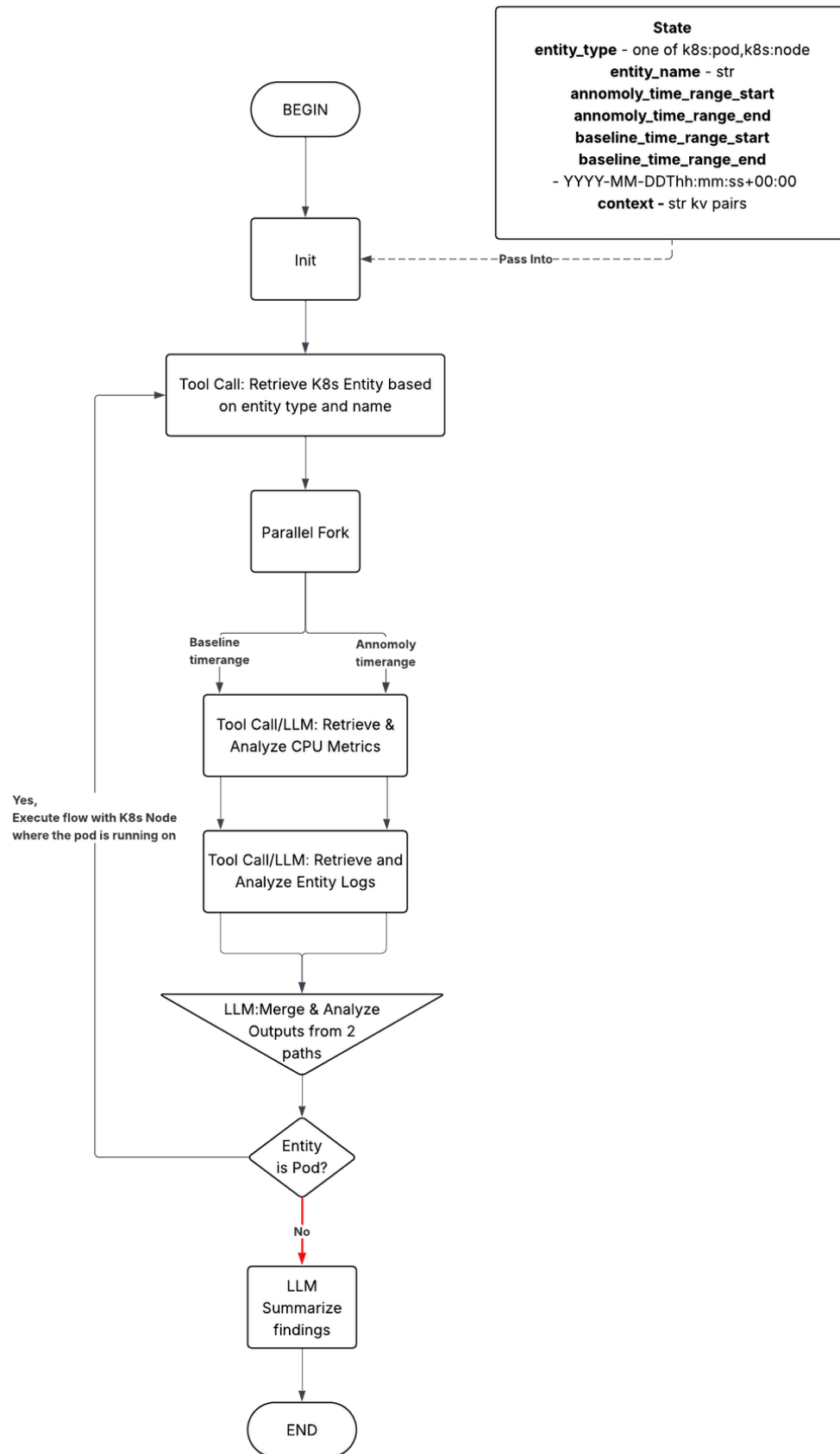


# Demo K8s Workflow for Agentic Framework Evaluation

## Summary

This sample workflow aims to distill our existing k8s ADT flow so that it can be used for the framework evaluation. This workflow starts with a known entity like a Kubernetes pod and traverse to the residing node for finding the root cause.

## Workflow Diagram



Some of the capabilities to test against:

- Parallel Chain
- Loop
- Condition/Decision
- Tool Call
- Summarization

#### Mock Tool Calls

These calls simulate a pod ( `frontend-6d8f4f79f7-kxzpl` ) that's crash loop back-offing due to CPU pressure from the pod and the node ( `node-1` ).

