Syn: GitOps on Stereoids with Kubernetes the Swiss Way

Josef Spillner[1], Daiana Boruta[1], Tobias Brunner[2], Simon Gerber[2], and Adrian Kosmaczewski[2]

[1] Zurich University of Applied Sciences, Winterthur, Switzerland
{josef.spillner,boru}@zhaw.ch
[2] VSHN AG, Zurich, Switzerland
{tobias.brunner,simon.gerber,adrian.kosmaczewski}@vshn.ch

Abstract

Kubernetes has become one of the leading platforms for deploying and operating microservice-based applications. With its native support for containers and cloud functions, it has strong service scheduling, scaling and isolation features. But deploying to Kubernetes is a path riddled with stumbling blocks. Syn automates Kubernetes with fully controlled and secured configuration management for multiple tenants, enabling reproducibility of issues and increasing reliability and flexibility in multi-cloud deployments.

1 Introduction

Application engineers and providers are facing a wide-spread transition to Kubernetes (K8s), a platform specifically designed for the messaging, elasticity, security and resilience needs of container-native microservices [3, 9, 10]. Applications deployed on K8s are described as sets of declarative manifest files referencing containers in private or public container registries as tangible microservice units. These manifests may be packaged in Helm charts, CNAB files or TOSCA/Toskose packages, to mention a few representatives of a fast-moving field [2]. These are non-K8s-native formats requiring further tooling and sometimes K8s extensions, and accordingly presuppose additional software engineering skills that demand costly training for ephemeral applicability.

On the operational level, K8s runs clusters of pods along with further resource objects, including user-defined resources, as logical management units [1]. Pods are wrapping deployed containers, their lifecycle states and elastic scaling states. Some of the containers have microservices interfaces (in particular cloud functions and operators wrapped as networked containers [7]) while others constitute non-invokable processes and auxiliary management containers (sidecars, one-off/init, cron) but are still colloquially called microservices in industry parlance. K8s has little understanding of the entirety of an application and the semantics of application liveness, as it merely manages the constituents objects including the microservices that form the application core on a syntactic level.

Deploying manifests or packages to Kubernetes without additional DevOps support frameworks has a number of disadvantages beyond the limited semantic interpretation of applications: (i) No transactionality, leaving semi-deployed objects in cases of errors; (ii) No immediate teardown and replay and thus no reproducibility; (iii) No enforcement of naming policies, tenant separation, change paths and other operational concerns, revealing great risks and attack vectors from a cyber-security perspective.

Syn combines years of DevOps experience in containerised environments [5] with current best practices to overcome these limitations. It combines GitOps deployment [6] of K8s object definitions onto immutable infrastructure, event-triggered K8s operators for increased automation, multi-cloud deployments, dependencies, secrets management and other security concerns,

as well as monitoring and alerting under one hood. On top, it offers unique concepts not yet found in any other container management platform with the goals to increase automation, reproducibility, reliability and flexibility.

In the following sections, we introduce the Syn concepts (Sect. 2), describe its implementation (Sect. 3), and evaluate the bootstrap and deployment performance (Sect. 4). We explain the creation of custom components (Sect. 5) before concluding with a look at possible next steps (Sect. 6).

2 Concepts

Syn itself is based on a microservices composition and introduces four novel concepts beyond state of the art container management platforms and deployment tools to improve the DevOps process:

- Bring Your Own Cloud (BYOC) with high modularity and thus flexibility. By specifying API endpoints to self-hosted Syn components, engineers can involve their own container registry or their own secrets management vault into the platform's workflows. The conventional binary separation between fully provider-managed and self-managed application platform is thus becoming a more fine-grained set of economic and technical decisions on a spectrum. Consequently, the concept of loose coupling is propagated from composite microservice-based applications to the underlying management platform itself, greatly streamlining the ability to control the deployment and provisioning through commits by developers. Components and the extensibility they bring to the platform are described in greater detail in Sect. 5.

- Explicit local mode. Configuration changes can pass a dry run followed by an analysis of which changes would be transitively propagated to which parts of the system. This alleviates the need to perform costly rollbacks.

- Strong multi-cloud support. Syn keeps centralised knowledge about an arbitrary number of K8s clusters running on various IaaS deployments and is aware of version and distribution flavour differences as well as tenants. Microservices are safely deployed with K8s object manifests generated and customised using this knowledge.

- Hierarchical code-driven development: Multiple Git repositories of various types are managed by Syn. The security of both repositories and application deployments is eased with internal credentials handling through confidential transit and storage. The breach of one server, one cluster or one infrastructure cloud provider limits any potential damage to this realm. In the wake of increasing global damage through cloud intrusions [8], this is an important property.

