To be presented at the 13th IEEE International Conference of Cloud Computing Technology and Science (CloudCom 2022), Bangkok, Thailand.

A Survey on Infrastructure-as-Code Solutions for Cloud Development

Håkon Teppan
IDE
University of Stavanger
Stavanger, Norway
hakon.teppan@gmail.com

Lars Halvdan Flå
Software Engineering, Safety and Security
SINTEF Digital
Trondheim, Norway
ORCID: 0000-0002-3069-6788

Martin Gilje Jaatun
IDE
University of Stavanger
Stavanger, Norway
ORCID: 0000-0001-7127-6694

***Abstract*—Cloud software is increasingly written according to the DevOps paradigm, where use of virtualization and Infrastructure-as-Code is prevalent. This paper surveys the state of the art of IaC cloud development, and proposes a combination of Cloud-Native software to build an on-premise PaaS for a Security Lab.**

***Index Terms*—Cloud, IaC, DevOps**

I. Introduction

Startup companies like Airbnb [1], Tesla [2], and Uber [3] have got a lot of traction in their specific industries due to more efficient ways to deliver their services [4]. They have in common to use Continuous Integration and Continuous Delivery (CI/CD) and the DevOps methodology to faster develop new features. An important part of DevOps and CI/CD is the concept of *Infrastructure-as-Code*, which enables the automatic creation and configuration of virtual computing resources using prepared scripts.

In a study done by Puppet in 2021 [5], it was found that DevOps increases the number of changes to a software product a company can do. The report grouped organizations in three categories based on how far in the "DevOps Evolution" they have come. The highest tier of these groups could provide changes in less than an hour, compared to between a week to six months for the lowest group, and less than a week for the mid-tier. This results in the developers having more time to work on new features. The highly evolved organizations also had less failure rate related to changes (less than 5 percent compared to 15 for the others).

A. Motivation

Cloud and Enterprise on-premise Kubernetes PaaS' could be expensive [6]. For a small team these solutions could be off the table. This project proposes an alternative.

This project uses Cloud-Native software from the provisioning of the host server, to configuring of the Kubernetes infrastructure, to the development of the applications. This solution will make it easier to add new functionality to the applications.

Research generally about GitOps and CI/CD is well documented [7] [8], but there is none found regarding a self-contained on-premises PaaS.

B. Problem Definition

Is a self-contained on-premise *container app as a service* a viable option to cloud and enterprise on-premise solutions?

C. Solution Approach

The lab solution is illustrated in Figure 1. In short, the user (A) connects to the application through the web browser. The application itself is hosted on the Kubernetes platform running on the physical server. The administrator and developer push changes to either the App Source Code or the Kubernetes Infrastructure Code (1). The commits will trigger the CI-pipeline (2), administrated by the GitLab repository. Afterwards, ArgoCD picks up the change, and changes the deployment accordingly (3). The Kubernetes platform K3s runs the environment based on the configuration given by ArgoCD (4).

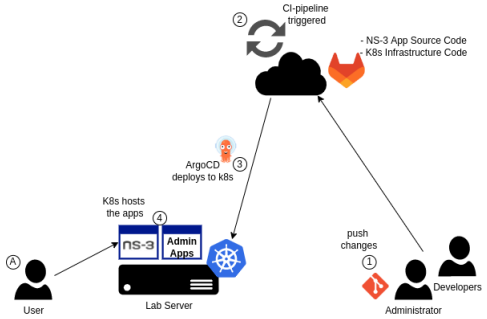

Fig. 1. Logical Overview

II. Cloud-Native Computing Foundation

Cloud-Native Computing Foundation (CNCF) is a Linux open-source foundation to help develop and support container-relevant technologies [9]. Cloud-Native is terminology for how applications should be designed to be optimized running on a container platform. It is a "best practice" guidance for container developers. Examples of Cloud Native platforms are k3s and Kubernetes.

The CNCF has collected all the projects they support, and sorted them by category [10]. The categories are *Provsioning, Runtime, Orchestration and Management, App Definition and*

Development, Observability and Analysis, and *Platform*. Each category has further subcategories. This paper will present the tools used in the lab for each category. The Runtime category will not be covered, because k3s includes the Containerd Runtime.

