KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

Extending containerd

Samuel Karp – @samuelkarp
Maksym Pavlenko – @mak_pav

Table of contents

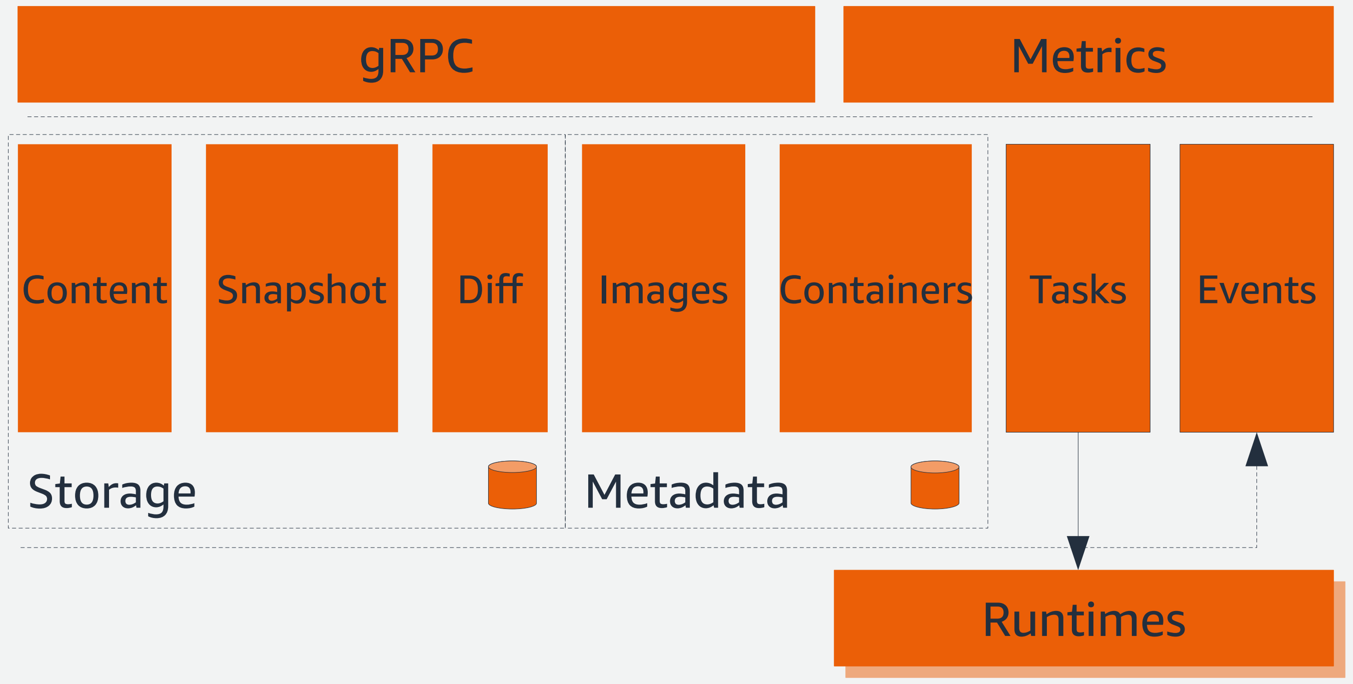
containerd

What is containerd?

- Small and focused container runtime
- Build on lessons from Docker
  - Strict scope to limit features
  - Modular, composable pieces