With these concepts, Syn supports a combination of multi-cluster and multi-tenant management, full GitOps operations, tooling bootstrapping, configuration management and reusable components, including for automatic maintenance and renovation of components and secrets management and unified service provisioning within the K8s cluster and across clouds.

The GitOps focus is evident from Fig. 1 that explains the basic entity relationships. Tenants and clusters are automatically created, and also torn down, through configuration stored in Git repositories. The current iteration of Syn supports private clusters, offering high isolation to security-sensitive tenants. A future option is the operation of shared clusters to increase the

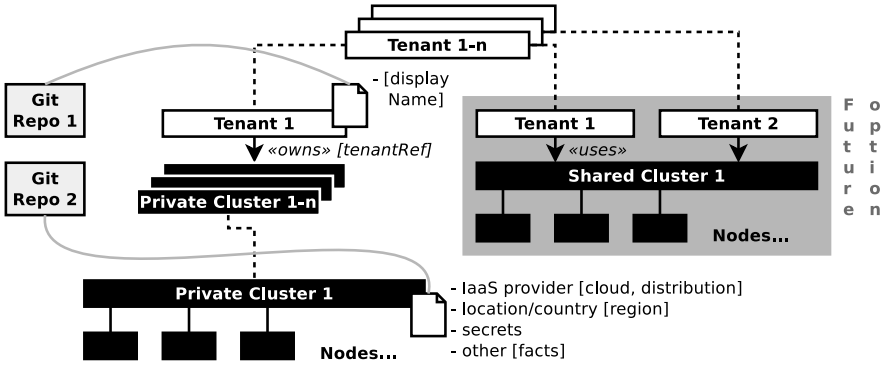

Figure 1: System model with tenants, clusters and repositories as main entities

utilisation rate. Moreover, the BYOC and multi-cloud concepts are also represented. Syn may be instructed to deploy a cluster on the `rma1` region of the `cloudscale` provider using the `openshift3` distribution of K8s. Once a cluster is set up, applications can be deployed to it. Addressing multiple clusters in a federation in a unified way during this deployment is still a challenge, as tools like `kubectl` are bound to single clusters. This limitation is increasingly solved with evolving approaches like KubeFed (Kubernetes Federation v2)[1] and corresponding research works on using multiple clusters [4].

3 Implementation

Syn has been implemented as a set of modular, loosely coupled modules and APIs around existing cloud-native stacks and tools such as Kubernetes (K8s), Kapitan, ArgoCD, Crossplan, Renovate and Vault as well as GitLab. The integration architecture around the main Syn parts (Lieutenant API and operator, Steward and Commodore) is shown in Fig. 2. Dashed parts of the system are not currently implemented but might complement the platform functionality in future versions.

Delivering a new microservices-based and containerised application means first preparing a K8s cluster. The preparation starts with registering arbitrary facts with Lieutenant. These include facts about available machines via Steward and confidential facts such as the machine SSH key pairs stored in Vault. Lieutenant then provides its inventory to the manifest generator Commodore via GitLab, creating and curating tenant and cluster repositories as needed. Commodore pulls all details from Git repositories to build the consolidated cluster catalogue, taking hierarchical configuration overrides (from tenant over cluster to the application level) into account. The build is then triggering the continuous delivery of the managed application on K8s through ArgoCD, adding externally managed services through Crossplane as needed. Eventually, the containerised application is deployed in a cloud-native way, exploiting the capabilities of both the K8s cluster and external infrastructure through appropriate distribution of containers and bindings to stateful services such as databases and message brokers.

Syn, named after the Swedish word for vision, is developed as open source project (documented at `http://syn.tools`) by VSHN AG, a DevOps company responsible for the multi-cloud hosting of customers on more than 15'000 machines with approximately 85'000 concur-

