[shared externally]

- All state are defined by https://sourcegraph.com/github.com/ray-project/ray@06ef4ab94e09738c4afe4d4fdc75a9cb41464b4c/-/blob/python/ray/workflow/workflow_state.py?L49:7
- Submitting a workflow
    - Submit the DAG to WMA(WorkflowManagermentActor found in workflow_access)
    - WMA -> write it into DB
    - WMA -> create a state and run https://sourcegraph.com/github.com/ray-project/ray@HEAD/-/blob/python/ray/workflow/workflow_executor.py
    -
- Dashboard (https://github.com/ray-project/ray/tree/master/dashboard)
    - Running in the head node
    - No driver here!
- Dashboard Agent (https://github.com/ray-project/ray/tree/master/dashboard)
    - Running in the work node (head node).
    - Each raylet will have one
    - Ray.init can run here
    - Agent can have a driver
- For dashboard and the dashboard agent, you can check this:
    - https://github.com/ray-project/ray/tree/master/dashboard/modules/log
        - Dashboard Agent
            - https://github.com/ray-project/ray/blob/master/dashboard/modules/log/log_agent.py
            - Log agent doing some work for logs
        - Dashboard
            - https://github.com/ray-project/ray/blob/master/dashboard/modules/log/log_head.py
            - Log head pulling the request from log agent
    - Serve Agent
        - https://github.com/ray-project/ray/blob/master/dashboard/modules/serve/serve_agent.py#L168
        - Here we access serve controller actor handle explicitly
        - You probably want to do the similar things for workflow actor manager
        - Call this to get actor handle:
            - https://sourcegraph.com/github.com/ray-project/ray@06ef4ab94e09738c4afe4d4fdc75a9cb41464b4c/-/blob/python/ray/workflow/workflow_access.py?L371-374
- Dashboard —> some request → Dashboard Agent (ray.init) – send request to WorkflowManagementActor → read the data from the memory for running workflow (read from storage for not running one [second step])
- Dashboard Agent
    - Define a WorkflowObservability module

        - func(workflow_id) -> (dict(task_id, metadata))
    - Inside the module, you'll have a workflow management actor handler.
    - You need to define the methods in WMA to retrieve the state of a workflow job
    - Then you'll send the useful state back to the dashboard

MVPs

- P0
    - Visualize the running workflow DAG
    - Check the progress of the running workflow
    - Some metadata of the workflow tasks.
    - Dynamic updates?
        - In dashboard agent, you can send a remote call to WMA (request new data)
            - Return if the state of some workflow changed
- P1
    - Check the status of the non-running DAG
        - Failed? Get the exception or error message?
    - Check the metadata of the workflow tasks
        - How long does it run
        - The size of the output
- P2
    - Workflow management
        - Kill a workflow
        - Restart/resume