OPENWHISK INVOKER PROPOSAL

CLUSTERED RESOURCES

CLUSTERED (ACTION CONTAINER) RESOURCES

- Resources meaning: action containers
  - Invoker should launch action containers using some cluster manager (e.g. k8s or mesos)
  - This cluster may change the size of action container pool at any time
- We are NOT including:
  - Invoker instances
  - Controller instances
  - Other OW dependency services (kafka, etc)

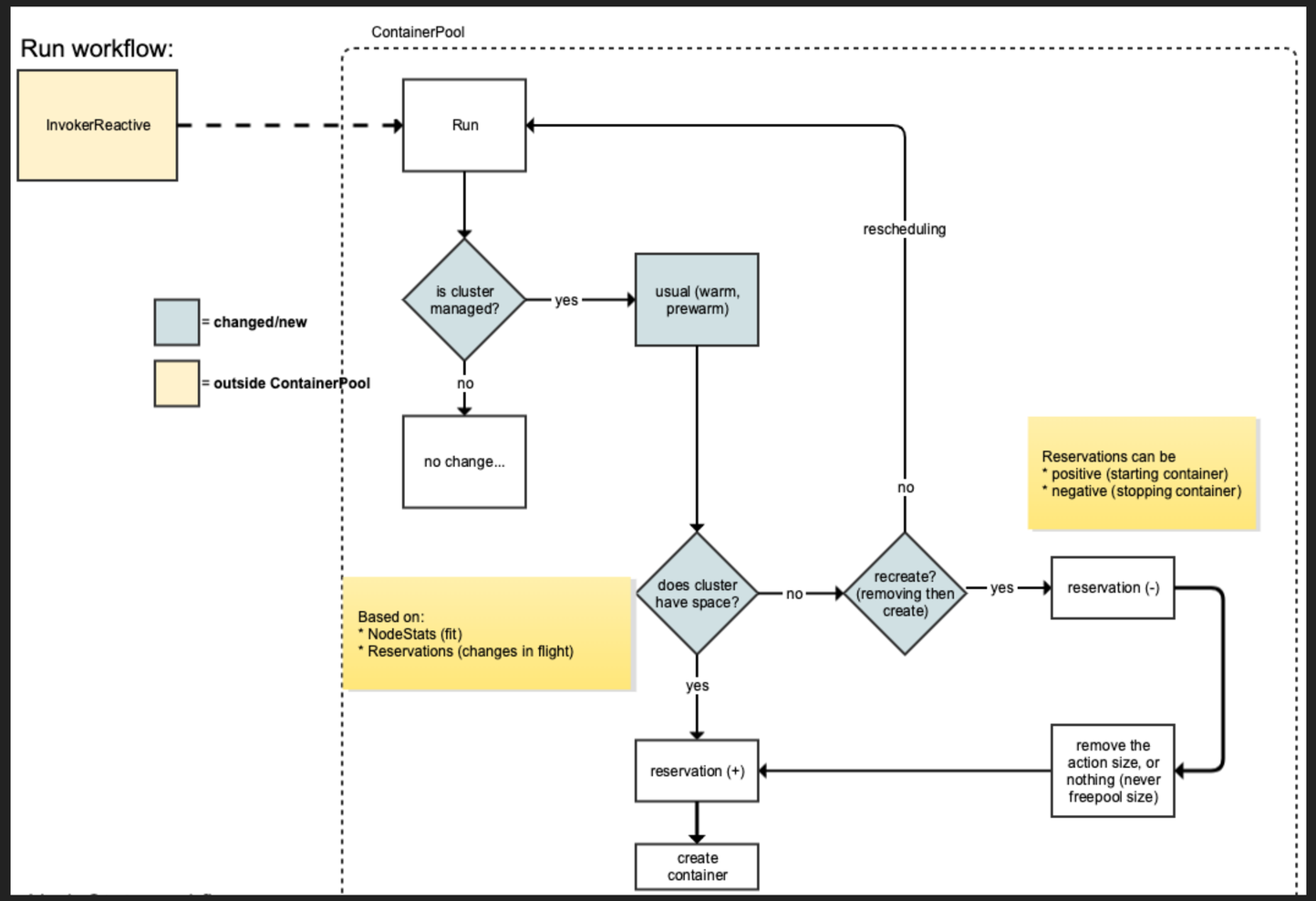
TODAY

- Controller: ShardingContainerPoolBalancer
  - Requires even(ish) distribution across controllers
  - Performs resource scheduling (same as ContainerPool?)
- Risks:
  - Uneven distribution: when round robin doesn’t
  - Resource scheduling is (un)affected by dynamic cluster size changes.
    - Invoker “user-memory” config doesn’t account for changing cluster size.
    - “But can we make user-memory dynamic?” Maybe, but this means dynamic shard sizes and relies on the even distributions still.
- It would be better for controller to “not care” about invoker config state, other than “should likely be able to launch a container”, which can be inferred from Healthy vs Unresponsive.

