NRI Intro

An Introduction to NRI, Use Cases and Demos

Krisztian Litkey, Feruzjon Muyassarov, Jukka Rissanen

Presentation Overview

- Overview / Recap of NRI
    - What is it ?
    - How does it work ?
    - Scope of NRI, What Can It Do ?
- Usage / Use Cases / Demos
    - Enabling NRI in the runtime
    - Examining NRI events
    - Plugging OCI Hooks into containerd
    - Sometimes you need a hack: annotated device injection
    - Plugging resource management into the runtime
- https://github.com/klihub/nri-intro
    - Demos and a copy of this presentation

What Is NRI, the Node Resource Interface ?

- A common framework for CRI-compatible OCI runtimes
- Provides low-level plumbing for plugging extensions into these runtimes
- Extensions are external to the runtime
- They are compatible across runtimes (‘common’)
- Extensions can hook into lifecycle events of pods and containers
- And make controlled changes to the configuration of containers

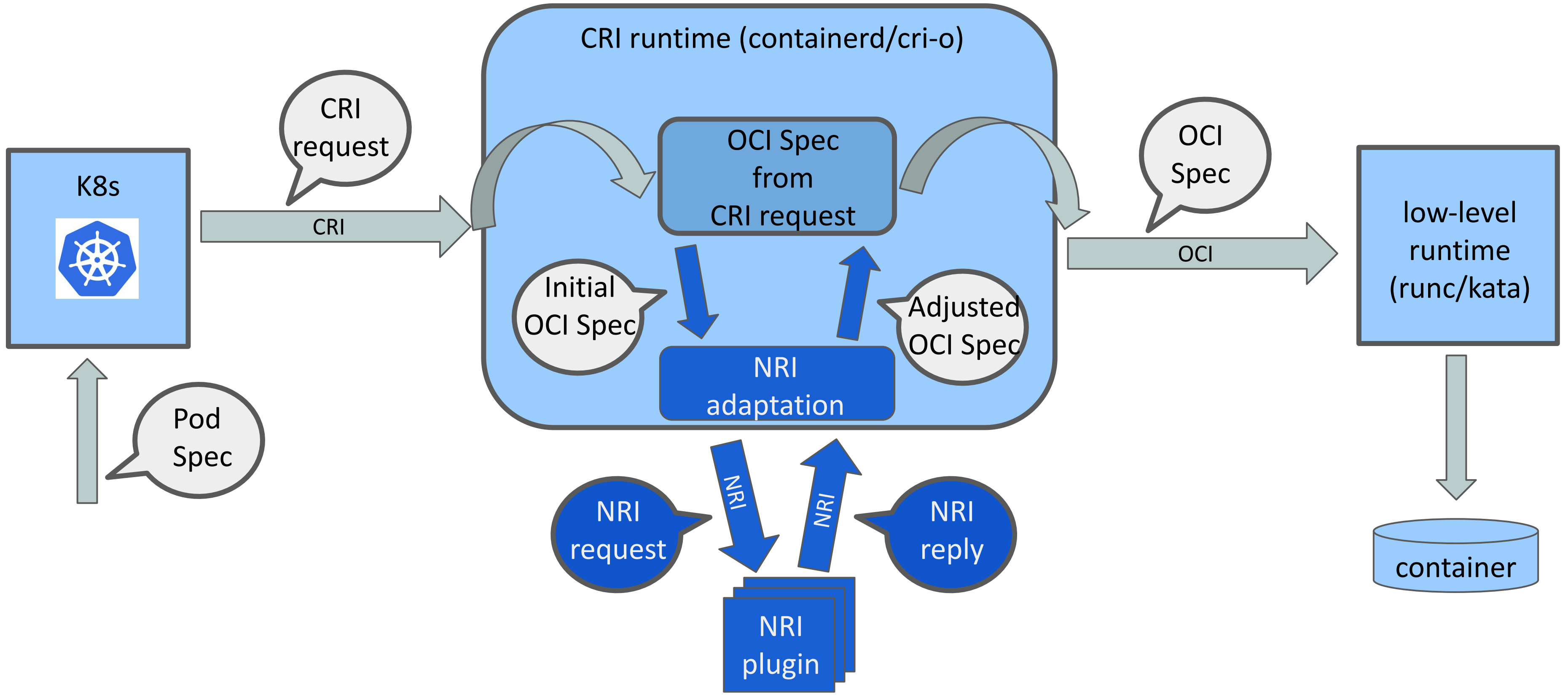
Summary:

- NRI allows one to plug in custom container customization logic to runtimes
- Your cluster + your NRI plugin => your rules

How Does NRI Work ?

- Runtimes rely on low-level runtimes (runc, kata) for manipulating containers
- Containers are practically immutable, apart from compute ‘resources’
- Once created, only ‘resources’ of a container can be updated subsequently
- Runtimes create an ‘OCI Spec’ to describe containers for low-level runtimes
- Basic operation of NRI is simple in principle:
  - NRI hooks itself into the container creation request processing of the runtime
  - After the initial OCI Spec has been created, but before it gets used
  - It passes (a subset of) the OCI Spec to each NRI plugin in turn, and
  - Performs any modifications to the OCI Spec as requested by the plugin
  - The final modified OCI Spec is passed to the low-level runtime to create the container

How Does NRI Work ?

