A Comparative Analysis of Open-source Serverless Computing Frameworks

***Abstract:** This document presents an in-depth, data-driven comparative analysis of two leading open-source serverless platforms—Knative and Fission—deployed on Kubernetes. Our evaluation focuses on three core dimensions: Performance: Including cold start latency, throughput scalability, and resource efficiency. Cost: Evaluating direct costs versus the hidden overhead of Kubernetes cluster management. Developer Experience: Assessing tooling, documentation, and overall ease of deployment. Key findings indicate that while Fission achieves signifcantly lower cold start times (≈0.13s vs. 2.65s), it faces challenges at high concurrency levels (notably, a 9.8% error rate under stress). In contrast, Knative exhibits superior throughput (up to 48.84 req/s at high concurrency) and is approximately three times more CPU efficient. These trade-offs provide clear guidance for selecting the platform that best matches specific workload requirements. The growing popularity of serverless architectures is driven by the promise of scalable, cost-effective, and vendor-neutral computing. However, while commercial solutions like AWS Lambda are well-documented, open-source alternatives such as Knative and Fission remain underexplored in the literature. This analysis addresses that gap by quantifying performance metrics, including cold start latency and throughput under varied load conditions, evaluating resource utilisation (CPU and memory) that directly impacts operational costs, and comparing the developer experience, considering the learning curve and the quality of available tooling and documentation.*
By benchmarking these platforms under identical conditions, this work provides actionable insights for organisations considering an open-source Kubernetes-based serverless platform.

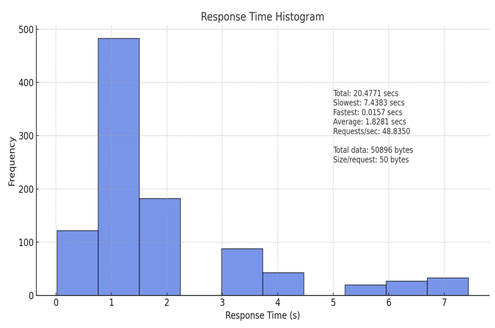

Figure 1. Data visualisation for high concurrency throughput. Knative Function.

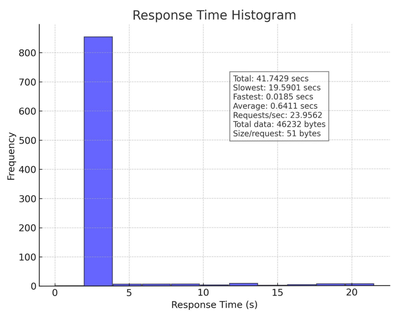

Figure 3. Data visualisation for high concurrency throughput. Fission Function.

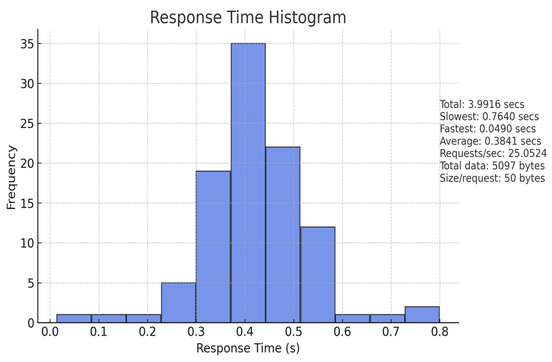

Figure 2. Data visualisation for low concurrency throughput. Knative Function.

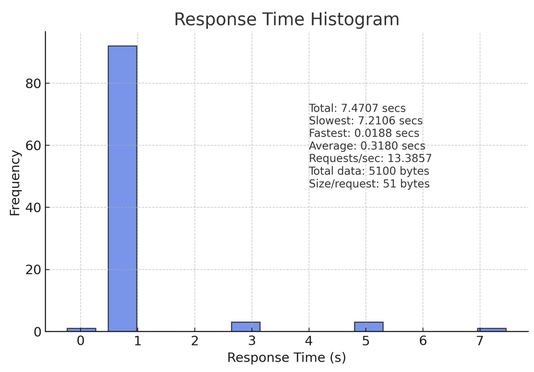

Figure 4. Data visualisation for low concurrency throughput. Fission Function.

