Adapting Kubernetes to IIoT and Industry 4.0 protocols - An initial performance analysis

Ahmad Alamouss and Holger Eichelberger

{alamoush,eichelberger}@sse.uni-hildesheim.de

Software Systems Engineering
University of Hildesheim

www.sse.uni-hildesheim.de
Content

1. Motivation ......................................................... 3
2. Contribution ...................................................... 4
3. Approach .......................................................... 5
4. Experiment and Results ......................................... 8
5. Conclusion ........................................................ 13
6. Future Work ....................................................... 14
Motivation

• Virtualization and containerization in the field of IIoT/Industry 4.0.

• Interoperability in IIoT/Industry 4.0.

• A lack of interoperability exists and significantly increases complexity and costs for IIoT/Industry 4.0 deployment/integration.

• To overcome the lack of interoperability, ongoing efforts aim at standardizing IIoT/Industry 4.0.

What-if question

➢ IIoT/Industry 4.0 communication protocols standardization.
➢ Container orchestration tool adaptation and performance impact.
Contribution

• We present an approach to replace the communication protocol of Kubernetes without modifying the code base.

• Communication protocols and information model
  ▪ MQTT: Message Queuing Telemetry Transport
  ▪ AMQP: Advanced Message Queuing Protocol
  ▪ AAS: Asset Administration Shell as an Industry 4.0 information model
Approach

• Kubernetes

- api: API server
- sched: Scheduler
- c-c-m: Cloud controller manager
- c-m: Controller manager
- kubelet: Kubelet
- k-proxy: Kube-proxy

Uses a proprietary REST-based protocol

Conflict with the aforementioned standardization ideas
Approach

• Kubernetes - Container Network Interface (CNI)
  – Network connectivity of containers
  – It does not support changing the master-worker communication protocol.
Approach

- **api**: API server
- **c-c-m**: Cloud controller manager
- **c-m**: Controller manager
- **kubelet**: Kubelet
- **k-proxy**: Kube-proxy
- **ectd**: Scheduler
Experiment and Results

• Orchestrator, we used Kubernetes (v1.23.3).

• VMWare ESXi cluster, 3 Virtual Machines (VM) with Linux Ubuntu Server 20.04.

• We set up four variants, the original Kubernetes and one variant per proxied protocol.

• Kubernetes does not ship with a (required) CNI, we created a simple networking.

• Avoid known CNIs like Calico or Flannel, which may introduce unexpected side effects on the experiment.
Experiment and Results

• Kubernetes testinfra which include 7118 tests.

• **Conformance test** with 348 tests to validate the impact of our proxies on the overall functionality of Kubernetes.

• **Performance test** to evaluate the performance of our proxies on Kubernetes in comparison to the original Kubernetes.
Experiment and Results

- We followed the same procedure for a selected variant/test aspect.

  Start Proxies (Master and Worker) → Join worker nodes to cluster → Run the test → Stop Proxies (Master and Worker) → Remove worker nodes from cluster

- These procedure is repeated 100 times for the **Performance test**.
- Automated experiment execution and result collection.
- Zenodo Link [https://doi.org/10.5281/zenodo.7158281](https://doi.org/10.5281/zenodo.7158281)
Experiment and Results

- **Conformance test** execution time is about 89 minutes with a variation of around 6 minutes.

- **Performance test** has 5% of variance.

<table>
<thead>
<tr>
<th>K8s variant</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original K8s</td>
<td>5,083</td>
<td>5,163</td>
<td>5,108</td>
<td>0,018</td>
</tr>
<tr>
<td>AAS</td>
<td>5,086</td>
<td>5,159</td>
<td>5,108</td>
<td>0,017</td>
</tr>
<tr>
<td>AMQP</td>
<td>5,085</td>
<td>5,16</td>
<td>5,108</td>
<td>0,016</td>
</tr>
<tr>
<td>MQTTv5</td>
<td>5,085</td>
<td>5,152</td>
<td>5,107</td>
<td>0,017</td>
</tr>
</tbody>
</table>

Execution time is in ms
Further, the proxying also increases the resource usage, as we believe in acceptable manner.

We used **nload** and **top** Linux functionality for the measurements and it is collected over time.

<table>
<thead>
<tr>
<th>AAS proxy</th>
<th>Master</th>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU load</td>
<td>2.67%</td>
<td>1.35%</td>
</tr>
<tr>
<td>Memory usage</td>
<td>4.06%</td>
<td>2.43%</td>
</tr>
<tr>
<td>Network traffic</td>
<td>1.99%</td>
<td>0.51%</td>
</tr>
</tbody>
</table>
Conclusion

• We replaced the communication protocol in Kubernetes to IIoT/Industry 4.0 protocols and information model to increase standardization.

• As far as we know, our approach does not cause a major effects on typical functionality or performance of Kubernetes.

• Limitation: the experiment is done on the small testing cluster environment.
Future Work

• Protocol openness (without proxy detours) may be a future interoperability feature of the orchestrators.

• Apply similar approach to other orchestrators such as Docker Swarm or Mesos.

• Evaluate the scalability with different cluster sizes.

• Aim at integrating other relevant protocols, e.g., OPC UA.