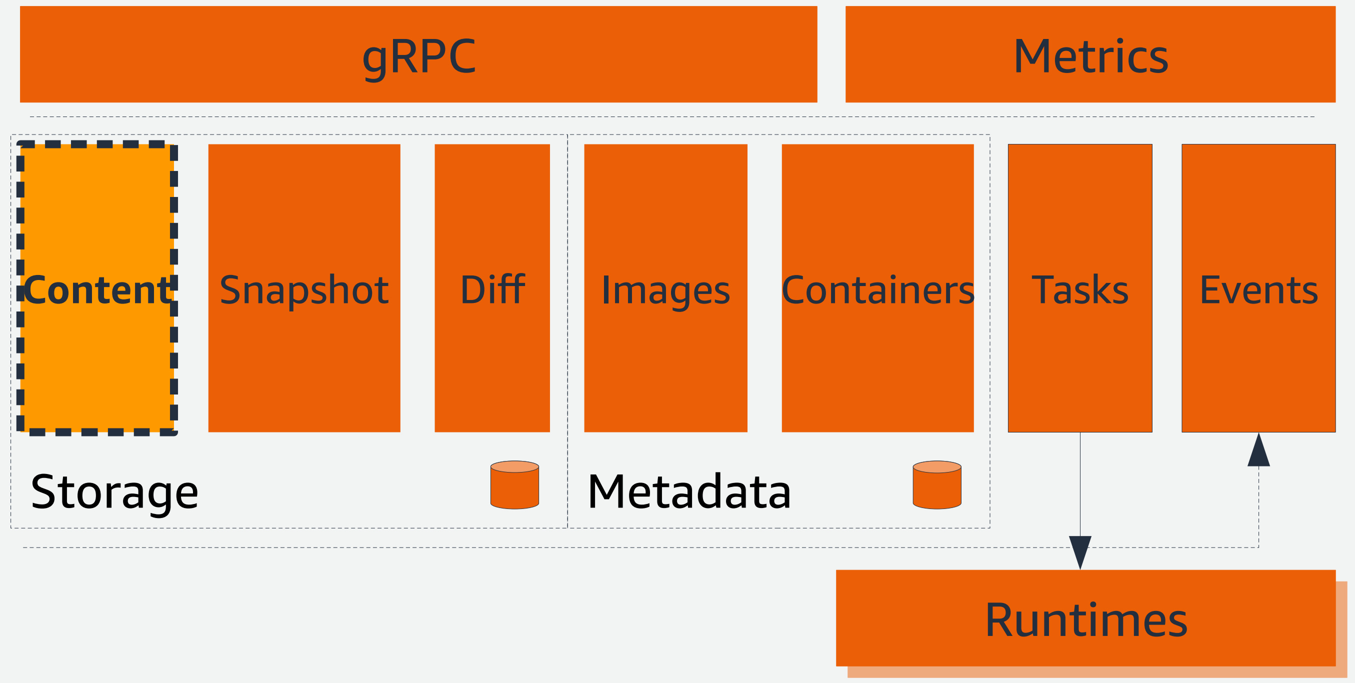
The containerd stack

- gRPC API and Services
- Storage services
  - Content store
  - Snapshotters
- Runtime (runc, OCI, v2)



