A Comprehensive Evaluation of Helm and Helmfile for Efficient Management of Microservices

Trinath Chakka

School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland

X00193221@myTUDublin.ie

Introduction

Microservices are supported by Kubernetes, a popular container orchestration platform run by the Cloud Native Computing Foundation that offers capabilities including automated deployment and scalability. This research evaluates critical performance aspects, focusing on chart execution time and service uptime on Helm vs Helmfile. The goal of the study is to get a better knowledge of Helm and Helmfile's respective advantages and disadvantages when it comes to coordinating deployments within Kubernetes frameworks by analysing various deployment methodologies.

Deployment Comparision Table

Model (Services)	Branches-Runs	Average Time for all branches
Helm Small (5)	Main-10	3m 59s
	Downgrade-10	3m 30s
	Upgrade-10	3m 31s
	Rerun-10	2m 17s
	Total = 40	Total:13m
Helm Medium (25)	Main-5	18m 6s
	Downgrade-5	20m 51s
	Upgrade-5	18m 40s
	Rerun-5	7m 7s
	Total = 20	Total:64m
Helm Large (40)	Main-3	28m 33s
	Downgrade-3	31m 59s
	Upgrade-3	30m 19s
	Rerun-3	16m 54s
	Total = 12	Total: 107m
Helmfile Small(5)	Main-10	4m 21s
	Downgrade-10	3m 25s
	Upgrade-10	3m 6s
	Rerun-10	1m 51s
	Total = 40	Total:12m
Helmfile Medium (25)	Main-5	16m45s
	Downgrade-5	15m50s
	Upgrade-5	13m18s
	Rerun-5	3m15s
	Total = 20	Total:49m
Helmfile Large (40)	Main-3	32m 28s
	Downgrade-3	20m 02s
	Upgrade-3	15m 22s
	Rerun-3	6m 40s
	Total = 12	Total:74m

Research Questions

RQ1 - How do chart execution times differ between Helm and Helmfile, particularly when small, medium, and large components application scenarios are deployed?

The outcomes clearly show that Helmfile consistently outperforms Helm, with deployment efficiency across small, medium, and large models increasing by a factor of three. Helmfile's continuous advantage over Helm is indicated by its faster deployment times. Helmfile's efficiency benefits are highlighted by the significant time savings seen in all deployment sizes, making it an appealing choice for enterprises looking to streamline Kubernetes deployment procedures. Helmfile's continued exceptional performance confirms its continued relevance as a tool of choice for orchestration in a variety of deployment situations.

RQ2 - What effect do Helm and Helmfile have on Kubernetes application service uptime, and how do their performances differ in terms of maintaining dependable and continuous service availability?

In downgrade and upgrade circumstances, Helmfile regularly performs better than Helm, but Helm typically delivers quicker service uptime, especially in big branches. Local state management and the declarative nature of Helmfile help to speed up service uptime during upgrades. This sophisticated knowledge highlights the need of picking orchestration tools customised to particular requirements for continuous service availability in a variety of Kubernetes environments, enabling organisations to make decisions between Helm and Helmfile based on their unique deployment needs.

RQ3 - How does Helmfile's declarative approach and local state maintenance effect the speed and reliability of complex deployment scenarios, and how does it handle Helm's management challenges?

In complex deployment scenarios, Helmfile's declarative style and local state management greatly improve speed, reliability, and efficiency over Helm. Helm's drawbacks are addressed by Helmfile, which enables faster and more effective deployments by maintaining a local state and expressing the planned deployment state reliably. This is especially useful in large-scale installations. The ability of Helmfile to track changes at a finer level is useful for managing version transitions and gives enterprises an easier-to-use orchestration tool for complex Kubernetes installations.