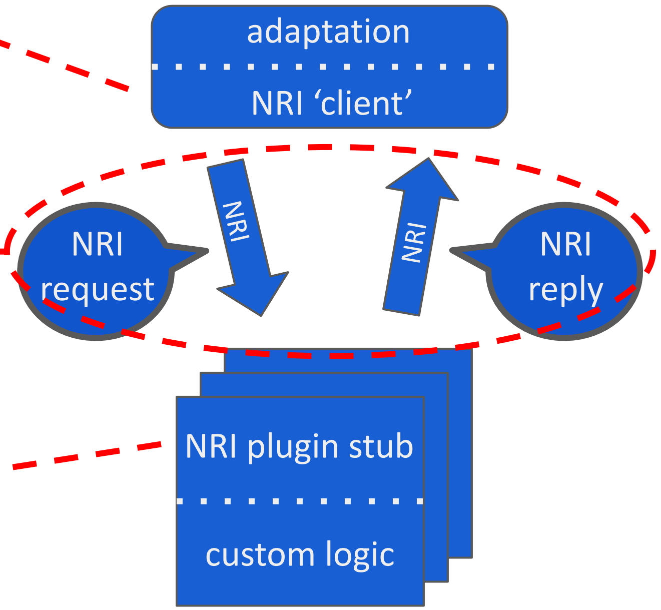


How Does NRI Work?

- NRI is a common ‘framework’
- Multiple pieces collectively achieve NRI’s goal
- Adaptation client:
  - Runtime agnostic library
  - Used by runtimes to integrate to NRI and interact with NRI plugins
- Plugin stub:
  - Plugin library, takes care of the low-level common details of writing a plugin
  - Connection setup, communication, plugin registration, event subscription, etc.
- NRI protocol:
  - Used by the above two to communicate and interact with each other
  - Protobuf-based API definition with ttRPC bindings
  - Defines the set of events, requests and data NRI associates with pods and containers
  - Defines the subset of container/OCI Spec data plugins can modify

How Does NRI Work ?

- runtime adaptation / client
  - runtime agnostic
  - Integrate to NRI / interact with plugins
- NRI protocol
  - protobuf-bases API
  - ttRPC bindings
  - execution model
    - Lifecycle events/requests
  - data model: event/request details
    - pod, container abstraction (~ OCI Spec)
    - NRI container adjustment, updates
- plugin stub
  - low-level details of plugin logic
  - IPC, connection, registration, etc.



Scope of NRI, What Can It Do ?

- Because of ‘resources’, NRI cares about more than just container creation
- Resources include CPU and memory which need to be arbitrated
- Container creation, update: allocation/reallocation of resources
- Container stopping: release of resources
- And… potentially (= usually): reassignment to other containers
- => NRI hooks into all lifecycle events of containers
- Allows most liberal modifications during creation
- Allows altering resources of other existing containers during creation, update and stopping

Scope of NRI, Plugin Inputs

- Pod data
  - ID, name, UID
  - K8s namespace, labels, annotations
  - Linux namespaces, cgroupfs parent
  - Runtime handler name
  - Pod ‘resources’ (see later) and ‘resource’ overhead
- Container data
  - ID, pod ID, name,
  - K8s labels, annotations
  - Args, environment variables, mounts, OCI hooks
  - Linux namespaces, devices
  - ‘Resources’ (see later), cgroupfs path

Scope of NRI, What Can Plugins Modify ?

- Container ‘adjustment’ (creation time modifications):
  - Annotations, environment variables
  - mounts, devices
  - OCI hooks
  - ‘Resources’ (basically parameters controllable via cgroupfs v1/v2)
    - Scheduling parameters
    - CPU/memory pinning
    - Memory, huge page limits
    - LLC cache allocation, block I/O throttling
    - …
- Container ‘updates’ (post-creation modifications):
  - ‘Resources’ (see above)

Usage / Use Cases / Demos

- Enabling NRI in the runtime
- Inspecting NRI events
- Plugging OCI Hooks into containerd
- Sometimes you need a hack: annotated device injection
- Complex Plugin Use Case: pluggable resource management in the runtime

Pluggable Resource Management In the Runtime

- CRI Resource Manager retrofitted to NRI
- All legacy CRI proxy functionality removed
- Two reference policies picked, others removed
  - Balloons: ultra-customizable set of custom, potentially workload-specific pools
  - Topology-aware: near zero-configuration, with automatic resource alignment and isolation
- Deployed as K8s DaemonSet
- Dynamic configuration using ConfigMaps
- Exposes resource availability as NodeResourceTopology CRD’s
- Being OSS-ed to bootstrap a community-maintained common plugins repo

Future Work

- Open sourcing our resource plugins to a community maintained repo
- Getting community feedback, gaining community involvement
- Ironing out the ‘experimentalness’
  - More and better test cases, better test coverage
  - Fixing bugs
  - Aim for protocol completeness and correctness
    - Missing events, missing data
    - Error handling
  - Documentation

Thank You - Q&A

- Any questions ?
- Comments ?