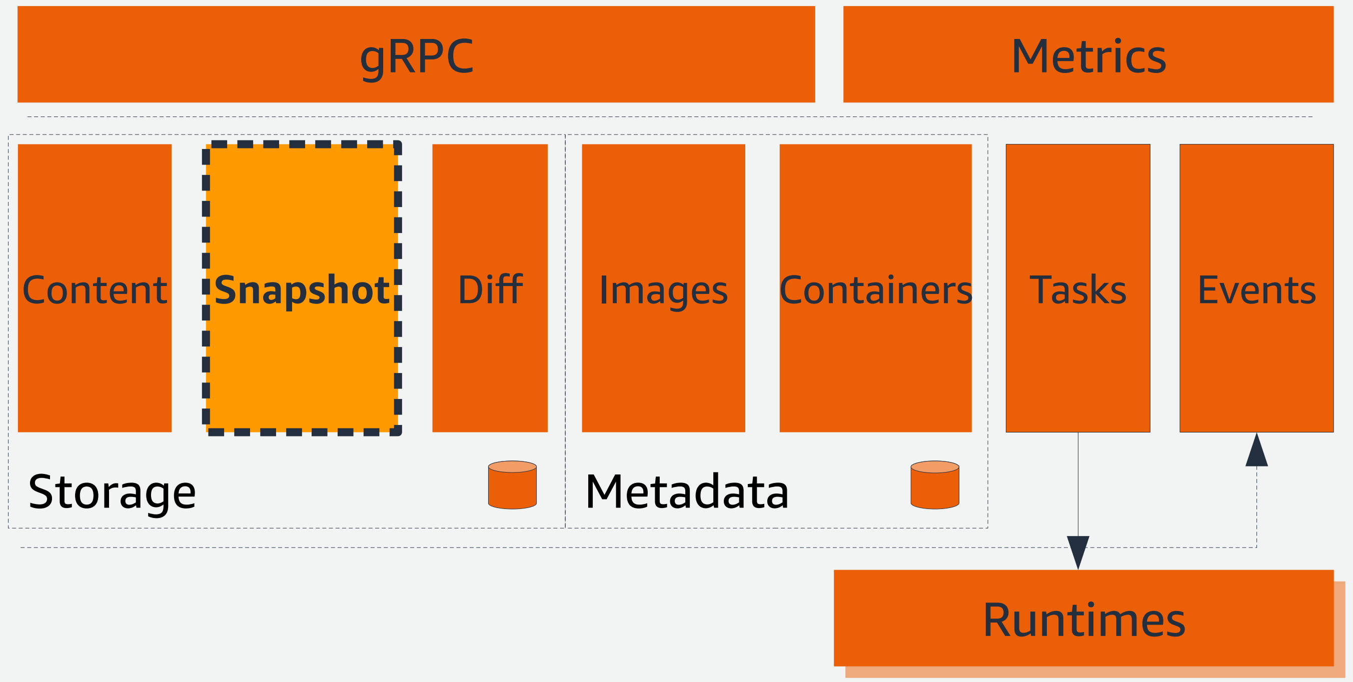
The containerd stack

- gRPC API and Services
- Storage services
  - Content store
  - Snapshotters
- Runtime (runc, OCI, v2)



