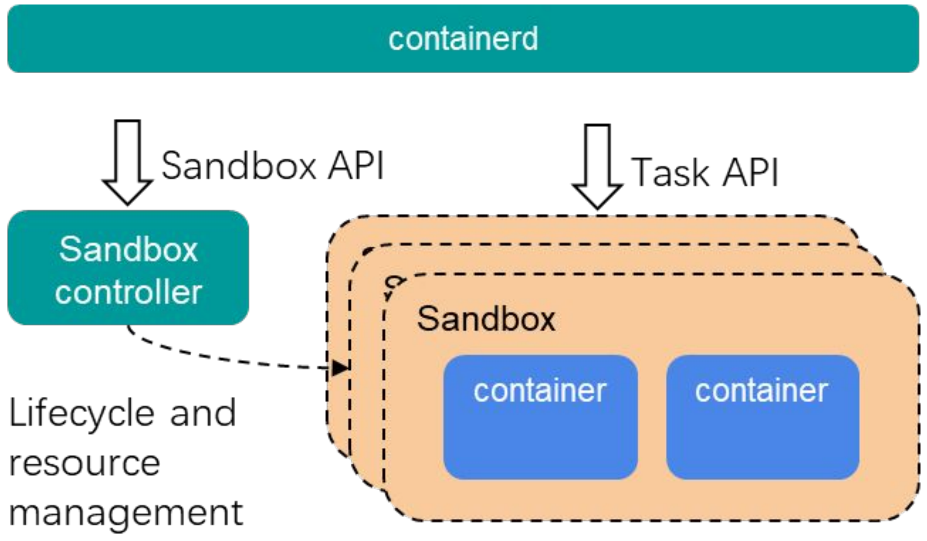


```
└─➤ pstree
systemd─┬─agetty
        ├─containerd───11*[{containerd}]
        ├─containerd-shim─┬─dumb-init───node─┬─node─┬─2*[esbuild───7*[{esbuild}]]
        │                 │                  │      └─11*[{node}]
        │                 │                  └─10*[{node}]
        │                 └─11*[{containerd-shim}]
        ├─containerd-shim─┬─frpc───6*[{frpc}]
        │                 └─11*[{containerd-shim}]
        ├─containerd-shim─┬─node───10*[{node}]
        │                 └─12*[{containerd-shim}]
        ├─containerd-shim─┬─vaultwarden───12*[{vaultwarden}]
        │                 └─11*[{containerd-shim}]
```



Sandbox In Containerd

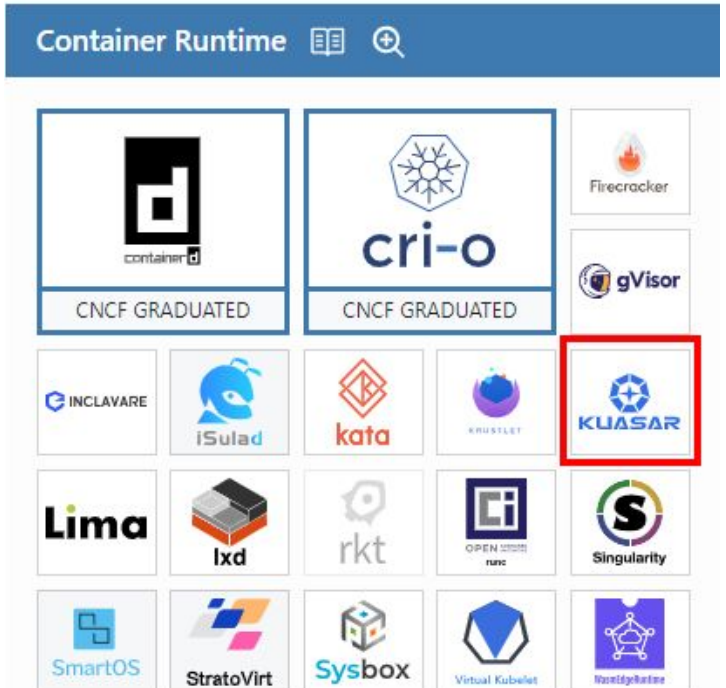
- An isolated environment to run Tasks(containers) in it.
  - It can be a black box
  - It exposes **Task API** (through uds/vsock/tcp...).
  - It can be highly integrated(one process/thread per sandbox).
  - It can be managed by **Sandbox Controller API**.