Findings

Fission achieves near-instant startup with an average cold start latency of approximately 0.13 seconds, benefiting from its lightweight container initialisation. In contrast, Knative experiences a delay of around 2.65 seconds due to the time required to scale from zero. In terms of throughput, Knative outperforms Fission at low concurrency (10 requests), processing 25.05 requests per second compared to Fission's 13.39 requests per second. Under high concurrency (100 requests), Knative scales effectively to 48.84 requests per second with zero errors, whereas Fission reaches 23.96 requests per second but suffers from a 9.8% error rate. Regarding resource efficiency, Knative demonstrates significantly lower CPU utilisation at 9.48%, making it approximately three times more efficient than Fission, which consumes 27.55%. This suggests Knative may offer lower operational costs for sustained workloads. However, memory consumption is similar across both platforms, averaging around 43.3%. Although both platforms are free to use, the total cost of ownership extends beyond their initial availability. Kubernetes cluster management incurs expenses, whether through cloud providers such as EKS or GKE for on-premise deployments. Additionally, operational overhead must be considered, as maintaining and troubleshooting Kubernetes environments requires specialised expertise and labour. Knative offers mature tooling with a robust CLI (kn, func) and strong Kubernetes integration, while Fission provides a unified CLI that simplifies operations but lacks advanced debugging tools. In terms of documentation, Knative offers comprehensive resources but assumes prior Kubernetes expertise, whereas Fission's documentation has some gaps but is more accessible for beginners. The learning curve for Knative is steeper, particularly for developers without a Kubernetes background, whereas Fission is more beginner-friendly but offers fewer advanced features. Knative's advanced tooling and scalability make it well-suited for high-throughput applications, while Fission's rapid cold starts make it a strong choice for latency-sensitive and edge computing scenarios.

Conclusion/Future Work

This comparative analysis highlights key strengths and trade-offs between the two platforms. Fission is best suited for applications where minimising cold start latency is critical, such as IoT or real-time edge computing, though its scalability under heavy load is limited. In contrast, Knative provides superior throughput and CPU efficiency, making it more suitable for high-traffic back-end services that benefit from dynamic scaling.

Test Configuration

Both platforms were deployed on dedicated Kubernetes clusters with the default configurations. Knative used the kn, func CLI and Kubernetes, running a Python function for light data processing including JSON parsing, loop iterations and a mock database lookup. Fission used the Fission CLI and Kubernetes for deployment, executing an identical Python function. For cold start latency, after a 300-second idle period, the function was invoked 10 times. Fission had an average cold start time of approximately 0.13 seconds, whereas Knative took around 2.65 seconds. Fissions low cold start is due to the configurable pool of containers, so functions have very low cold-start latencies, typically ~100msec and Knaitves higher cold start is due to the scale down to zero behaviour based on traffic patterns. Throughput was measured using the hey HTTP load generator under different levels of concurrency. At low concurrency (100 requests and 10 concurrently), Knative processed about 25.05 requests per second, while Fission handled around 13.39 requests per second. At high concurrency (1000 requests and 100 concurrently), Knative achieved approximately 48.84 requests per second, whereas Fission reached about 23.96 requests per second but experienced a 9.8% error rate. This highlights Knaives supperior handling of high traffic loads. Resource usage was tracked using Python's psutil module, averaging measurements over 100 requests. In terms of CPU consumption, Knative utilised about 9.48%, while Fission used approximately 27.55%. Memory usage was similar for both platforms, averaging around 43.3%.

```
Fission Cold Start Latency Measurements
Request 1 cold start latency: 0.1161 seconds
Request 2 cold start latency: 0.1280 seconds
Request 3 cold start latency: 0.1245 seconds
Request 4 cold start latency: 0.1577 seconds
Request 5 cold start latency: 0.1238 seconds
Request 6 cold start latency: 0.1187 seconds
Request 7 cold start latency: 0.1337 seconds
Request 8 cold start latency: 0.1287 seconds
Request 9 cold start latency: 0.1287 seconds
Request 10 cold start latency: 0.1173 seconds
Average cold start latency over 10 requests: 0.1277 seconds
```

Figure 5. Cold start latency test results for 10 requests. Fission Function.

```
Knative Cold Start Latency Measurements
Request 1 cold start latency: 2.5195 seconds
Request 2 cold start latency: 2.4268 seconds
Request 3 cold start latency: 2.4241 seconds
Request 4 cold start latency: 2.4396 seconds
Request 5 cold start latency: 2.5841 seconds
Request 6 cold start latency: 2.4663 seconds
Request 7 cold start latency: 2.5517 seconds
Request 8 cold start latency: 3.1391 seconds
Request 9 cold start latency: 3.3552 seconds
Request 10 cold start latency: 2.5785 seconds
Average cold start latency over 10 requests: 2.6485 seconds
```

Figure 6. Cold start latency test results for 10 requests. Knative Function.