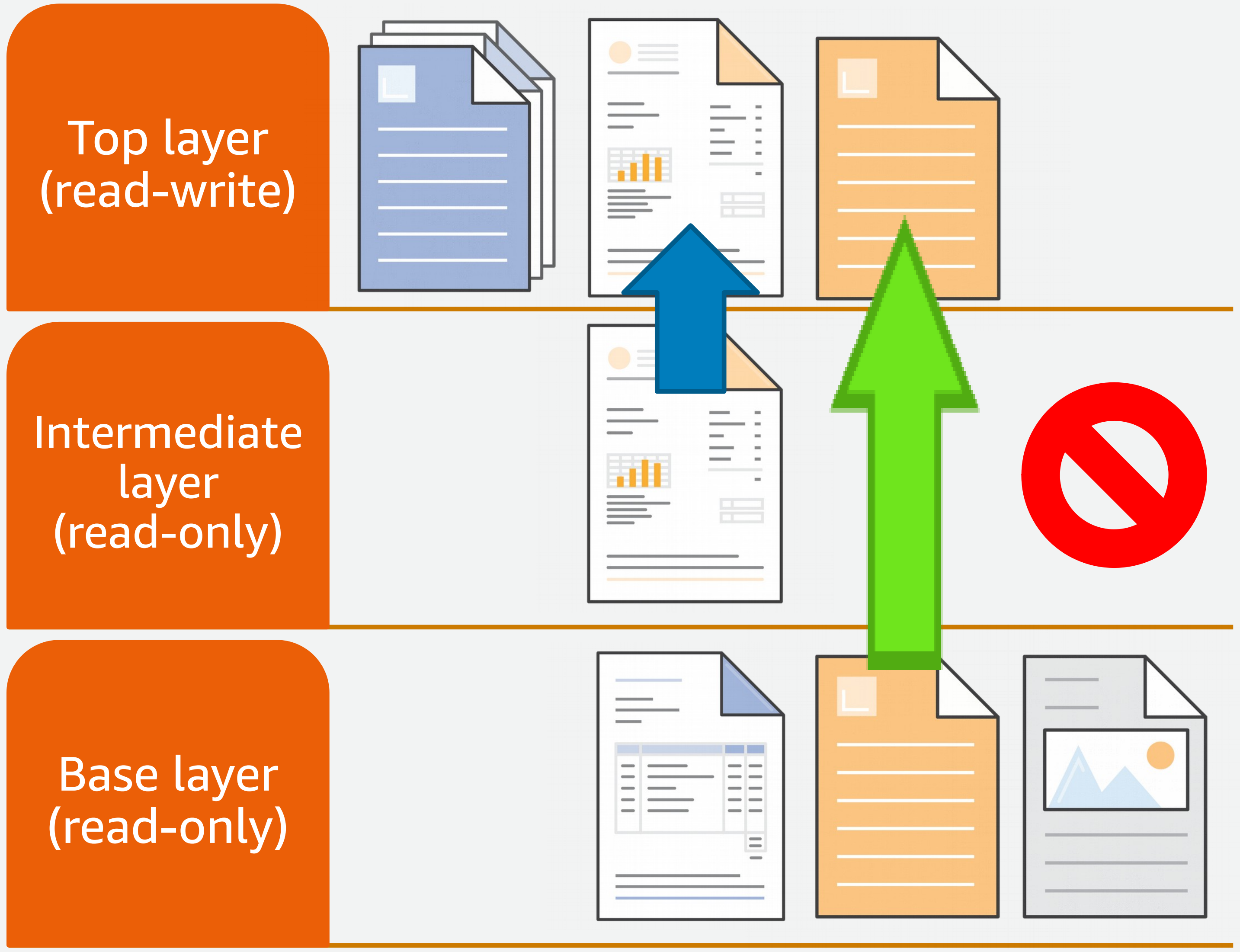
The containerd stack

- gRPC API and Services
- Storage services
  - Content store
  - Snapshotters
- Runtime (runc, OCI, v2)



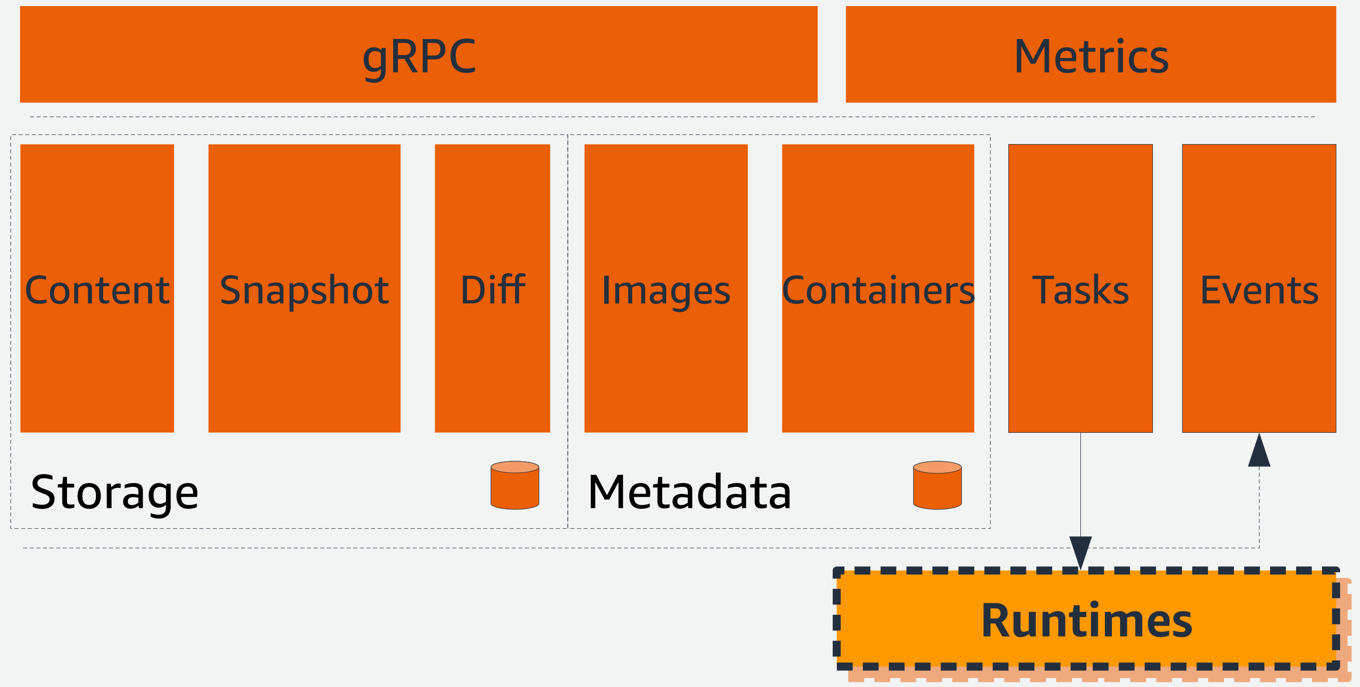
Container image layers

- A copy-on-write view of files
- New files exist in the top layer
- Modified files are "copied up"
- Unmodified files stay in original layer
- Deleted files are hidden, not removed