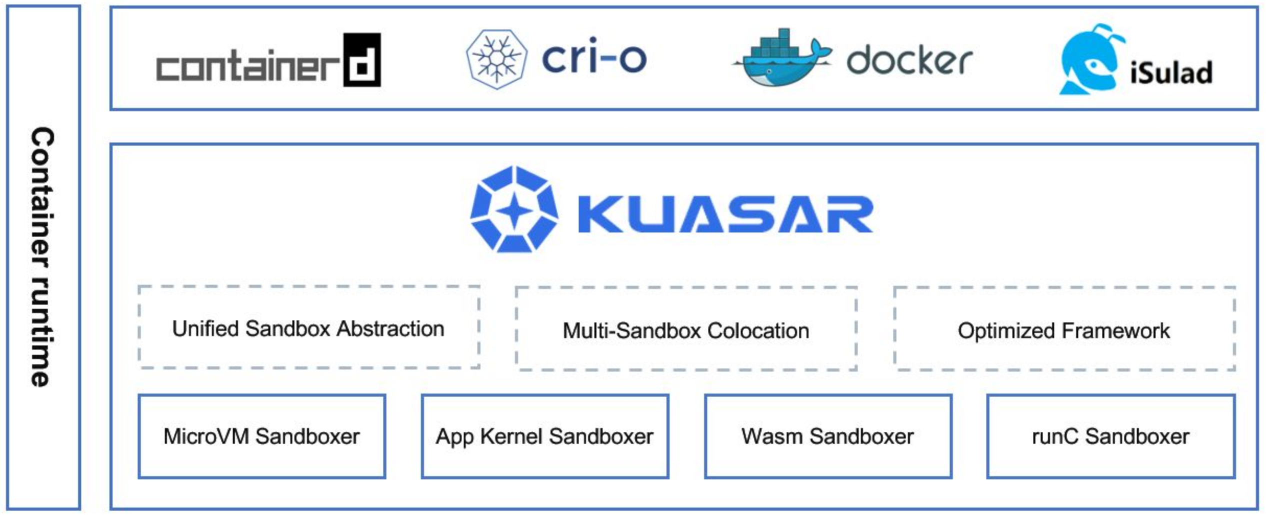


Kuasar: A CNCF Sandbox Project

A low-level container runtime that provides different kind of sandbox controllers(or sandboxers).

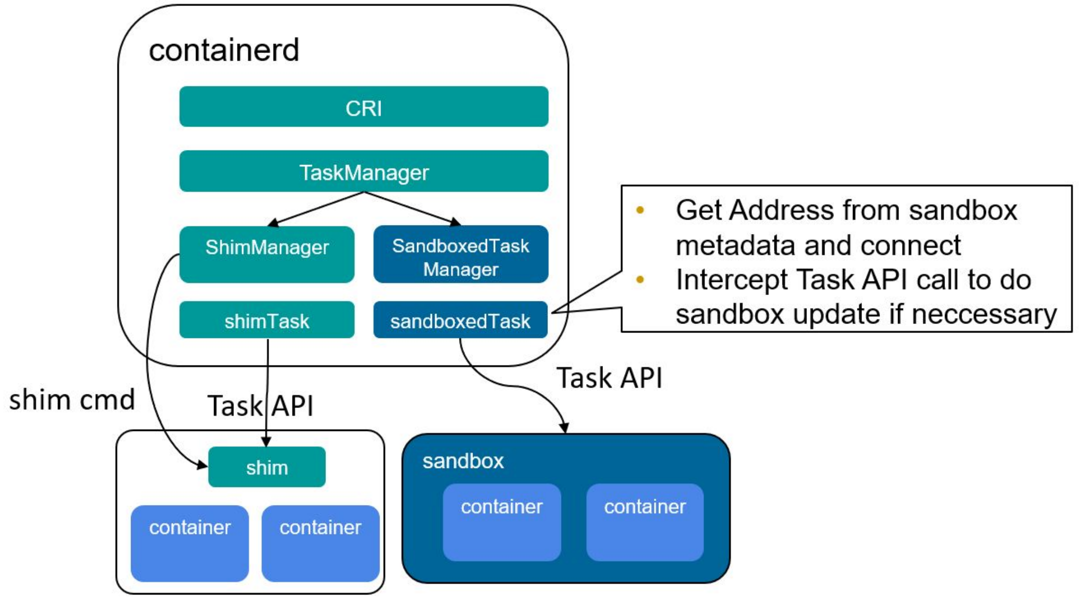
- A pure rust framework to provide sandboxer implementations to containerd
- A set of sandboxers based on the framework: MicroVM/App Kernel/WebAssembly/runC…