Furthermore, the projects get sorted in groups based on numerous criteria defined by the CNCF to indicate how stable they are for a production environment [9]. These groups are *Graduated, Incubating* and *Sandbox*. *Graduated* and *Incubating* projects are qualified to use in a production environment. CNCF distinguishes qualified projects in these two groups, to indicate which projects have newly been accepted, and those that have had this quality for a considerable time. A fourth option is for a project to be a member.

Technologies discussed in this paper, their maturity, and category in CNCF are summarized in Table I.

TABLE I
PROJECT MATURITY AND CATEGORY IN CNCF

Project	Relation	Category
Prometheus	Graduated	Observability and Analysis
Kubernetes	Graduated	Orchestration and Management
Helm	Graduated	App Definition and Development
ArgoCD	Incubating	App Definition and Development
Podman	Member	App Definition and Development
Traefik	Member	Orchestration and Management
MetalLb	Member	Orchestration and Management
K3s	Member	Platform
FCOS	-	Provisioning

III. Software Delivery Methodologies

Software Delivery explains the lifetime of a software, from the inital request from the user to the delivery of the finished software [11]. In the subsequent sections, the methodologies important for this thesis will be introduced.

A. DevOps

DevOps is a collective term for cultural and technical principals that enables organizations and teams to deliver services faster and agile [12]. It states how practices, people, and culture should be applied for delivering IT.

B. Continuous Integration and Continuous Deployment

Continuous Integration and Continuous Deployment (CI/CD) is a method in Information Technology (IT) for delivering a service through an agile workflow [13]. An increasingly popular implementation of CI/CD is GitOps [7].

1) GitOps: GitOps is a method to deliver operations to a DevOps workflow [7]. It was first described in 2018, and is increasingly popular by developers and operators, because of the agile way of creating and realising changes to a production network. In GitOps, the source code of the given project is described in a git repository. This means that there is a centralized source for developing and storing the current state of the production network. In Figure 2, the git repository is in the center. The left-most circle illustrates Continuous Integration (CI), where developers test, deploy and build a new version of the source code. The right-most circle shows the Continuous Deployment (CD), which is making sure the state specified in the source code reflects the state of the Kubernetes deployment. CD is also responsible for monitoring and operating the deployment. The CD part of GitOps is often managed by a tool, automatically, e.g. ArgoCD [14]. The CI part is a combination of developers and automated tools.

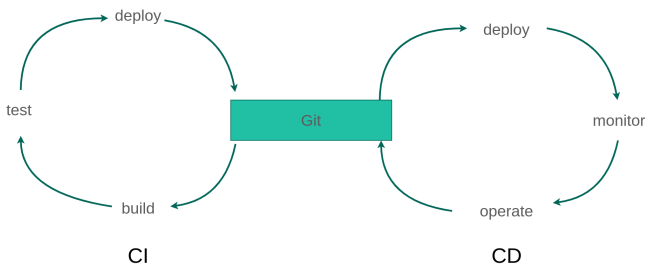

Fig. 2. GitOps Workflow

It also described how this source code can first be pushed to test nodes for testing. If the test succeeds, it can automatically be pushed to the production network. If it fails, the version will be rolled back to the last known working state. This way of working eliminates a lot of repeating tasks and enables the DevOps member to work more efficiently.

IV. Provisioning

A. Provisioning Configuration Files

Several Linux OS' have their own configuraiton file for provisioning their machines, e.g. Kickstart for Fedora [15], and Ignition for FCOS [16].

Both are plain-text files containing a set of instructions on how to provision a server. They are both run only once at the server's first boot.

They do however differ on how they do upgrades. Both support automatic updates, FCOS by configuring the Zincati service, and Fedora Server by using `dnf-automatic`. Fedora Server will be updated by updating packages individually. Similarly to git, Zincati prepares a new *commit* of the OS containing the updates. The server gets updated by rebooting with the latest commit.

Ignition files have functionality to create `systemd` services. This is also possible with `Kickstart`, but will require to create it in the *post* section [17] with more additional commands to do the same as `Ignition`. The `Ignition` file is in `JSON` format, which is quick for a computer to read. It is, however, not so human-readable. The solution is therefore to create the configuration as a `YAML`-formatted file. The Fedora community has made a tool called Butane [18] for converting the `YAML` file to the correct `JSON` format.