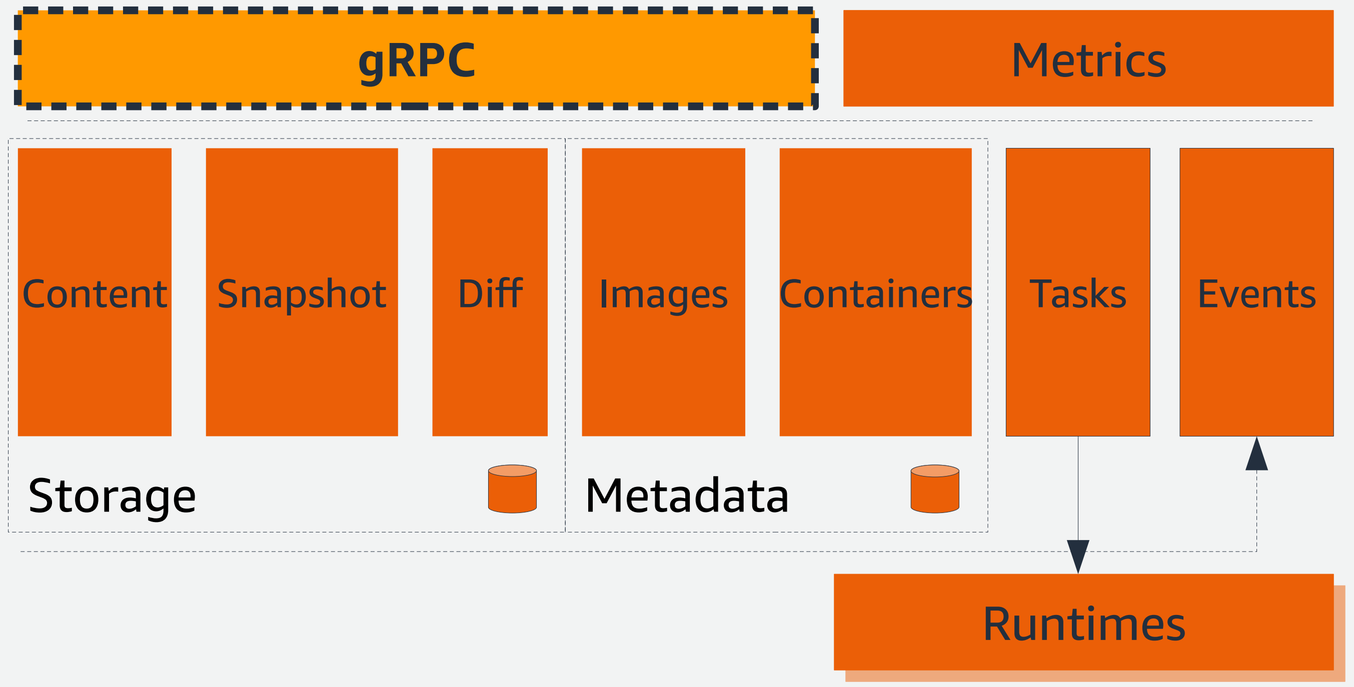
The containerd stack

- gRPC API and Services
- Storage services
  - Content store
  - Snapshotters
- Runtime (runc, OCI, v2)



The containerd stack

- gRPC API and Services
- Storage services
  - Content store
  - Snapshotters
- Runtime (runc, OCI, v2)



Core modularity

- Small, separate services
- Use services together for higher-level functionality
- Services modeled with interfaces
- Services are implemented as plugins
- Client library to tie it all together

Extension

containerd extension points

- Client library extensions
- “CLI”/executable plugins
- gRPC proxy plugins
- Go plugins
- Built-in plugins

Client library extensions

- “Smart” client in Go provides interfaces
- Write your own implementations when you want something different!
- Requires that you control the client code

- Examples
  - Pulling images
  - I/O handling for containers

Client library extensions – Pulling images

- Pulling images happens in the client library
- Network access and protocol support
- Default implementation is Docker registry

- Examples
  - Distributed/peer-to-peer protocol like BitTorrent
  - Other registry protocols like Amazon ECR
  - Maybe you want to store images in git-lfs?
  - Anything you can think of!