#### Time ranges

```
1 normal_range = (  
2     datetime.fromisoformat("2024-06-26T09:00:00+00:00"),  
3     datetime.fromisoformat("2024-06-26T10:00:00+00:00"),  
4 )  
5 high_range = (  
6     datetime.fromisoformat("2024-06-26T10:00:00+00:00"),  
7     datetime.fromisoformat("2024-06-26T10:30:00+00:00"),  
8 )
```

```
1 import random  
2 from datetime import datetime, timedelta  
3 from typing import Any  
4  
5  
6 def get_entities(entity_type: str, entity_name: str) -> dict[str, Any]:  
7     # Mock pod data  
8     mock_pod = {  
9         "frontend-6d8f4f79f7-kxzpl": {  
10             "node_name": "node-1",  
11             "pod_ip": "10.1.2.34",  
12             "host_ip": "192.168.1.10",  
13             "start_time": "2024-06-26T09:58:12Z",  
14             "labels": {"app": "frontend", "env": "prod", "tier": "web"},  
15             "annotations": {"prometheus.io/scrape": "true"},  
16         }  
17     }  
18  
19     # Mock node data  
20     mock_node = {  
21         "node-1": {  
22             "internal_ip": "192.168.1.10",  
23             "capacity": {"cpu": "4", "memory": "16Gi"},  
24             "labels": {"topology.kubernetes.io/zone": "us-west1-a",  
25 "kubernetes.io/role": "worker"},  
26             "taints": [],  
27             "conditions": [{"type": "Ready", "status": "True",  
28 "last_heartbeat_time": "2024-06-26T10:00:01Z"}],  
29         }  
30     }  
31  
32     # Dispatcher logic
```

```

31     if entity_type == "k8s:pod":
32         if entity_name in mock_pod:
33             return {"entity_type": "pod", "name": entity_name, "metadata":
mock_pod[entity_name]}
34         raise ValueError(f"Pod '{entity_name}' not found.")
35
36     if entity_type == "k8s:node":
37         if entity_name in mock_node:
38             return {"entity_type": "node", "name": entity_name, "metadata":
mock_node[entity_name]}
39         raise ValueError(f"Node '{entity_name}' not found.")
40
41     raise ValueError(f"Unsupported entity type: {entity_type}. Supported types are
'k8s:pod' and 'k8s:node'.")
42
43
44 def get_cpu_utilization(entity_type: str, entity_name: str, start: str, end: str) ->
list[dict[str, Any]]:
45     known_pods = ["frontend-6d8f4f79f7-kxzpl"]
46     known_nodes = ["node-1"]
47
48     # Parse start and end times
49     try:
50         start_dt = datetime.fromisoformat(start)
51         end_dt = datetime.fromisoformat(end)
52     except Exception as err:
53         raise ValueError(f"Invalid time format for start '{start}' or end '{end}'.
Expected ISO format.") from err
54
55     # Validate entity
56     if entity_type == "k8s:pod" and entity_name not in known_pods:
57         raise ValueError(f"Unknown pod: {entity_name}")
58     if entity_type == "k8s:node" and entity_name not in known_nodes:
59         raise ValueError(f"Unknown node: {entity_name}")
60     if entity_type not in ["k8s:pod", "k8s:node"]:
61         raise ValueError(f"Unsupported entity_type '{entity_type}'")
62
63     # Recognized mock time ranges
64     normal_range = (
65         datetime.fromisoformat("2024-06-26T09:00:00+00:00"),
66         datetime.fromisoformat("2024-06-26T10:00:00+00:00"),
67     )
68     high_range = (
69         datetime.fromisoformat("2024-06-26T10:00:00+00:00"),
70         datetime.fromisoformat("2024-06-26T10:30:00+00:00"),
71     )
72
73     def generate_series(start: datetime, end: datetime, usage_range: tuple) ->
list[dict[str, Any]]:
74         points = []
75         current = start
76         while current <= end:
77             usage = round(random.uniform(*usage_range), 2)
78             points.append({"timestamp": current.isoformat(), "cpu_percent": usage})
79             current += timedelta(minutes=5)
80         return points
81
82     if start_dt == normal_range[0] and end_dt == normal_range[1]:

```

```

83     usage_range = (10.0, 30.0)
84     elif start_dt == high_range[0] and end_dt == high_range[1]:
85         usage_range = (80.0, 95.0)
86     else:
87         raise ValueError(f"No mock data available for time range '{start} to
{end}'")
88
89     return generate_series(start_dt, end_dt, usage_range)
90
91
92 def get_logs(entity_type: str, entity_name: str, start: str, end: str) ->
list[dict[str, Any]]:
93     known_pods = {
94         "frontend-6d8f4f79f7-kxzpl": [
95             # Baseline timerange: 2024-06-26T09:00:00+00:00 to 2024-06-
26T10:00:00+00:00
96             {
97                 "type": "Info",
98                 "reason": "Created",
99                 "message": "Created container frontend",
100                "timestamp": "2024-06-26T09:02:00+00:00",
101            },
102            {
103                "type": "Info",
104                "reason": "Pulled",
105                "message": "Successfully pulled image 'frontend:v1.2.3'",
106                "timestamp": "2024-06-26T09:03:00+00:00",
107            },
108            {
109                "type": "Info",
110                "reason": "Started",
111                "message": "Started container frontend",
112                "timestamp": "2024-06-26T09:05:00+00:00",
113            },
114            {
115                "type": "Info",
116                "reason": "Started",
117                "message": "Started container frontend",
118                "timestamp": "2024-06-26T09:55:00+00:00",
119            },
120            # High timerange: 2024-06-26T10:00:00+00:00 to 2024-06-26T10:30:00+00:00
121            {
122                "type": "Warning",
123                "reason": "Unhealthy",
124                "message": "Liveness probe failed: HTTP probe failed with status
code 500",
125                "timestamp": "2024-06-26T10:19:30+00:00",
126            },
127            {
128                "type": "Warning",
129                "reason": "BackOff",
130                "message": "Back-off restarting failed container",
131                "timestamp": "2024-06-26T10:20:00+00:00",
132            },
133        ]
134     }
135
136     known_nodes = {

