Highlighting cAdvisor: Container Performance Monitoring

cAdvisor (Container Advisor) is a running daemon that collects, aggregates, processes, and exports information about running containers. It provides an essential layer of insight into the resource utilization and performance of your Dockerized services. When integrated with Grafana Alloy and Grafana, cAdvisor empowers you with real-time operational visibility into your local data platform.
This guide will demonstrate basic and advanced use cases of cAdvisor, leveraging your **Advanced Track** local environment setup.
Reference: This guide builds upon the concepts and setup described in **Section 4.2. Core Technology Deep Dive** of the **Core Handbook** and the **Progressive Path Setup Guide Deep-Dive Addendum**, specifically emphasizing cAdvisor's role in the **Observability** section.

Basic Use Case: Monitoring Container CPU and Memory Usage

Objective: To demonstrate how cAdvisor collects fundamental resource metrics (CPU, memory) for all running containers and how these metrics are visualized in Grafana.
Role in Platform: Provide foundational visibility into individual container health and resource consumption, allowing for basic performance monitoring and identification of resource-hungry services.
Setup/Configuration (Local Environment - Advanced Track):

1. **Ensure all Advanced Track services are running:** docker compose up --build -d from your project root. This includes cAdvisor, grafana-alloy, and grafana.
2. **Verify cAdvisor is running:** Check Docker logs for the cAdvisor container to ensure it's healthy (docker compose logs cadvisor). It should be collecting metrics.
3. **Verify Grafana is accessible:** Go to http://localhost:3000 (initially anonymous or configure admin user).
4. **Confirm Grafana Alloy is configured to scrape cAdvisor:** Review your observability/alloy-config.river to ensure it includes a Prometheus scrape_config for cAdvisor (usually on port 8080 or as configured in docker-compose.yml).
   Example docker-compose.yml snippet for cAdvisor and Grafana Alloy integration:
   ```
   # ... other services
   grafana-alloy:
     # ...
     depends_on:
       # ...
       cAdvisor:
   ```

```
      condition: service_healthy
  cAdvisor:
   image: gcr.io/cadvisor/cadvisor:v0.47.0 # Or latest stable
   volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
    - /dev/disk/:/dev/disk:ro
   privileged: true # Required for cAdvisor to access host information
   ports:
    - "8080:8080" # cAdvisor UI and metrics endpoint
```

Example observability/alloy-config.river snippet (conceptual):

```
# ... other components
prometheus.scrape "cadvisor" {
  targets    = [{"__address__" = "cadvisor:8080"}] # 'cadvisor' is the service name in
docker-compose
  forward_to = [prometheus.remote_write.default.receiver]
}
# ...
```

Steps to Exercise:

1. **Generate some activity:**
   - Start python3 simulate_data.py to generate traffic to FastAPI, which will then generate activity for Kafka and Spark. This will ensure your containers are actively working and consuming resources.
2. **Access Grafana Dashboards:**
   - Go to http://localhost:3000.
   - Navigate to a pre-provisioned dashboard designed for container monitoring (e.g., "Docker Container Overview" or "Host & Container Metrics"). If you don't have a specific one, you can import a community dashboard (e.g., "cAdvisor / Prometheus Host and Container" dashboard ID 14210) or create a new panel.
   - Look for panels showing "CPU Usage by Container" and "Memory Usage by Container."
3. **Observe Metrics:**
   - You should see graphs displaying the CPU and memory consumption for individual Docker containers (e.g., fastapi_ingestor, kafka, spark, airflow-webserver, etc.).
   - The graphs will fluctuate based on the workload generated by simulate_data.py. For example, the fastapi_ingestor will show CPU usage when receiving requests, and spark will show higher CPU/memory during data processing.

Verification:

- **Grafana:** Clear and continuously updating graphs for CPU and memory usage per container are visible, reflecting the activity of your data platform services.
- **cAdvisor UI (Optional):** You can also directly access the cAdvisor UI at http://localhost:8080 (though grafana-alloy primarily scrapes its /metrics endpoint). Here, you can see detailed resource usage per container.

Advanced Use Case 1: Identifying Resource Bottlenecks and "Noisy Neighbors"

Objective: To demonstrate how cAdvisor metrics in Grafana can help identify which containers are consuming the most resources, potentially bottlenecking the host machine or impacting other services ("noisy neighbors").
Role in Platform: Facilitate resource optimization, capacity planning, and troubleshooting performance degradation due to resource contention.
Setup/Configuration:

1. **Ensure Basic Use Case setup is complete.**
2. **Increase Workload:** Temporarily increase the workload significantly to push resource limits.
    - Modify simulate_data.py to have a very low DELAY_SECONDS (e.g., 0.001 or remove it) to generate maximum traffic.
    - Run multiple instances of simulate_data.py concurrently.
    - Alternatively, use Locust (locust -f locust_fastapi_ingestor.py) with a high number of users (e.g., 200) and a high spawn rate.