[1] KubeFed: `https://github.com/kubernetes-sigs/kubefed`

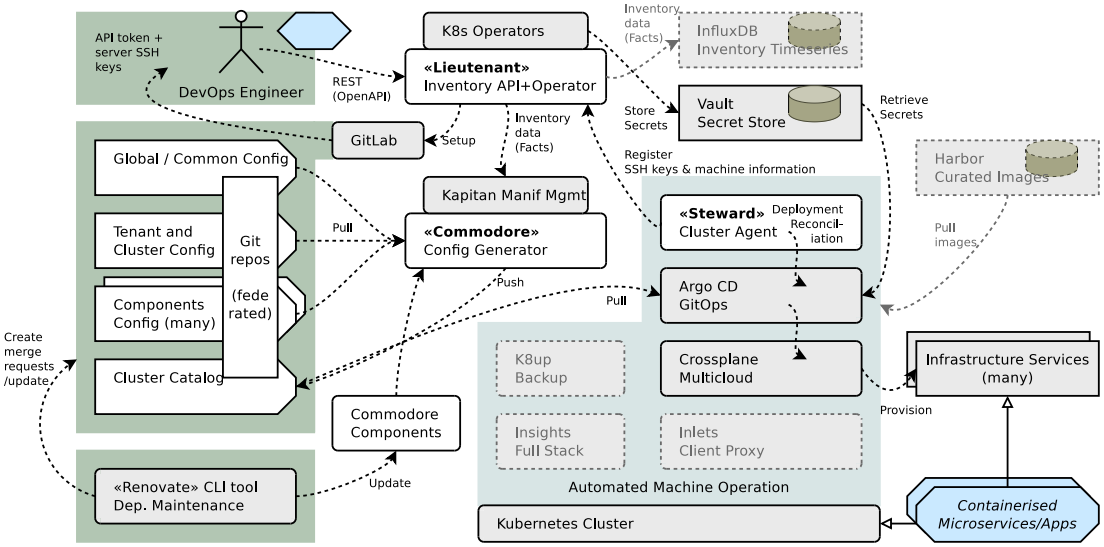

Figure 2: Syn integration architecture based on Kubernetes ecosystem components

rently running managed services. Customers choose to be switched to Syn at their own pace. While still under development, the first customers have proven the concepts of Syn in production on multiple clouds since April 2020, and Kubernetes fleets of increasing size are managed by Syn with reduced effort.

4 Evaluation

We demonstrate the degree of automation and reproducibility with a scenario. The procedure follows the instructions of *Getting Started with Project Syn*[2] and further automates them, reducing some of the choices to a sensible default in an evaluation context. The time to deploy an application is measured repeatedly and the predictability of the timing is determined. The reproducibility is checked by a combination of cleaning the environment (deleting Lieutenant objects that tear down the Git repositories, an deleting the cluster) as well as idempotency policies (`image-pull-policy=Always` in K8s). All experiments are conducted on a virtualised Intel Core (Broadwell) processor with 2400 MHz and 2 GB RAM atop a K3d-managed K3s instance, a lightweight distribution of K8s.

Fig. 3 shows the performance of bootstrapping Syn before and after attachment to the Git repositories. The total average time for bootstrapping is 94.3 s and includes the K8s cluster setup, the deployment of the ingress controller and the Lieutenant API deployment. Running Lieutenant to set up the repositories and Commodore to build and push the catalogue brings the total average time to 140.3 s. Kickstarting Steward to get the Syn instance GitOps-manage and to serve ArgoCD eventually leads to a significant increase and a total average time of 306.6 s.

Most bootstrapping steps show a stable performance, although the busy-waiting loops for the deployment of the Traefik ingress controller via Helm chart and ArgoCD are subject to both maximum slowness and deviation. Optimising these two phases could save almost 70% of

[2]Getting Started tutorial: https://syn.tools/syn/tutorials/getting-started.html

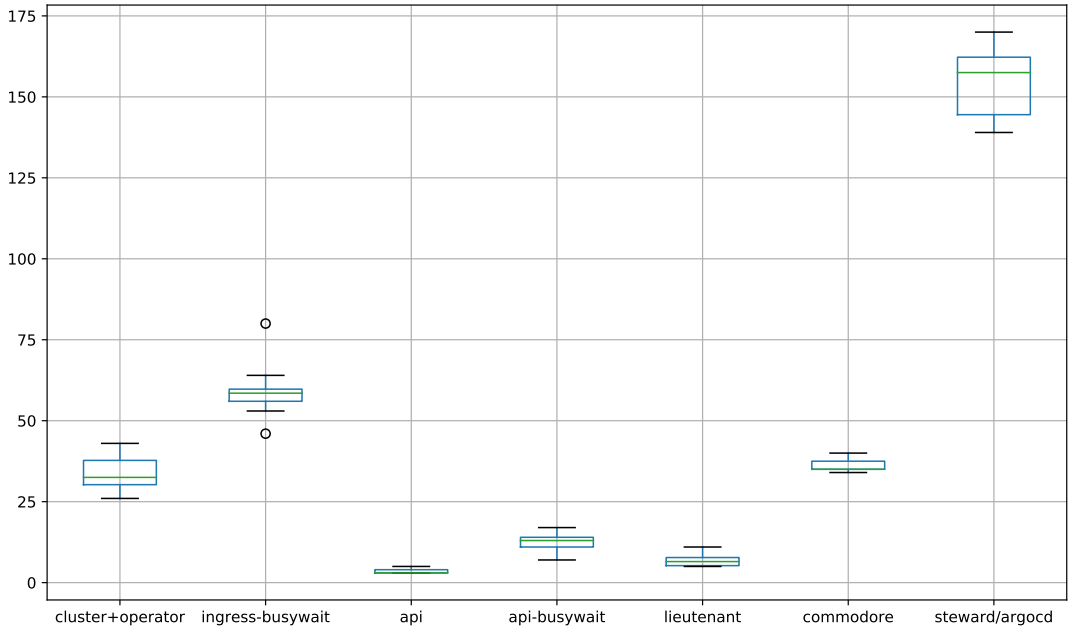

Figure 3: Measurement of deployment times (in s)

the overall readiness time. Traefik for instance is only required for K3s and can be eliminated for Minikube. As K8s distributions evolve, the extent of necessary pre-configuration of the base container platform still needs to be determined.