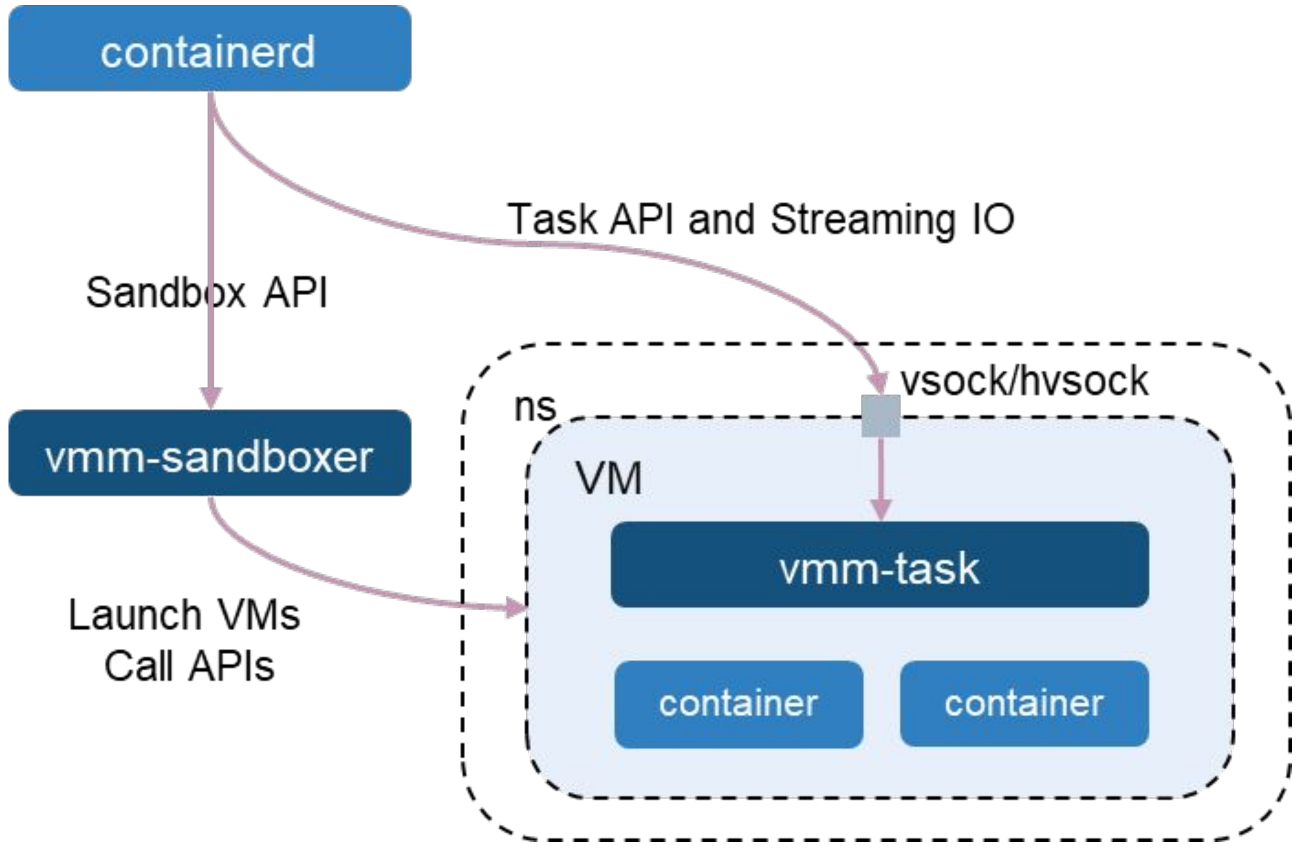
Kuasar and Containerd

- Kuasar is the first runtime implemented with proxy plugin of sandbox controller.
- 17 PRs submitted to containerd, to improve sandbox API and framework
  - Decouple PodSandbox(pause container) controller and cri plugin
  - Decouple shim plugin and task plugin
  - Add Endpoint address of sandbox
  - Support Vsock connection to task api
  - Support io by Streaming API
  - Retry for wait to remote sandbox controller
- 4 PRs is expecting to be merged
  - Sandboxed task
  - Call Update API in container lifecycle



Kuasar-vmm

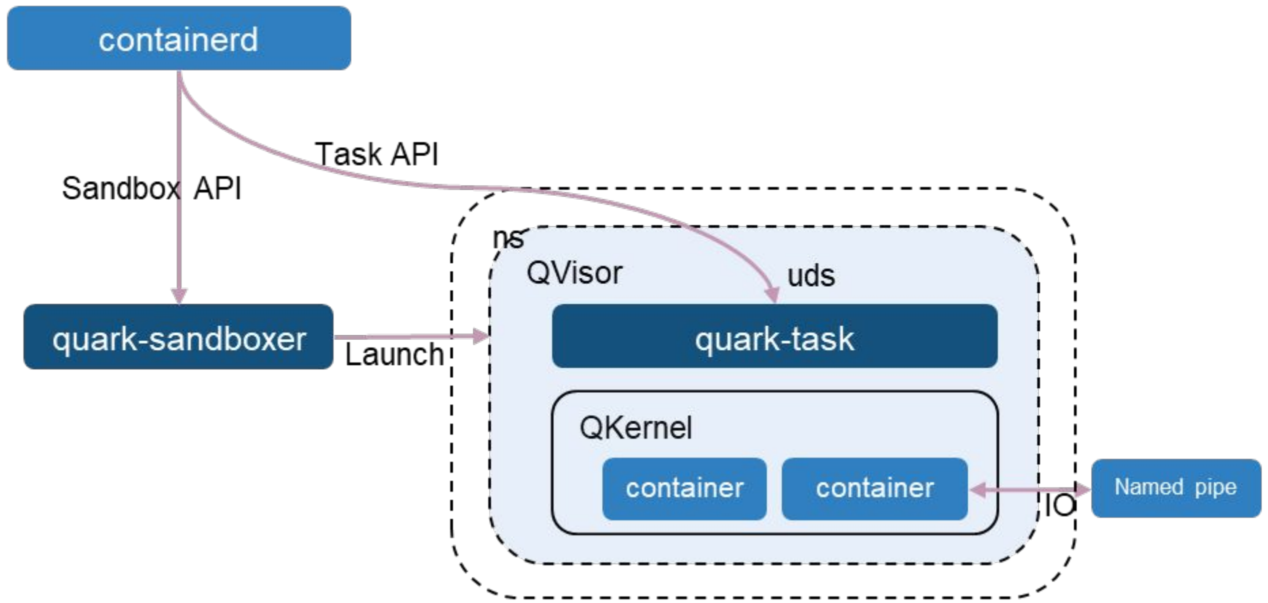
- **Vmm-sandboxer** focus on the vm lifecycle and resource management.
  - Support launch vm by cloud-hypervisor/qemu/stratovirt.
- **Task Service** is running inside the VM as the PID 1 process.
  - Containerd connect it with vsock/hvsock
- **Streaming IO**: IO stream transferred by the Streaming API.
- **No shim process**
  - Arch and maintenance is simplified.
  - Performance is improved.