Steps to Exercise:

1. **Apply Heavy Load:** Start simulate_data.py with very low delay, or use Locust as described.
2. **Monitor Grafana:** Go to your container monitoring dashboard in Grafana.
    - Focus on **CPU and Memory utilization graphs that show metrics broken down by container name.**
    - Observe the **host machine's overall CPU/Memory usage** (often available on the same dashboard from node_exporter, which grafana-alloy might also collect).
3. **Identify Bottlenecks:**
    - Look for containers whose CPU usage consistently hits high percentages (e.g., 80-100%) or whose memory usage steadily climbs towards its allocated limit.
    - If the overall host CPU/memory is saturated, identify which individual containers are contributing most to that saturation. For example, spark containers typically consume high CPU/memory during processing, but if fastapi_ingestor is unexpectedly high under normal load, it might indicate an issue.
4. **Simulate Resource Constraint:**
    - Edit your docker-compose.yml to intentionally limit resources for a container (e.g., fastapi_ingestor).
      # In docker-compose.yml
      fastapi_ingestor:

```
    # ... existing config
    deploy: # Use deploy for resource limits in Compose V3
      resources:
        limits:
          cpus: '0.5' # Limit to 0.5 CPU core
          memory: 128M # Limit memory to 128MB
```

- Run docker compose up -d --build to apply changes.
- Resume simulate_data.py (or Locust) and observe the FastAPI container's performance and cAdvisor metrics in Grafana.

Verification:

- **Grafana:** Under heavy load, you will clearly see which containers are consuming the most CPU and memory. When you apply resource limits, you'll observe the container's CPU usage capping at the defined limit, and if the workload exceeds that, its latency will increase, and errors might occur, demonstrating how limits are enforced and detectable via cAdvisor.
- **Performance Impact:** The service with limited resources (e.g., FastAPI with cpus: '0.5') will show degraded performance (higher latency, lower RPS) even if the host has available capacity, highlighting the impact of container resource configuration.

Advanced Use Case 2: Custom Metric Collection (Conceptual via Sidecar)

Objective: While cAdvisor primarily collects system-level container metrics, it's often used in conjunction with Prometheus/Grafana Alloy for a holistic view. This use case demonstrates how you would conceptually expose application-specific metrics from your FastAPI application so they can be scraped by Grafana Alloy, complementing cAdvisor's data.
Role in Platform: Extend observability beyond infrastructure to application-level insights (e.g., API request counts, processing times, business metrics), critical for defining SLIs/SLOs.
Setup/Configuration:

1. **Modify FastAPI to expose Prometheus metrics:**
   - In your fastapi_app/app/main.py, install prometheus_fastapi_instrumentator.
   - Instrument your FastAPI app to expose metrics on a dedicated endpoint (e.g., /metrics).

   Example fastapi_app/app/main.py snippet:
```
# fastapi_app/app/main.py (conceptual additions)
from fastapi import FastAPI
from prometheus_fastapi_instrumentator import Instrumentator
# ... other imports and FastAPI app initialization ...

app = FastAPI(title="Financial/Insurance Data Ingestor API")

# Instrument FastAPI for Prometheus metrics
Instrumentator().instrument(app).expose(app) # Exposes metrics on /metrics endpoint
```

```
@app.get("/health", tags=["Monitoring"])
async def health_check():
    return {"status": "healthy", "message": "Welcome to Financial/Insurance Data Ingestor API!"}

# ... your existing FastAPI ingestion endpoints ...
```

2. **Update docker-compose.yml:** Ensure the FastAPI service's Prometheus endpoint is accessible to Grafana Alloy.
   Example docker-compose.yml snippet:
```
# ...
fastapi_ingestor:
  build: ./fastapi_app
  # ... other config
  ports:
    - "8000:8000" # Your main API port
    # No need to expose metrics port if Grafana Alloy is in the same docker network
```

3. **Update observability/alloy-config.river:** Add a scrape config for FastAPI's metrics endpoint.
   Example observability/alloy-config.river snippet (conceptual):
```
# ...
prometheus.scrape "fastapi_ingestor" {
  targets    = [{"__address__" = "fastapi_ingestor:8000"}] # 'fastapi_ingestor' is the service name
  metrics_path = "/metrics" # The path where FastAPI exposes its metrics
  forward_to = [prometheus.remote_write.default.receiver]
}
# ...
```