The Ignition file used downloaded and installed the binaries required to run the platform, that is argo-cd, sealed secrets, and k3s. The provisioning of the physical box required some manual steps, which was documented as a procedure.

Figure 3 shows the steps to provision the physical server. The steps will be explained in this section. Before executing the steps, the following prerequisites have to be in place first. The project's git reposiotry [19] needs to be accessible locally, which means cloning the `coreos` directory from the git repository to the administrator's computer. The software `podman` or `Docker` need to be installed. For writing to the USB-stick in step 2, the Rufus software needs to be installed [20] (or equivalent solutions). The web server hosting the Ignition file used to provision the server needs to be configured. The web server will run at the administrator's computer. This procedure uses the Python module called `http.server` [21]. This module uses by default the current directory of where the Python program is running and the network port 8000.

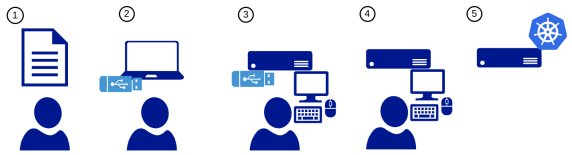

Fig. 3. Provisioning Procedure

The numbers in the list below correlate with the steps in Figure 3. Both uses the FCOS and Ignition as the example. The two first steps is done on the Administrator's computer, where the following steps is located on the physical server.

1) Modify the two configuration files - the *live-media* and the *provisioning* version
2) Embed the live-media configuration file to the USB-stick
3) Run the OS from memory by using the live-media mounted with an USB-stick
4) Install the OS by running the `coreos-installer`, and specify the URL to the ignition file as an argument
5) The server is ready to host a Kubernetes cluster

V. Orchestration

A. Traefik

Traefik is an Ingress Controller [22] that is part of the CNCF landscape [10]. Traefik administrates traffic by reading specific rules. These rules come in the form of a CRD, called `IngressRoute` [23]. The destination `Service` is required. There are multiple options to adjust the traffic, for instance redirecting to different `Service`'s based on the URL, or sending authentication requests to another Traefik specific resource, called `Middleware`.

The OSS version of Traefik has a limited number of authentication alternatives [23]. The enterprise version supports OAuth 2.0 authentication, but the pricing and the rest of the features are intended for large corporations. A better alternative is to use the `ForwardAuth` component in Traefik to redirect authentication requests to a third-party service. This service is Keycloak, and will be covered in detail in Section V-B.

It can also manage the authentication process to the applications located in the cluster. Not all authentication mechanisms are supported in the OSS version of Traefik Ingress Controller. It supports for instance Basic Authentication (`BasicAuth`), Digestive Authentication (`DigestAuth`), and Forwarded Authentication (`ForwardAuth`). Both `BasicAuth` and `DigestAuth` sends the credentials with vulnerable format. `ForwardAuth` redirects the authentication process to a separate service (Identity Provider). The Identity Provider can either be self-hosted or public. Example of a self-hosted Identity Providers is Keycloak [24]. Keycloak supports several popular standards, e.g. OAuth 2.0, OpenID Connect, and SAML 2.0, and its project is supported by Red Hat.

Examples of public Identity Providers are Google [25] and GitLab [26]. Google requires a Google Workspace subscription to be able to only accept members from a specific organization. The free version only supports given anyone with a Google account access. This defeats the purpose of using an Identity Provider. GitLab's solution requires no extra subscription, and supports given access to member of a given GitLab project.

B. Authentication with Keycloak

Keycloak is an OSS for managing identity and access to several supported applications and services, e.g. Kubernetes [24]. It has functionality to support modern identity and access protocols, e.g. OpenID Connect, SAML 2.0, OAuth 2.0. It is part of the CNCF Landscape [10], but not part of the Graduated and Incubating groups. The project is however sponsored by Red Hat [24].

As mentioned in Section V-A, Keycloak can be used in combination with Traefik Ingress Controller, by holding the role as the `ForwardAuth` in the `Middleware` resource. Traefik can then use Keycloak to outsource the authentication process.