Client library extension – default resolver

```
img, err := client.Pull(
    namespaces.NamespaceFromEnv(ctx),
    "my.registry/myrepository:mytag",

    containerd.WithPullUnpack)
```

Client library extension – Amazon ECR resolver

```
// import "github.com/awslabs/amazon-ecr-containerd-resolver"
resolver, _ := ecr.NewResolver()
img, err := client.Pull(
    namespaces.NamespaceFromEnv(ctx),
    "ecr.aws/arn:aws:ecr:us-west-2:123456789012:repository/myrepository:mytag",
    containerd.WithResolver(resolver),
    containerd.WithPullUnpack)
```

Client library extension – Resolver interface

```
type Resolver interface {
    Resolve(ctx context.Context, ref string) (string, oci.Descriptor, error)
    Fetcher(ctx context.Context, ref string) (Fetcher, error)
    Pusher(ctx context.Context, ref string) (Pusher, error)
}
```

Client library extension – Resolver interface

```
type Resolver interface {
    Resolve(ctx context.Context, ref string) (string, oci.Descriptor, error)
    Fetcher(ctx context.Context, ref string) (Fetcher, error)
    Pusher(ctx context.Context, ref string) (Pusher, error)
}
type Fetcher interface {
    Fetch(ctx context.Context, desc oci.Descriptor) (io.ReadCloser, error)
}
```

Client library extension – Resolver interface

```
type Resolver interface {
    Resolve(ctx context.Context, ref string) (string, oci.Descriptor, error)
    Fetcher(ctx context.Context, ref string) (Fetcher, error)
    Pusher(ctx context.Context, ref string) (Pusher, error)
}
type Fetcher interface {
    Fetch(ctx context.Context, desc oci.Descriptor) (io.ReadCloser, error)
}
type Pusher interface {
    Push(ctx context.Context, desc oci.Descriptor) (content.Writer, error)
}
```

“CLI”/executable plugins

- Command-line interface conventions
- Separate program from containerd
- containerd defines semantics for STDIO, flags, working directory, file names, etc

- Examples
  - Runtimes (OCI and “v2”)
  - Log forwarding
  - Stream processing/media transformation

“CLI”/executable plugins – Runtimes

runc	**firecracker-containerd**
Default runtime Linux containers	Alternative runtime Firecracker microVMs
Adheres to OCI standard	Adheres to containerd “v2” interface
Specification covers: • command-line arguments/flags • working directory • input files • exit codes	Specification covers: • command-line arguments/flags • working directory • input files • gRPC/ttrpc on a Unix domain socket • exit codes

“CLI”/executable plugins – “v2” runtimes

- **Binary prefixes with `containerd-shim-foo-bar`**

“CLI”/executable plugins – “v2” runtimes

- Binary prefixes with **`containerd-shim`**`-foo-bar`
- **Be located within `PATH`**

“CLI”/executable plugins – “v2” runtimes

- Binary prefixes with **`containerd-shim`**`-foo-bar`
- Be located within **`PATH`**
- **Define program lifecycle through `start` and `delete` arguments**

“CLI”/executable plugins – “v2” runtimes

```
$ containerd-shim-foo-bar start
/path/to/socket.sock

$ containerd-shim-foo-bar delete
```

“CLI”/executable plugins – “v2” runtimes

- Binary prefixes with **`containerd-shim`**`-foo-bar`
- Be located within **`PATH`**
- Define program lifecycle through **`start`** and **`delete`** arguments
- **Implement `TaskService` as a ttrpc service**