```

```

137     "node-1": [
138         # Baseline timerange: 2024-06-26T09:00:00+00:00 to 2024-06-
26T10:00:00+00:00
139         {
140             "type": "Normal",
141             "reason": "KubeletReady",
142             "message": "kubelet is posting ready status",
143             "timestamp": "2024-06-26T09:10:00+00:00",
144         },
145         {
146             "type": "Normal",
147             "reason": "NodeHasSufficientMemory",
148             "message": "Node has sufficient memory available",
149             "timestamp": "2024-06-26T09:20:00+00:00",
150         },
151         {
152             "type": "Normal",
153             "reason": "NodeHasNoDiskPressure",
154             "message": "Node has no disk pressure",
155             "timestamp": "2024-06-26T09:30:00+00:00",
156         },
157         # High timerange: 2024-06-26T10:00:00+00:00 to 2024-06-26T10:30:00+00:00
158         {
159             "type": "Normal",
160             "reason": "KubeletReady",
161             "message": "kubelet is posting ready status",
162             "timestamp": "2024-06-26T10:10:00+00:00",
163         },
164         {
165             "type": "Warning",
166             "reason": "CPUPressure",
167             "message": "Node is under CPU pressure",
168             "timestamp": "2024-06-26T10:15:00+00:00",
169         },
170     ]
171 }
172
173 try:
174     start_dt = datetime.fromisoformat(start)
175     end_dt = datetime.fromisoformat(end)
176 except Exception as err:
177     raise ValueError(f"Invalid time format for start '{start}' or end '{end}'.
Expected ISO format.") from err
178
179 def filter_events(events, start_dt, end_dt):
180     filtered = []
181     for event in events:
182         event_ts = datetime.fromisoformat(event["timestamp"])
183         if start_dt <= event_ts <= end_dt:
184             filtered.append(event)
185     return filtered
186
187 if entity_type == "k8s:pod":
188     if entity_name not in known_pods:
189         raise ValueError(f"Unknown pod: {entity_name}")
190     events = filter_events(known_pods[entity_name], start_dt, end_dt)
191 elif entity_type == "k8s:node":
192     if entity_name not in known_nodes:

```

```
193         raise ValueError(f"Unknown node: {entity_name}")
194         events = filter_events(known_nodes[entity_name], start_dt, end_dt)
195     else:
196         raise ValueError(f"Unsupported entity_type '{entity_type}'")
197
198     return events
199
200
201 if __name__ == "__main__":
202     # Example usage
203     print(get_entities("k8s:pod", "frontend-6d8f4f79f7-kxzpl"))
204     print(get_entities("k8s:node", "node-1"))
205     print(
206         get_cpu_utilization(
207             "k8s:pod", "frontend-6d8f4f79f7-kxzpl", "2024-06-26T09:00:00+00:00",
208             "2024-06-26T10:00:00+00:00"
209         )
210     )
211     print(
212         get_cpu_utilization(
213             "k8s:pod", "frontend-6d8f4f79f7-kxzpl", "2024-06-26T10:00:00+00:00",
214             "2024-06-26T10:30:00+00:00"
215         )
216     )
217     print(get_cpu_utilization("k8s:node", "node-1", "2024-06-26T09:00:00+00:00",
218                               "2024-06-26T10:00:00+00:00"))
219     print(get_cpu_utilization("k8s:node", "node-1", "2024-06-26T10:00:00+00:00",
220                               "2024-06-26T10:30:00+00:00"))
221     print(get_logs("k8s:pod", "frontend-6d8f4f79f7-kxzpl", "2024-06-26T09:00:00+00:00", "2024-06-26T10:00:00+00:00"))
222     print(get_logs("k8s:pod", "frontend-6d8f4f79f7-kxzpl", "2024-06-26T10:00:00+00:00", "2024-06-26T10:30:00+00:00"))
223     print(get_logs("k8s:node", "node-1", "2024-06-26T09:00:00+00:00", "2024-06-26T10:00:00+00:00"))
224     print(get_logs("k8s:node", "node-1", "2024-06-26T10:00:00+00:00", "2024-06-26T10:30:00+00:00"))
```