* Containerd is a forked version as Update API is not merged

Kuasar-quark

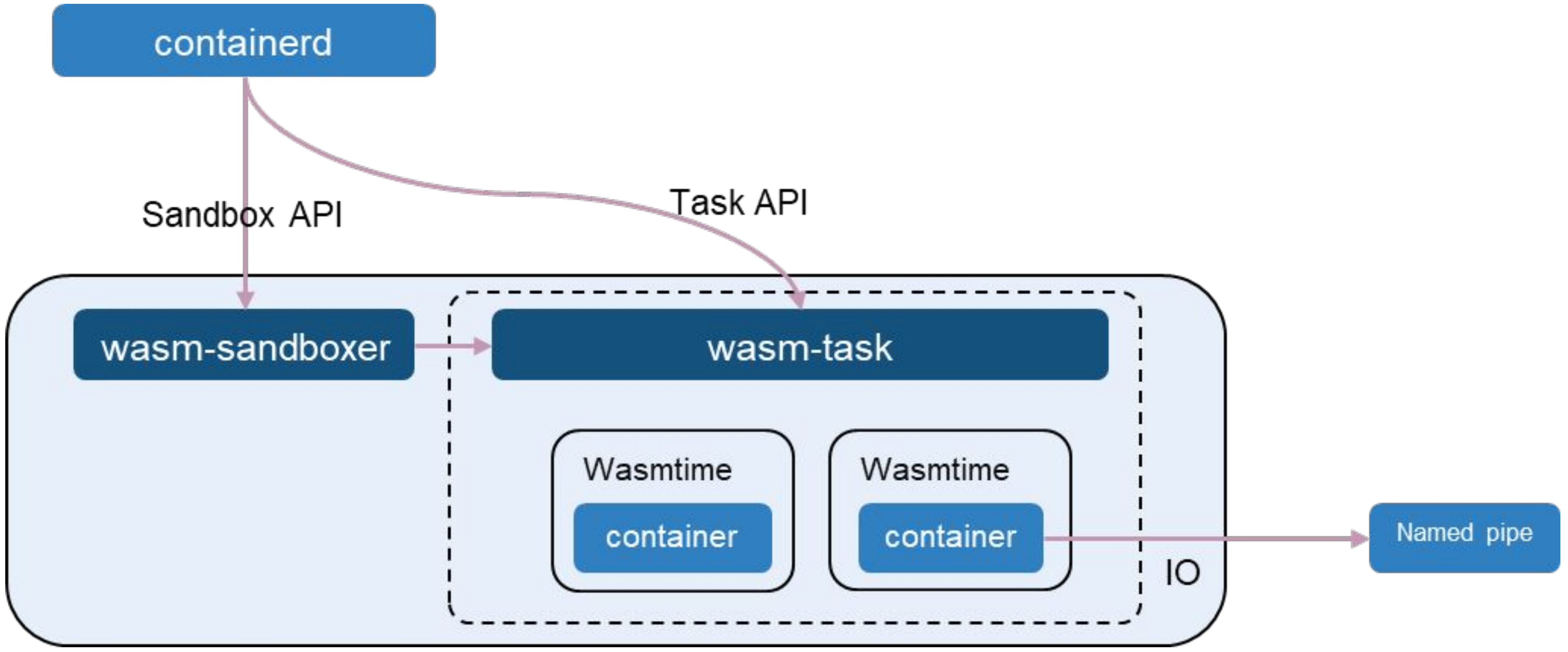
- **Quark:** Application kernel sandbox.
  - Hypervisor(Qvisor) and kernel(Qkernel) rewritten in rust.
  - Secure container with kernel isolation.
  - Significant performance.
- **Task Service** is directly running inside the Qvisor.
  - No extra shim process.
- **No Shim process**