“CLI”/executable plugins – “v2” runtimes

```
type TaskService interface {
    State(context.Context, *StateRequest) (*StateResponse, error)
    Create(context.Context, *CreateTaskRequest) (*CreateTaskResponse, error)
    Start(context.Context, *StartRequest) (*StartResponse, error)
    Delete(context.Context, *DeleteRequest) (*DeleteResponse, error)
    Pids(context.Context, *PidsRequest) (*PidsResponse, error)
    Pause(context.Context, *PauseRequest) (*types1.Empty, error)
    Resume(context.Context, *ResumeRequest) (*types1.Empty, error)
    Kill(context.Context, *KillRequest) (*types1.Empty, error)
    Exec(context.Context, *ExecProcessRequest) (*types1.Empty, error)
    Update(context.Context, *UpdateTaskRequest) (*types1.Empty, error)
    Wait(context.Context, *WaitRequest) (*WaitResponse, error)
    …
}
```

“CLI”/executable plugins – “v2” runtimes

- Binary prefixes with **`containerd-shim`**`-foo-bar`
- Be located within **`PATH`**
- Define program lifecycle through **`start`** and **`delete`** arguments
- Implement **`TaskService`** as a ttrpc service
- **Can use containerd’s shim helpers**

“CLI”/executable plugins – “v2” runtimes

```
func main() {
    shim.Run("foo.bar", myShim)
}

func myShim(
    ctx context.Context,
    id string,
    publisher shim.Publisher,
    callback func(),
) (shim.Shim, error){
    // my implementation here!
}
```

“CLI”/executable plugins – “v2” runtimes

- Binary prefixes with **`containerd-shim`**`-foo-bar`
- Be located within **`PATH`**
- Define program lifecycle through **`start`** and **`delete`** arguments
- Implement **`TaskService`** as a ttrpc service
- Can use containerd’s shim helpers
-
```
sudo ctr run \
    --runtime foo.bar \
    docker.io/library/hello-world:latest \
    my-hello-world-container
```

gRPC proxy plugins

- Plugins run as separate processes
- Expose the service API over a Unix domain socket
- containerd acts as a pass-through
- Proxy plugin registered in containerd’s config file
- Snapshot and content services supported as proxy plugins

gRPC proxy plugins - Snapshotters

- Snapshotters provide image- and container-filesystems
- Many implement a form of copy-on-write
- Several built in to containerd
- Out-of-process gRPC proxy plugins enable new development

- Examples
  - Block-device snapshotters: devicemapper and lvm
  - Ongoing discussion about network-based snapshotters

gRPC proxy plugins - Snapshotters

- **Implement `Snapshotter` as a gRPC service**

gRPC proxy plugins - Snapshotters

```go
type Snapshotter interface {
    Stat(context.Context, string) (Info, error)
    Update(context.Context, Info, ...string) (Info, error)
    Usage(context.Context, string) (Usage, error)
    Mounts(context.Context, string) ([]mount.Mount, error)
    Prepare(context.Context, string, string, ...Opt) ([]mount.Mount, error)
    View(context.Context, string, string, ...Opt) ([]mount.Mount, error)
    Commit(context.Context, string, string, ...Opt) error
    Remove(context.Context, string) error
    Walk(context.Context, func(context.Context, Info) error) error
    Close() error
}
```

gRPC proxy plugins - Snapshotters

- Implement `Snapshotter` as a gRPC service
- **Registered in containerd configuration**

gRPC proxy plugins - Snapshotters

```
[proxy_plugins]
  [proxy_plugins.foo-snapshotter]
    type = "snapshot"
    Address = "/var/run/foo-snapshotter.sock"
```

gRPC proxy plugins - Snapshotters

- Implement **`Snapshotter`** as a gRPC service
- Registered in containerd configuration

-
  ```
  sudo ctr run \
      --snapshotter foo-snapshotter \
      docker.io/library/hello-world:latest \
      my-hello-world-container
  ```

Go plugins

- Similar power/flexibility to built-in plugins
- **Can** add at runtime
- Loaded from containerd’s plugins folder (or configured folder)
- Name includes OS, architecture, and OS-specific extension: `myplugin-linux-amd64.so`

- Strongly tied to how containerd was built
  - OS, architecture
  - Version of Go
  - Versions of every common package
- You’re responsible for ensuring compatible build environment

Built-in plugins

- Default plugins are (mostly!) built-in
- In the source tree of containerd
- Can’t add at runtime
- Most powerful/flexible
- Most effort required