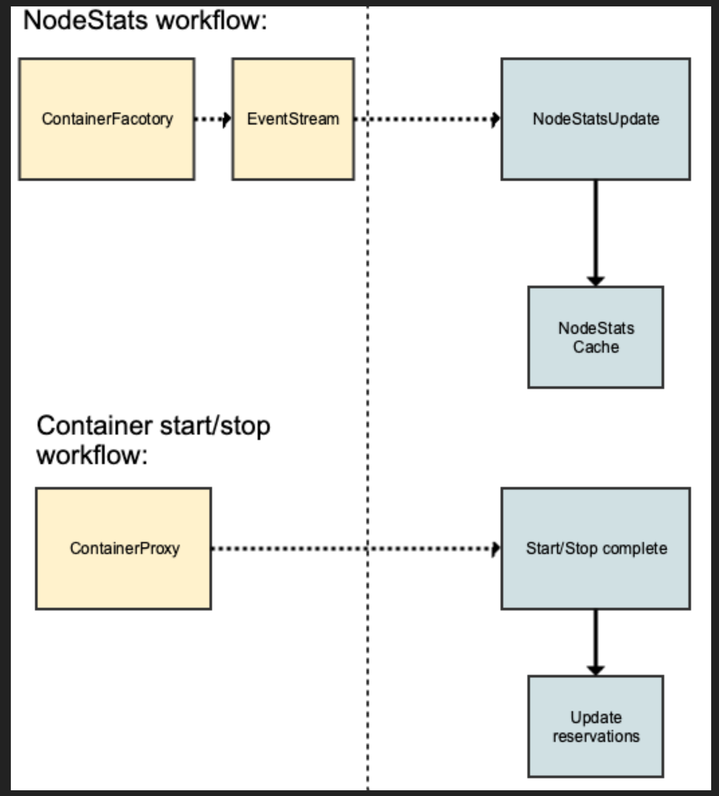
PROPOSAL SUMMARY

- Invoker
  - Never over provision - favor stopping idle container and waiting for resources, vs “waiting forever”.
  - Collect NodeStats from event stream to determine action container “launchability” (estimated).
  - Track Reservations: container starts/stops, not yet reflected in NodeStats (but does affect launchability).
  - Prewarms: start only when resources are available (TODO: periodically check)
- Controller
  - No changes, but rely on whisk.container-pool.user-memory=∞ (or some large number)

CONTAINERPOOL RUN WORKFLOW



CONTAINERPOOL OTHER WORKFLOWS



IMPL DETAILS

- NodeStats
  - Is this a mesos thing? No. (but there is a resemblance to “mesos offers”)
  - Actions need contiguous resources (e.g. 100MB on single node is more usable than 1MB on 100 nodes), so aggregate cluster metrics cannot be used.
  - ContainerFactory impls can use any means to collect data, and emit to ContainerPool via NodeStatsUpdate events.
    - k8s can use `kubectl describe node` or metrics api, or ???
    - mesos can use offer data, or metrics api, or ???
  - ContainerPool will cache NodeStats for reference (do not query system NodeStats per activation!)

IMPL DETAILS

- Reservations
  - Is this a mesos thing? No
  - For the times between ContainerPool resource allocation, and “actual” resource allocation.
  - Allows for delays in updates to NodeStats.
  - Prevents (in some cases) over-provisioning during burst load.

RISKS

- Invoker
  - Unresponsive: Controller only stops using “home invoker” when it becomes Unresponsive
    - In normal usage, there will be a delay between when invoker reaches inability to launch containers, and controller detects Unresponsive status
    - TODO: improve Controller tracking data to be based on “activation weight” that might push invoker to become Unresponsive
  - Overprovisioning: ContainerPool must avoid over provisioning, where:
    - Cluster cannot provision containers
    - ContainerPool did not anticipate ability to create a container accurately
    - Otherwise, may result in “waiting forever” to create a container
    - TODO: add some buffer amount (in addition to NodeStats+Reservation values) to calculation of “launchability”.
- Controller
  - Sharding between invokers not being used
  - TODO: provide an alternate sharding resource, like “activation weight”; container memory size is no longer a concern of invoker, but the memory required to init containers, and process activations’ requests+responses is.
  - Side note: currently container init uses substantial memory compared to size of action code, so it would be good to optimize this soon, either by loading code direct in container, or some other /init protocol change.