C. Sealed Secrets

As pointed out by a CNCF blog post [27], secret management in an Infrastructure Code Repository is very important. Secrets are not stored encrypted by default, as described in this Section. This means that confidential information is stored in plain text when using the repository as a single source of truth. There are several OSS' for storing secrets securely, as mentioned in the CNCF blog post. The one that will be in focus in this project, is developed by Bitnami, called Sealed Secrets [28]. It is supported by ArgoCD [29], but it is not part of the CNCF Landscape [10]. As mentioned above, CNCF has recommended it in the blog post.

The `kubeseal` binary [28] was required to be installed on the physical server. This was done through the Ignition file explained in section IV-A (see Teppan [30] for the full provisioning script). Creating the sealed secrets required the administrator to be connected to the physical server [28]. The secret manifests were created as normal, before sending the file to the `kubeseal` command. The resulting file (the

SealedSecret) was committed to the Infrastructure Code repository.

D. MetalLB

For the workloads to be configured with a valid IP address so that external nodes can reach them, a network load-balancer is needed. For on-premise Kubernetes solution that support L2 load balancing, it was found only one OSS alternative that was part of the CNCF Landscape [10], called MetalLB [31]. It is still in alpha release. In the absence of better alternatives, MetalLB was chosen. It is important to be aware of the potential risk of running alpha releases in a production environment.

Due to the small amount of workloads needed in this project, only a few local IP addresses were reserved from the local subnet to be used by MetalLB. This is done by configuring the network router to reserve the IP addresses. In Listing 1, MetalLB is instructed to use the 10.0.100.0/27 IP range for allocating addresses to the services running on the cluster.

Listing 1. MetalLB Configuration

```
configInline:
  address-pools:
   - name: default
     protocol: layer2
     addresses:
     - 10.0.100.0/27
```

VI. Application Development

A. CD with ArgoCD

ArgoCD is an OSS that enables GitOps to Kubernetes [14]. It checks with the configured git repository for changes. If a change is found, ArgoCD will push this change to the configured Kubernetes cluster. It sorts out everything from storing secrets, such as container image registry username and password, updating container images after an update, and monitoring the namespace.

ArgoCD combined with the GitLab's CI/CD pipeline can deliver CI/CD for a Kubernetes deployment [32]. An alternative is to use Tekton [33] together with ArgoCD, which is what Red Hat's OpenShift utilizes [8].

1) Configuring Access to Repository for ArgoCD: To be able to automatically deploy the latest changes in the GitLab repository to the Kubernetes cluster, ArgoCD needs to be configured with a WebHook [34]. API-token was used for ArgoCD to authenticate to the GitLab repository. Furthermore, a specific namespace for this integration was made, called argocd. This namespace contains the API-token as a Kubernetes secret and a pod that runs the service for communicating with the remote repository.

2) The ArgoCD Application and Infrastructure Code Repository: Both the Kubernetes Infrastructure and ArgoCD Application Code is found in the same git repository. This implementation uses the *Apps of Apps* installation pattern, recommended by ArgoCD [35]. These files manifest how the Kubernetes installation should look like. There is also another git repository, that is separated from this, and that is the Application Source Code. They are split, since ArgoCD recommends separating the Infrastructure Code and the Application Source Code [36]. The following sections will explain in detail the ArgoCD Application Code and the Infrastructure Code.

B. Managing Kubernetes Infrastructure Code

By *Kubernetes Infrastructure Code* it is referred to the collection of code that is needed to run the Kubernetes PaaS. It is written declaratively, to support the GitOps methodology described in Section III-B1. In the following sections, the popular tools for managing this type of code repository will be introduced, that is Helm and Kustomize. ArgoCD also falls in this category, but has its own section VI-A dedicated to it.

1) Helm Charts: Creating Kubernetes manifests for every application would require a lot of work to create and maintain. That is where *Helm charts* come into picture. An Helm Chart is a collection of manifests for a given workload [37]. It is in many ways similar to a package manager. A package manager keeps track of the version history of all its packages and is a centralized hub for end-users to get packages. Helm serves a similar purpose, only with manifests, or Helm charts. Helm is a good alternative when using well-known applications, e.g. monitoring with Prometheus [38] and Grafana [39]. Using Helm charts means less administration for local administrators. Helm charts are maintained by a community of developers, and therefore the quality and update frequency of each chart is varying.