- Examples
  - Default snapshotters
  - Default content store
  - Default diff service
  - Default image service
  - Default container service
  - CRI plugin

Built-in plugins – Build your own

- Build in your own plugins
- ...by building your own containerd binary
- You don’t have to fork containerd!
- You solve your own build environment and distribution
- You’re responsible for keeping up to date

Built-in plugins – Build your own

- Write your own `main()` function

Built-in plugins – Build your own

```
func main() {
    app := command.App()
    if err := app.Run(os.Args); err != nil {
        fmt.Fprintf(os.Stderr, "containerd: %s\n", err)
        os.Exit(1)
    }
}
```

Built-in plugins – Build your own

- Write your own **`main()`** function
- **`import` the plugins you want**

Built-in plugins – Build your own

```
import (
    // main function
    "github.com/containerd/containerd/cmd/containerd/command"

    // builtins, see
    // https://github.com/containerd/containerd/blob/master/cmd/containerd/builtins.go
    _ "github.com/containerd/containerd/diff/walking/plugin"
    _ "github.com/containerd/containerd/gc/scheduler"
    _ "github.com/containerd/containerd/runtime/restart/monitor"
    _ "github.com/containerd/containerd/services/containers"
    _ "github.com/containerd/containerd/services/content"
    _ "github.com/containerd/containerd/services/diff"
    _ "github.com/containerd/containerd/services/events"
    _ "github.com/containerd/containerd/services/healthcheck"
    _ "github.com/containerd/containerd/services/images"
    _ "github.com/containerd/containerd/services/introspection"
    _ "github.com/containerd/containerd/services/leases"
    _ "github.com/containerd/containerd/services/namespaces"
```

Built-in plugins – Build your own

```
    _ "github.com/containerd/containerd/services/opt"
    _ "github.com/containerd/containerd/services/snapshots"
    _ "github.com/containerd/containerd/services/tasks"
    _ "github.com/containerd/containerd/services/version"
    // Linux builtins, see
    // https://github.com/containerd/containerd/blob/master/cmd/containerd/builtins_linux.go
    _ "github.com/containerd/containerd/metrics/cgroups"
    _ "github.com/containerd/containerd/runtime/v1/linux"
    _ "github.com/containerd/containerd/runtime/v2"
    _ "github.com/containerd/containerd/runtime/v2/runc/options"

    // snapshotters
    _ "github.com/containerd/containerd/snapshots/devmapper"
    _ "github.com/containerd/containerd/snapshots/overlay"

    // Your plugin!
    _ "github.com/foobar/foobar/foobar-api”
)
```

Built-in plugins – Build your own

- Write your own **`main()`** function
- **`import`** the plugins you want
- **Register your plugin with `init()`**

Built-in plugins – Build your own

```
func init() {
    plugin.Register(&plugin.Registration{
        Type:     plugin.ServicePlugin,
        ID:       "myPlugin.ID",
        Requires: []plugin.Type{
            plugin.MetadataPlugin,
        },

        InitFn: func(ic *plugin.InitContext) (interface{}, error) {
            // Init your plugin here
        },
    })
}
```

Demo!

Demo summary

- Pull image from Amazon ECR with **`amazon-ecr-containerd-resolver`** client library extension
- Custom containerd binary with **`firecracker-control`** built-in plugin
- **`devmapper`** snapshotter (now embedded, former gRPC proxy plugin)
- **`containerd-shim-aws-firecracker`** runtime (executable plugin) to run Firecracker microVMs
- Inside VM, use **`containerd-shim-runc-v1`** (default runtime) for runc

Q&A

Samuel Karp and Maksym Pavlenko

A brief note before we finish —

Session surveys provide valuable information to speakers

Feedback that is very helpful:

- Topics you were excited to learn about
- Suggestions for improving understanding and clarity

Feedback that is extremely unhelpful:

- Comments unrelated to talk content (please refer to the CNCF Code of Conduct)

The “hallway track” is always open!

Feedback and questions welcome

- skarp@amazon.com or @samuelkarp
- makpav@amazon.com or @mak_pav

For support, use the AWS Forums or contact AWS Support

Thank you!

Samuel Karp (@samuelkarp)
Maksym Pavlenko (@mak_pav)