* Containerd is a forked version as Update API is not merged

Kuasar-wasm

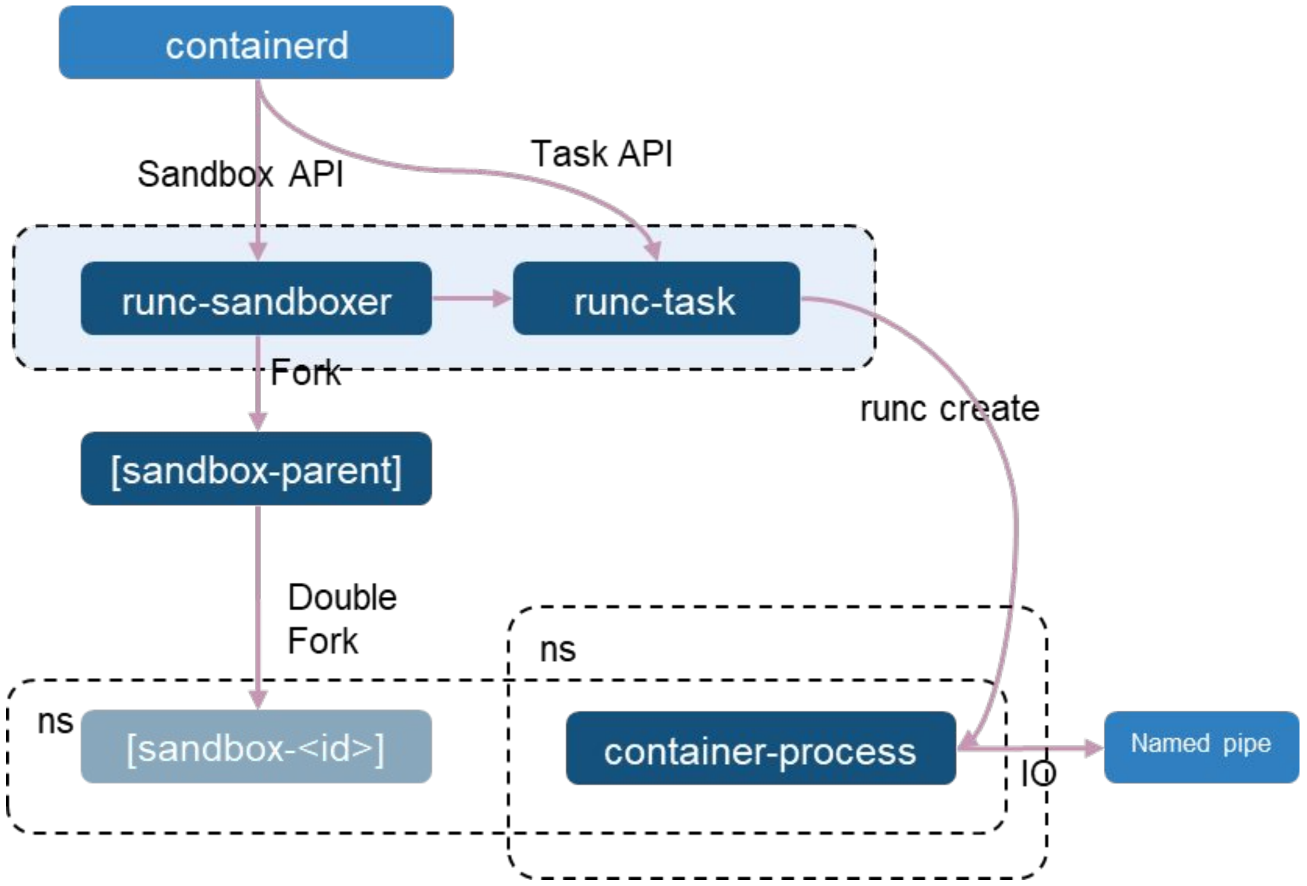
- All in a single process.
  - Low start latency for wasm app.
  - Low overhead for each container.
- Wasmtime/WasmEdge supported
- **No shim process**