Test Strategy

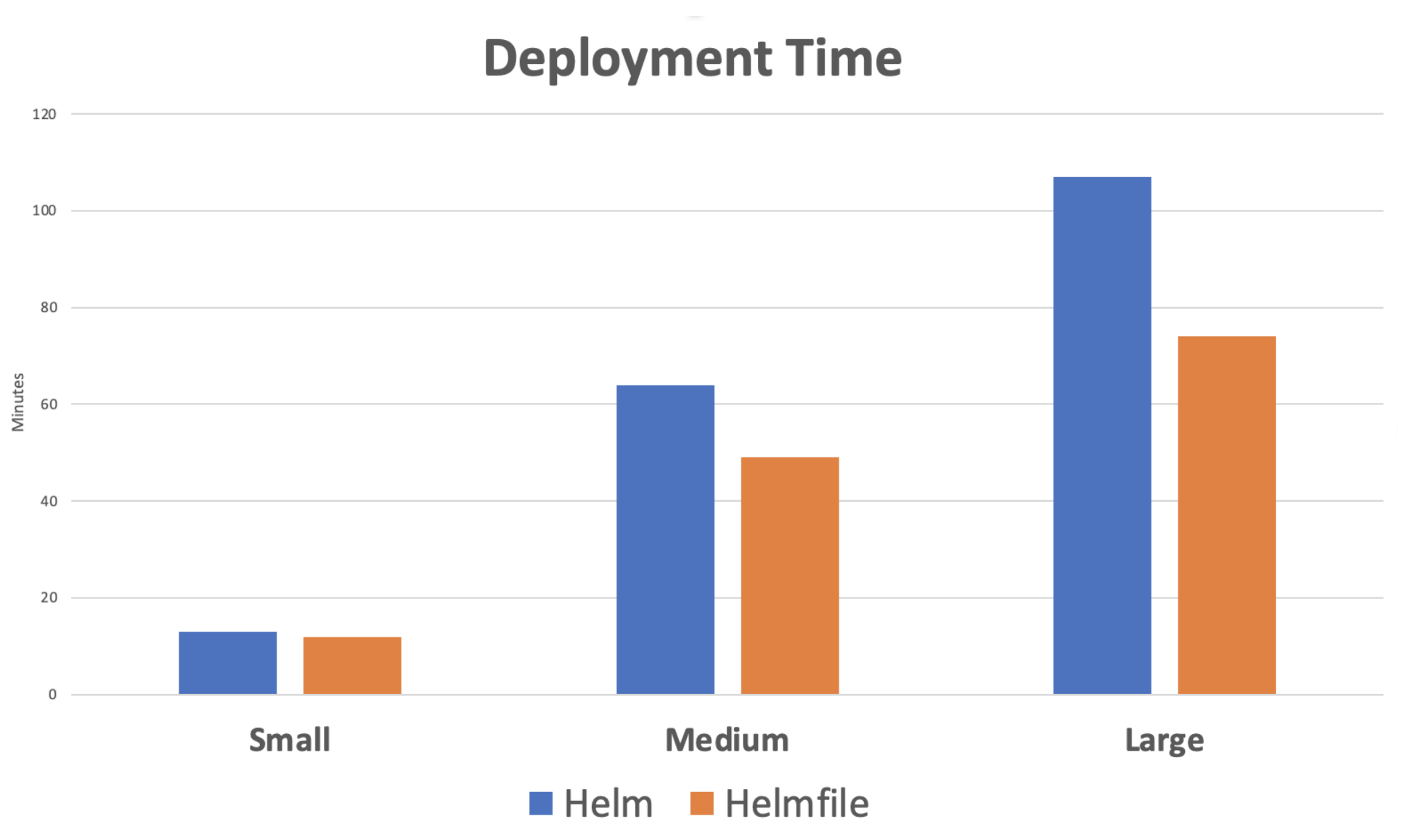
The experiment was carried out on Azure, using Azure DevOps for rigorous testing. The infrastructure was divided into two separate projects for Helm and Helmfile, with repositories aimed to small, medium, and large applications. Each repository, constructed with 5, 25, and 40 services. To enhance precision and robustness, we conducted 10 cycles for the small branch, 5 cycles for the medium branch, and 3 cycles for the large branch. Each repository included four important branches: "main", "downgrade", "upgrade" and "rerun". The results of all tests are gathered in the deployment comparison table above and the corresponding picture for a visual representation is available here.

Main Branch: Latest Chart version.

Downgrade Branch: Last Stable version for compatibility validation.

Upgrade Branch: Same versions as main branch for compatibility check.

Rerun Branch: Evaluates deployment speed with no version changes



Conclusions and Future Work

Helmfile shows to be more effective and adaptable in small, medium, and large-scale Kubernetes installations, improving dependability and deployment times. Helmfile's attraction for easier Kubernetes administration, especially in large-scale deployments, is highlighted by rich insights provided by statistical analysis and graphical representations. Helmfile beats Helm in deployment performance by using locally cached resources, cutting down on the times associated with Helm's sequential approach.

Future endeavors should prioritize enhancing compatibility, scalability testing, CI/CD integration, security fortification, and user experience. These cooperative projects are essential to Helm and Helmfile's ongoing development and relevance in the changing Kubernetes orchestration scene. More specifically, it is critical to prioritise compatibility with changing Kubernetes versions in order to provide smooth upgrades and feature integration.

QR Code for Recording