Cloud-Native Serverless Computing with Knative on Kubernetes

Zheng Yang
NanJing University of Science and Technology
Email: 202383930030@nuist.ecu.cn

Abstract

Serverless computing has become an important paradigm in modern cloud computing by allowing developers to focus on application logic without managing underlying infrastructure. While commercial serverless platforms provide convenience and scalability, they often introduce vendor lock-in and limited portability. Knative is an open-source, cloud-native framework that enables serverless workloads on top of Kubernetes. This report presents a comprehensive study of Knative, including its architecture, core components, and practical usage scenarios. Through experimental observations and analysis, we evaluate the advantages and limitations of Knative in cloud-native environments and discuss its potential role in future serverless platforms.

1. Introduction

Cloud computing has fundamentally transformed how software systems are developed and deployed. Early cloud infrastructures primarily relied on virtual machines, which provided flexibility but still required significant manual configuration and operational effort. With the emergence of containers and orchestration platforms such as Kubernetes, cloud-native architectures have become the dominant approach for building scalable and resilient applications.

Serverless computing represents a further abstraction in this evolution. By removing the need to provision and manage servers, serverless platforms allow developers to concentrate on business logic while benefiting from automatic scaling and resource optimization. However, most popular serverless platforms, such as AWS Lambda and

Azure Functions, are tightly coupled with specific cloud providers. This coupling introduces challenges related to portability, extensibility, and long-term vendor lock-in.

Knative addresses these challenges by providing serverless capabilities directly on Kubernetes. As an open-source project, Knative enables developers to deploy and manage serverless workloads in a platform-agnostic manner. By leveraging containers as the execution unit, Knative supports multiple programming languages and integrates seamlessly with existing cloud-native ecosystems.

This report explores Knative as a cloud-native serverless framework. We examine its design principles, architectural components, and operational behavior. The report also discusses experimental observations and evaluates the suitability of Knative for modern cloud-native applications.

2. Background and Related Work

Serverless computing, also referred to as Function-as-a-Service (FaaS), allows applications to be executed in response to events without explicit server management. Major cloud providers have popularized this model through managed platforms that automatically handle scaling, availability, and fault tolerance.

Despite their convenience, proprietary serverless platforms often lack flexibility and transparency. Developers have limited control over runtime environments, and applications are tightly bound to provider-specific APIs and services. To address these issues, several open-source serverless frameworks have been proposed, including OpenFaaS, Kubeless, and Apache OpenWhisk.

Knative distinguishes itself by adopting a strictly cloud-native approach. Instead of introducing a separate execution platform, Knative extends Kubernetes using custom

resource definitions and controllers. This design allows serverless services to coexist with traditional containerized workloads in the same cluster.

Recent research and industrial practice emphasize the importance of portability and standardization in cloud-native systems. Kubernetes has become the de facto standard for container orchestration, motivating the development of serverless solutions that build on its abstractions. Knative aligns with this trend by offering serverless capabilities while preserving Kubernetes' declarative configuration model.

3. Knative Architecture

Knative consists of modular components that collectively enable serverless execution on Kubernetes. The two primary components are Knative Serving and Knative Eventing, with Knative Functions providing a higher-level developer experience.

Knative Serving manages stateless services and provides features such as automatic scaling, scale-to-zero, revision management, and traffic splitting. Each Knative Service abstracts underlying Kubernetes resources, allowing users to deploy applications without directly managing Pods or Deployments.

Knative Eventing supports event-driven architectures by enabling asynchronous communication between producers and consumers. It introduces concepts such as brokers and triggers, which decouple event sources from event handlers and improve system flexibility.

From an architectural perspective, Knative extends Kubernetes through custom resource definitions. These resources enable declarative configuration and lifecycle management of serverless services. At runtime, components such as autoscalers and activators manage traffic flow and scaling behavior.

Networking plays a critical role in Knative's architecture. Incoming requests are routed through a networking layer that ensures services remain reachable even when scaled to zero. Observability is also integrated through standard cloud-native tools, enabling monitoring and debugging of serverless workloads.

4. Experiment Setup and Use Cases

To evaluate Knative in practice, a local Kubernetes environment is created using Minikube. Knative Serving and Eventing are installed following official documentation. A simple containerized application is deployed as a Knative Service to demonstrate automatic scaling behavior.

When the service receives incoming requests, Knative automatically increases the number of active replicas. As traffic decreases, the service scales down and eventually reaches zero replicas. This behavior demonstrates the core serverless property of on-demand resource allocation.

Event-driven use cases are also explored by configuring Knative Eventing components. Events sent to a broker are routed to services based on predefined rules, illustrating asynchronous and loosely coupled communication patterns.

These experiments reflect common cloud-native development scenarios. The results highlight Knative's ability to adapt dynamically to workload changes while maintaining efficient resource usage.