* Containerd is a forked version as Update API is not merged

Kuasar-runc

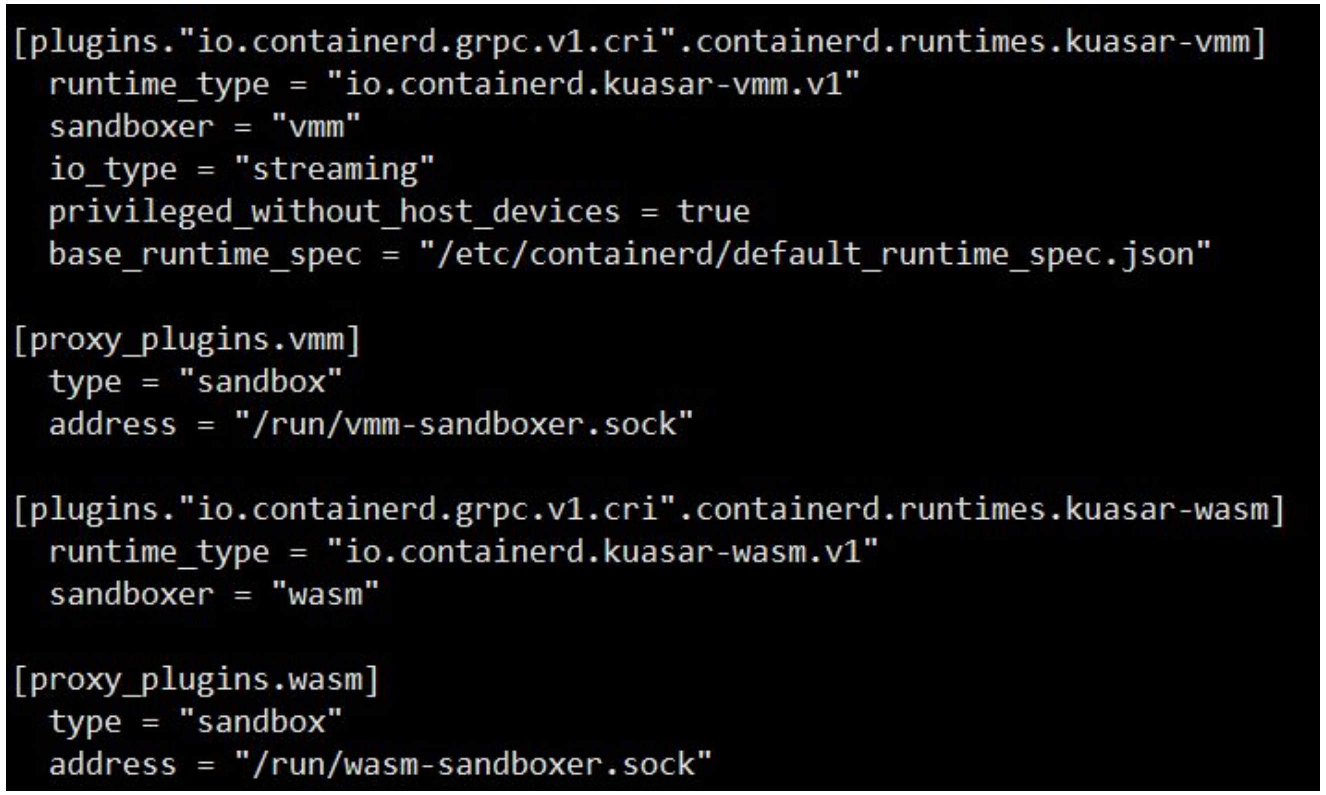
- **Sandbox process:** forked by an empty sandbox parent.
  - Exist only when shareProcessNamespace.
  - Memory overhead smaller than 100k.
- Task Service forked by Sandboxer
  - Fork a new task service everytime sandboxer restart.
  - The task service only exit when all containers it manages is removed.
- **No shim process**.
  - Only runc created containers



* Containerd is a forked version as Update API is not merged

How to use

- 1. Start sandboxer
  - /usr/local/bin/vmm-sandboxer --listen /run/vmm-sandboxer.sock --dir /run/kuasar-vmm
- 2. Configure the /etc/containerd/config.toml
- 3. Start the sandbox with --runtime or create RuntimeClass in kubernetes

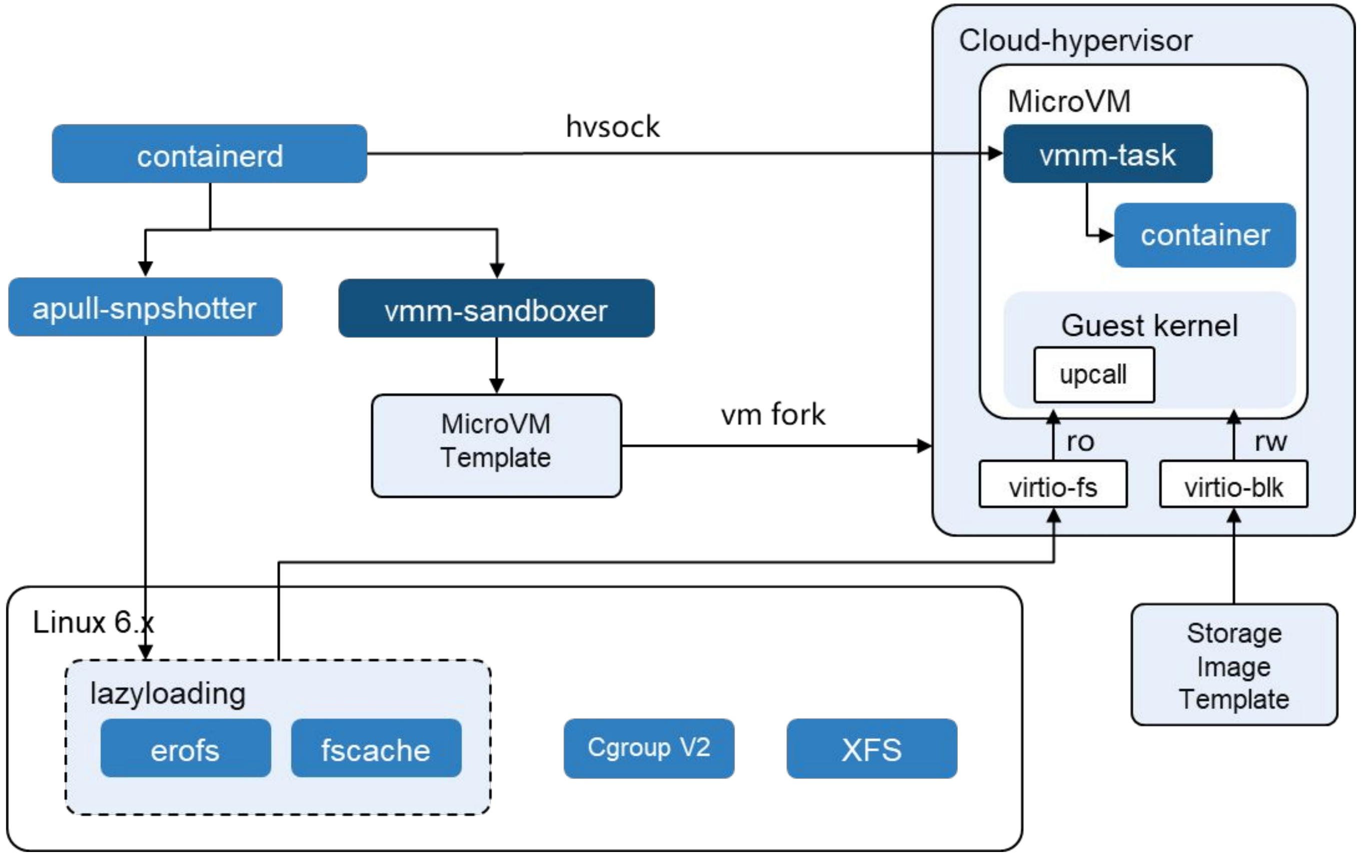


```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.kuasar-vmm]
  runtime_type = "io.containerd.kuasar-vmm.v1"
  sandboxer = "vmm"
  io_type = "streaming"
  privileged_without_host_devices = true
  base_runtime_spec = "/etc/containerd/default_runtime_spec.json"

[proxy_plugins.vmm]
  type = "sandbox"
  address = "/run/vmm-sandboxer.sock"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.kuasar-wasm]
  runtime_type = "io.containerd.kuasar-wasm.v1"
  sandboxer = "wasm"

[proxy_plugins.wasm]
  type = "sandbox"
  address = "/run/wasm-sandboxer.sock"
```