2) Kustomize: Kustomize is a service for configuring Kubernetes manifests [40]. It can therefore also customize and manage Helm charts [41]. This is useful when specializing the chart to fit the local environment. Examples of Kustomize configuration are a database password and a custom URL to a website.

C. GitLab CI

To build the container, a container tool is needed. GitLab has Runners, that can execute a CI pipeline [42]. In a container application development repository, this pipeline should build, test and push the container image. An executor is the environment that processes the stages [43]. The GitLab Runner supports several type of executors. For this usecase, the Docker executor will be used.

VII. Observability and Analysis

This section will describe observability related to servers, kubernetes infrastructure, and containerized applications.

In the observability category of *CNCF Landscape* [10], there are four projects that falls in the Graduated and Incubating groups, where Prometheus is the only Graduated. The Grafana project is not part of the qualified projects, but are however supported by the CNCF.

The log shipping had to be configured for the workloads to ship metrics to Prometheus. This was done in

the `kustomization.yaml` for each workload, under the `commonAnnotations` key. The configuration is illustrated in Listing 2.

Listing 2. Prometheus Log Shipping

```
commonAnnotations::
  prometheus.io/scrape: "true"
  prometheus.io/path: /metrics
  prometheus.io/port: "8080"
```

VIII. Platform

K3s is an open-source Kubernetes distribution created by Rancher [44]. Its main selling point is the lightweight binary, meaning it can be installed on almost any computer. It is built for running from a low resource computer to a multi-cluster production environment. The binary file for it is only 40 MB, and a node running it only requires 512 MB of RAM. This means that the master node can host pods. This is something that the other alternatives can not.

IX. Discussion

We note that 4 of the projects discussed in the paper are considered to be suitable for production, each belonging to their own category. Among the solutions considered in this paper the Provisioning category does not have a project sufficiently mature for production. FCOS with Ignition were selected as the provisioning tool due to the minimal setup on the physical server. For a larger environment, this solution is not recommended. This can be reflected in CNCF's supported provisioning tools which is aimed for a larger environment.

Runtime is not mentioned, since k3s provides a container runtime out of the box (containerd).

Among the projects discussed in this paper, considerable attention has been given to Kubernetes. While there are several graduated projects in the Orchestration & Management category, Kubernetes is the only one in the subcategory of Scheduling & Orchestration. Furthermore, it appears to be by far the most popular, with more than double the number of stars and eight times the number of commits compared to any of the other projects in this category.

The CNCF maturity of the different tools has been a factor in the selection process. In the selection of the provisioning tool, a non Cloud-Native tool were selected. Some recommended tools were considered, but it was argued that they would complicate such a small environment. Combining FCOS and k3s was successful without further complicating the setup. However, for further works to improve this setup, one or several of the recommended tools should be considered to replace FCOS. For instance, a solution could be to manage the physical server with Chef or Ansible.

X. Conclusion

We have presented a survey on Infrastructure-as-Code solutions for producing secure cloud software, which serves as a foundation for creating an IaC-based "lab-in-a-box" [30]. For our purposes, we conclude that GitLab and Kubernetes containers are most suitable.

Acknowledgments

This paper is based on the first author's MSc work [30] at the University of Stavanger, with support from the Norwegian Research Council through FME Cineldi, project no. 257626/E20.

References

[1] M. Ashley and B. Wolfe, "Scaling a migration to continuous delivery (airbnb)," 2019, spinnaker Summit 2019. [Online]. Available: https://devops.com/devops-chat-continuous-delivery-at-airbnb/

[2] S. Vöst and S. Wagner, "Towards continuous integration and continuous delivery in the automotive industry," *ArXiv*, vol. abs/1612.04139, 2016.

[3] J. Wang, J. Li, Y. Zhang, and Y. Bai, "Continuous integration and deployment for machine learning online serving models," *Uber Engineering*, 2021.

[4] Dawson, Brian, "Digital Disruptors, How AirBnB, Tesla, and Uber Used Software to Transform Entire Industries," https://www.cloudbees.com/blog/digital-disruptors-how-airbnb-tesla-and-uber-used-software-innovation-transform, accessed: 2022-05-11.

[5] Puppet, "State of DevOps Report 2021," https://puppet.com/resources/report/2021-state-of-devops-report, Puppet Labs, Tech. Rep., 2021, accessed: 2022-05-11.