5. Performance Analysis and Observations

Performance remains a critical metric when evaluating serverless platforms, as it directly impacts user experience and operational efficiency. Experimental observations indicate that Knative delivers consistent and stable performance once services are running, with minimal response variability. However, cold start

latency—experienced when services scale from zero replicas—remains a measurable overhead. This latency is influenced by multiple factors, including the size of container images, the complexity of initialization logic, and the current availability of cluster resources.

Although cold start times are often acceptable for asynchronous, background, or event-driven workloads, they can negatively affect latency-sensitive, user-facing applications. In practice, mitigating cold start overhead may involve optimizing container images by minimizing dependencies, pre-warming pods, or fine-tuning scaling parameters. Despite this limitation, Knative demonstrates predictable scaling behavior and efficient resource utilization. Unlike traditional always-on services, its ability to scale to zero not only reduces idle resource consumption but can also significantly lower infrastructure costs, especially in dynamic environments with fluctuating workloads.

Overall, Knative strikes a balance between operational efficiency and performance predictability, making it a viable choice for many cloud-native scenarios, provided that cold start considerations are adequately addressed.

6. Practical Implications for Cloud-Native Applications

Knative is particularly well suited for workloads that benefit from elasticity and on-demand execution. Typical use cases include event-driven data processing pipelines, lightweight RESTful APIs, microservices that respond to external triggers, and background tasks initiated by queues or messaging systems. Its serverless abstraction enables developers to focus on business logic while the platform handles scaling and resource management.

For teams already leveraging Kubernetes, Knative integrates naturally into existing workflows. Continuous integration and deployment pipelines can be extended to include serverless services without introducing proprietary tooling or vendor lock-in. This allows organizations to maintain a consistent DevOps experience while gradually adopting serverless patterns.

However, Knative may not be the ideal solution for all workloads. Stateful services, such as databases or applications with in-memory caches, may struggle under serverless scaling constraints. Similarly, latency-sensitive services requiring near-instant response times may suffer from cold start delays. Therefore, careful workload analysis is essential to identify which services are best suited for Knative. Strategic decisions may involve combining serverless and traditional always-on deployments to achieve a balanced architecture that meets both performance and cost objectives.

7. Discussion and Challenges

While Knative provides powerful abstractions for building serverless applications, it introduces additional operational complexity. Installation and configuration require a deep understanding of Kubernetes internals, networking components, and container orchestration principles. Debugging can also be more challenging due to multiple abstraction layers between the developer and the underlying infrastructure, often necessitating advanced monitoring and logging tools.

Cold start latency remains one of the most persistent challenges, particularly for applications that scale frequently from zero. Mitigation strategies include optimizing container images, leveraging warm pools, and fine-tuning autoscaling parameters. Another consideration is the trade-off between scalability and resource predictability; while Knative can dynamically scale to zero, highly bursty workloads may still experience short-term latency spikes.

Despite these challenges, Knative offers a flexible, open-source alternative to proprietary serverless platforms, providing organizations with freedom from vendor lock-in and the ability to run workloads across hybrid or multi-cloud environments. Its deep integration with Kubernetes ensures that cloud-native teams can leverage existing skills and tools while gradually adopting serverless paradigms, making it a compelling choice for organizations aiming to modernize their application architectures.

8. Conclusion

This report provided a comprehensive examination of Knative as a cloud-native serverless framework built on top of Kubernetes. By integrating automatic scaling, scale-to-zero capabilities, event-driven execution, and container-based deployment, Knative offers a flexible and portable approach to serverless computing that aligns well with modern cloud-native design principles. Unlike proprietary serverless platforms, Knative preserves platform independence while enabling developers to leverage the rich ecosystem of Kubernetes.

Through architectural analysis and experimental observations, this study demonstrated that Knative is particularly effective for stateless and event-driven workloads that benefit from elastic scaling and on-demand resource allocation. The ability to dynamically adjust resource usage in response to workload fluctuations highlights Knative's potential for improving infrastructure efficiency and reducing operational costs in cloud-native environments.

At the same time, this report also identified several limitations that must be considered when adopting Knative in practice. Cold start latency remains a notable challenge, especially for latency-sensitive applications. In addition, the operational complexity associated with Kubernetes-based platforms can pose a learning barrier for teams without sufficient cloud-native experience. These factors suggest that

Knative is best suited for carefully selected workloads rather than as a universal replacement for all application deployment models.

Looking forward, several directions for future research and development emerge from this study. Performance optimizations aimed at reducing cold start overhead could significantly enhance Knative's suitability for interactive applications. Improvements in developer tooling and observability may further lower the operational burden and simplify debugging and maintenance. Moreover, the integration of Knative with emerging domains such as machine learning inference, data streaming, and edge computing represents a promising area for continued exploration.

In conclusion, Knative represents an important step toward open, cloud-native serverless computing. When applied appropriately, it enables scalable, portable, and efficient application deployment on Kubernetes, making it a valuable component of the evolving cloud-native ecosystem.

References

Knative Documentation. https://knative.dev

Cloud Native Computing Foundation. Cloud Native Landscape. https://landscape.cncf.io

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J. Borg, Omega, and Kubernetes. Communications of the ACM, 2016.