Use Case 1: Lightweight Secure Container

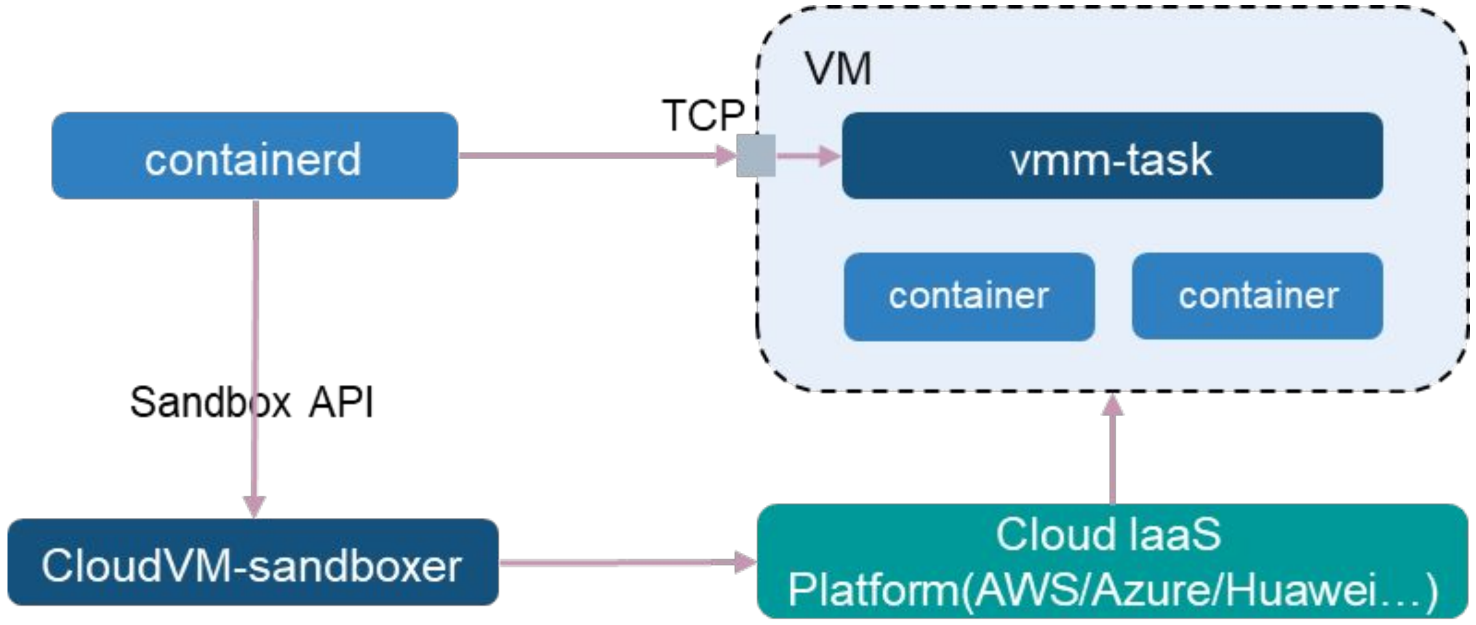
- 1. VM Template based on mmap and copy-on-write
- 2. Condensed and pre-patched guest
- 3. Upcall in guest to remove acpi
- 4. Parallel cgroup creation