The scripts and reference data to reproduce the evaluation are available through Renku[3].

5 Custom Components

To perform GitOps at scale, as in any software development process it is necessary to foster reuse and parameterisation. Commodore components, an IaC abstraction over K8s manifest contents, are used to inject additional content through the compiled configuration catalogue.

While Commodore understands tenants and clusters natively, it can be taught other concepts through such custom components. Components are described with the Jsonnet template language along with variable substitutions and functions described in YAML files. Alternatives such as Jinja2, Kadet and Helm are also supported. Components access the inventory of Kapitan, underlying Commodore, and subsequently modify arbitrary configuration values in the hierarchy which are then reflected in the resulting K8s manifests. An representative component is the OpenShift 4 Registry. It creates a `namespace` object whose annotations include network policy labels depending on whether or not this was configured in the inventory. Other components exist for handling storage classes, modifying network policies, adding the K8s metrics server, Prometheus monitoring adapters and other cross-cutting functionality. A guide on how to create custom components is available online[4].

[3]Renku Collaborative Data Science: https://renkulab.io/projects/js.zhaw/syn-experiments
[4]Commodore components: https://syn.tools/commodore/writing-a-component.html

6 Conclusions and Future Work

Syn can be considered one of the most comprehensive frameworks for allowing GitOps at scale under real-world conditions where tenants need to be separated and tenant preferences (cloud provider X, configuration flag Y, Kubernetes extension Z) need to be considered. By automating these changes in a declarative manner without ad-hoc manual changes per cluster, Syn achieves both reproducibility and immutable infrastructure goals. The ability to administer multiple Kubernetes clusters in sync further contributes to increased reliability.

In the future, we plan to take the idea further towards predictable deployments through a-priori configuration checks, transactional rollouts (and rollbacks) of configuration, microservice quality examination and attestation-based container management.

References

[1] David Balla, Csaba Simon, and Markosz Maliosz. Adaptive scaling of Kubernetes pods. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–5. IEEE, 2020.

[2] Matteo Bogo, Jacopo Soldani, Davide Neri, and Antonio Brogi. Component-aware Orchestration of Cloud-based Enterprise Applications, from TOSCA to Docker and Kubernetes. *CoRR*, abs/2002.01699, 2020.

[3] Víctor Medel Gracia, Unai Arronategui, José Ángel Bañares, Rafael Tolosana, and Omer Rana. Modeling, Characterising and Scheduling Applications in Kubernetes. In Karim Djemame, Jörn Altmann, José Ángel Bañares, Orna Agmon Ben-Yehuda, and Maurizio Naldi, editors, *Economics of Grids, Clouds, Systems, and Services - 16th International Conference, GECON 2019, Leeds, UK, September 17-19, 2019, Proceedings*, volume 11819 of *Lecture Notes in Computer Science*, pages 291–294. Springer, 2019.

[4] Jiaming Huang, Chuming Xiao, and Weigang Wu. RLSK: A Job Scheduler for Federated Kubernetes Clusters based on Reinforcement Learning. In *2020 IEEE International Conference on Cloud Engineering, IC2E 2020, Sydney, Australia, April 21-24, 2020*, pages 116–123. IEEE, 2020.

[5] Hui Kang, Michael Le, and Shu Tao. Container and Microservice Driven Design for Cloud Infrastructure DevOps. In *2016 IEEE International Conference on Cloud Engineering, IC2E 2016, Berlin, Germany, April 4-8, 2016*, pages 202–211. IEEE Computer Society, 2016.

[6] Thomas A. Limoncelli. GitOps: a path to more self-service IT. *Commun. ACM*, 61(9):38–42, 2018.

[7] Ilias Mavridis and Helen D. Karatza. Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing. *Future Gener. Comput. Syst.*, 94:674–696, 2019.

[8] Md. Shazibul Islam Shamim, Farzana Ahamed Bhuiyan, and Akond Rahman. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. *CoRR*, abs/2006.15275, 2020.

[9] Josef Spillner and Daiana Boruta. Kubernetes Literature Dataset. Dataset v0.2, Zenodo, DOI: 10.5281/zenodo.3517806, October 2019.

[10] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned. In *11th IEEE International Conference on Cloud Computing, CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018*, pages 970–973. IEEE Computer Society, 2018.