Steps to Exercise:

1. **Rebuild and restart affected services:** docker compose up --build -d fastapi_ingestor grafana-alloy (or all services).
2. **Generate traffic:** Run python3 simulate_data.py to create API activity.
3. **Access Grafana:** Go to http://localhost:3000.
4. **Create Custom Dashboard Panel:**
   - Create a new dashboard or add a panel to an existing one.
   - Configure the panel to use your Prometheus data source (which Grafana Alloy sends metrics to).
   - Write a PromQL query for FastAPI metrics (e.g., http_requests_total{job="fastapi_ingestor"}, http_request_duration_seconds_bucket{job="fastapi_ingestor"}).
5. **Observe Custom Metrics:**

    - You should see graphs showing API request counts, latency histograms, etc., derived directly from your FastAPI application.

Verification:

- **Grafana:** Your custom panels correctly display metrics from the FastAPI application, demonstrating that Grafana Alloy can scrape application-specific Prometheus endpoints, thus enriching your observability data beyond what cAdvisor provides.
- **API /metrics endpoint (Optional):** You can directly access http://localhost:8000/metrics to see the raw Prometheus metrics exposed by FastAPI.

Advanced Use Case 3: Monitoring Cluster-wide Resource Utilization & Alerting

Objective: To use cAdvisor and Grafana to get an aggregated view of resource utilization across the entire Docker Compose environment (simulating a cluster) and set up basic alerts for high resource consumption.

Role in Platform: Provide a holistic view of the local environment's health, enable proactive alerting on potential resource exhaustion before it impacts critical data pipelines.

Setup/Configuration:

1. **Ensure all Advanced Track services are running and generating data.**
2. **Confirm grafana-alloy is collecting from all cAdvisor targets** (as in Basic Use Case).
3. **Configure Grafana Alerting:**
    - In Grafana, go to "Alerting" -> "Alert Rules".
    - Create a new alert rule.

Example Alert Rule (conceptual):# Alert for high overall CPU usage on the Docker host (simulated cluster)
Rule type: Grafana managed alert
Data source: Prometheus (Grafana Alloy target)
Query (PromQL): sum(rate(container_cpu_usage_seconds_total{container!=""}[1m])) by (instance) / sum(machine_cpu_cores) * 100
  # This query calculates the total CPU usage across all containers on a given instance (host)
  # and divides by total CPU cores to get percentage.
  # Adjust `container!=""` to filter containers if needed.
Condition: WHEN last() OF A IS ABOVE 80
For: 5m # Wait for 5 minutes of sustained high usage
Labels:
  severity: warning
  component: infrastructure
  service: docker_host
Annotations:
  summary: "High CPU usage on Docker host"
  description: "The Docker host (simulating your data platform cluster) CPU utilization has been above 80% for 5 minutes. This may impact overall performance and lead to degraded service."
  remediation: "1. Identify top CPU-consuming containers via Grafana 'Docker Container Overview' dashboard. 2. Consider optimizing Spark jobs, reducing ingestion rate, or allocating

more CPU to your Docker Desktop/VM."
  dashboard_link: "http://localhost:3000/d/<your_container_dashboard_uid>"

Steps to Exercise:

1. **Generate Sustained High Load:**
    - Use Locust with a very high number of users (e.g., 500) and a high spawn rate, running for a sustained period (e.g., 10-15 minutes). This will push your local machine's resources to their limits.
    - Alternatively, run multiple simulate_data.py instances with DELAY_SECONDS=0.
2. **Monitor Grafana:**
    - Observe the "Host & Container Metrics" dashboard, looking at overall CPU and memory usage of the docker_host (or whichever instance name cAdvisor reports).
3. **Trigger Alert:** Allow the load to run long enough for the CPU (or memory) to consistently stay above the 80% threshold for 5 minutes.
4. **Observe Alert:**
    - In Grafana, navigate to "Alerting" -> "Alert Rules".
    - The alert rule you created should transition to "Firing" status.
    - You might see a notification within Grafana or via configured notification channels (if set up, e.g., email, Slack, which is typically outside Docker Compose scope for local setup).

Verification:

- **Grafana:** The alert rule changes status to "Firing", and its history shows the alert state changes. The dashboard metrics clearly show the sustained high resource utilization that triggered the alert. This demonstrates how cAdvisor metrics, combined with Grafana's alerting capabilities, enable proactive monitoring and notification for potential system health issues at a cluster-wide level.

This concludes the guide for cAdvisor.