Memory overhead: **100MB → 17MB**
Start latency: **850ms → 12ms**



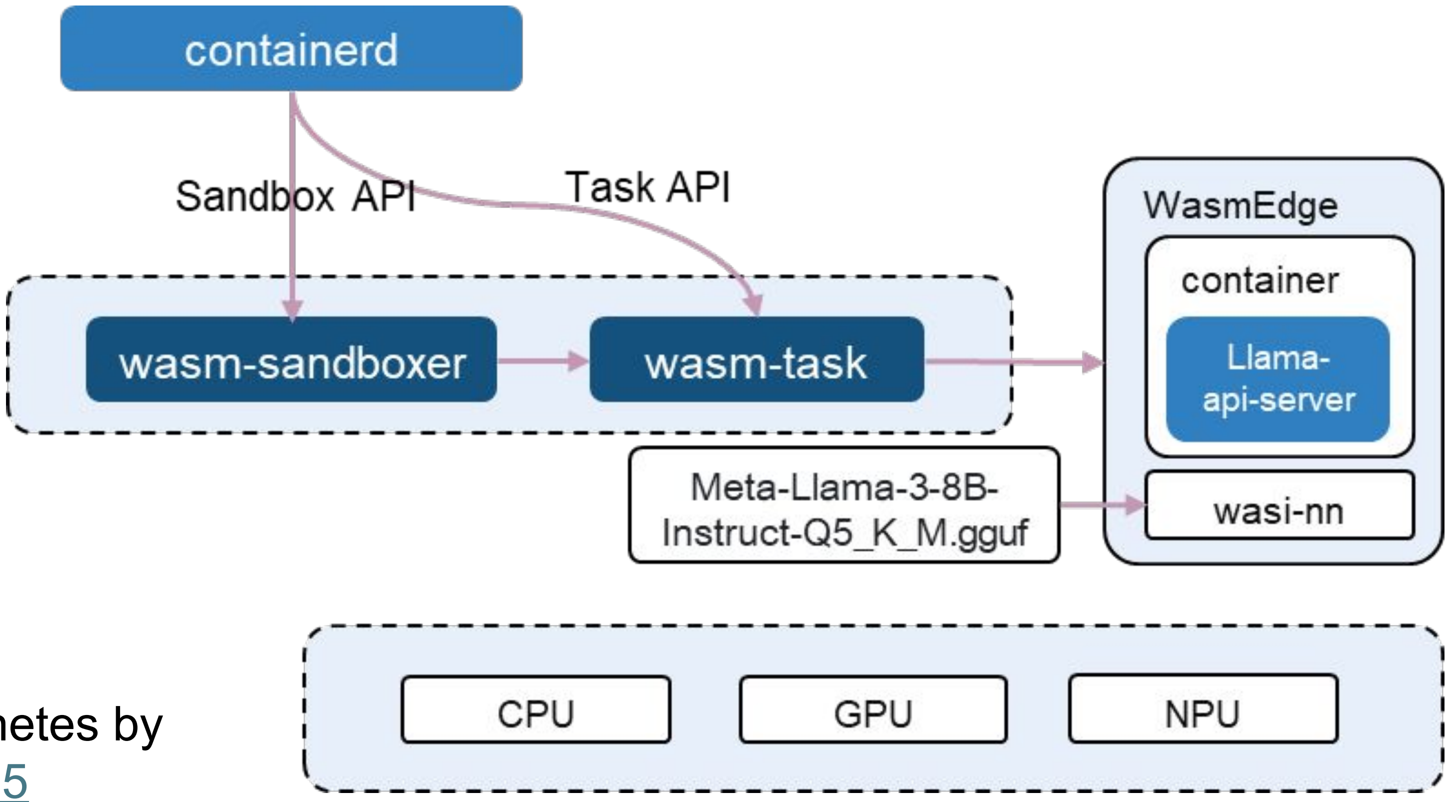
Use Case 2: Remote sandbox

- 1. Remote VM as a sandbox
- 2. Call IaaS API to create/delete VM
- 3. Task API and Streaming IO by TCP
- 4. Make a unified platform for VM and Serverless Container



Use Case 3: LLM running in wasm

- 1. Fast and lightweight deploy
- 2. Cross cpu/gpu and OSes
- 3. Zero python dependency



Keynote: Deploying LLM Workloads on Kubernetes by WasmEdge and Kuasar: https://sched.co/1eYa5

https://github.com/kuasar-io/kuasar/blob/main/docs/wasm/How-to-run-Llama-3-8B-with-Kubernetes.md

Welcome to join us!

Containerd: https://github.com/containerd/containerd

Containerd community meeting:
https://docs.google.com/document/d/1Q8KyVJd26oAQ3MafbnkVBgJCopPl2Bw45H9L9br9Vus/edit#heading=h.bhv4ajd2ibx0

Kuasar: https://github.com/kuasar-io/kuasar

Kuasar homepage: https://kuasar.io/

Kuasar slack:
https://cloud-native.slack.com/archives/C052JRURD8V