[6] Amazon, "AWS Pricing Calculator," https://calculator.aws/#/createCalculator/EKS, accessed: 2022-05-21.

[7] F. Beetz, A. Kammer, and S. Harrer, *GitOps, Cloud-native Continuous Deployment*, 1st ed. Berlin, Germany: InnoQ Deutschland GmbH, 2021.

[8] W. Pernath, *Getting GitOps, A practical platform with OpenShift, Argo CD, and Tekton*. Raleigh, United States of America: Red Hat Developer, 2022.

[9] The Cloud Native Computing Foundation, "Graduated and Incubating Projects," https://www.cncf.io/projects/, accessed: 2022-06-05.

[10] ——, "Cloud Native Landscape," https://landscape.cncf.io/, accessed: 2022-06-05.

[11] J. Humble and D. G. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ: Addison-Wesley, 2010. [Online]. Available: http://my.safaribooksonline.com/9780321601919

[12] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*, 1st ed. IT Revolution Press, 2018.

[13] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[14] The ArgoCD Authors, "What is Argo CD?" https://argo-cd.readthedocs.io/en/stable/, accessed: 2022-06-14.

[15] The Fedora Project, *Automating the Installation with Kickstart*, Fedora Community, 2021, https://docs.fedoraproject.org/en-US/fedora/latest/install-guide/advanced/Kickstart_Installations/.

[16] ——, *Ignition*, Red Hat Inc, 2021, https://coreos.github.io/ignition/.

[17] Red Hat, "A.3 Scripts in Kickstart File," https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/performing_an_advanced_rhel_installation/kickstart-script-file-format-reference_installing-rhel-as-an-experienced-user, accessed: 2022-06-14.

[18] The Fedora Project, "Getting Started," https://coreos.github.io/butane/getting-started/, accessed: 2022-06-14.

[19] H. Teppan, "Infrastructure as code for smart grid security lab," https://gitlab.com/iac-thesis-uis/sintef-lab, 2022.

[20] Pete Batard, "Rufus," https://rufus.ie/en/, accessed: 2022-06-12.

[21] The Python Authors, "HTTP Servers," https://docs.python.org/3/library/http.server.html, accessed: 2022-06-12.

[22] The Traefik Authors, "Welcome," https://doc.traefik.io/traefik/, accessed: 2022-06-05.

[23] ——, "Traefik & Kubernetes," https://doc.traefik.io/traefik/routing/providers/kubernetes-crd/, accessed: 2022-06-05.

[24] The Keycloak Authors, "Open Source Identity and Access Management," https://www.keycloak.org/, accessed: 2022-06-05.

[25] The Google Authors, "Setting up your OAuth consent screen," https://support.google.com/cloud/answer/10311615, accessed: 2022-06-14.

[26] The GitLab Authors, "Configure GitLab as an OAuth 2.0 authentication identity provider," https://docs.gitlab.com/ee/integration/oauth_provider.html, accessed: 2022-06-14.

[27] S. Kok, “Secrets management: essential when using Kubernetes,” https://www.cncf.io/blog/2022/01/25/secrets-management-essential-when-using-kubernetes/, accessed: 2022-06-14.
[28] Bitnami, “”Sealed Secrets” for Kubernetes,” https://github.com/bitnami-labs/sealed-secrets, accessed: 2022-06-14.
[29] The ArgoCD Authors, “Secret Management,” https://argo-cd.readthedocs.io/en/stable/operator-manual/secret-management/, accessed: 2022-06-14.
[30] H. Teppan, “Utilize GitOps for Smart Grid Security Lab,” Master’s thesis, University of Stavanger, 2022, accepted: 2022-07-20T15:51:30Z Publisher: UiS. [Online]. Available: https://uis.brage.unit.no/uis-xmlui/handle/11250/3007299
[31] The MetalLB Contributors, “MetalLB,” https://metallb.universe.tf/, accessed: 2022-06-14.
[32] The CNCF, “CICD Pipelines using Gitlab CI & Argo CD with Anthos Config Management,” https://www.cncf.io/blog/2021/01/27/cicd-pipelines-using-gitlab-ci-argo-cd-with-anthos-config-management/, accessed: 2022-06-14.
[33] The Tekton Authors, “Cloud Native CI/CD,” https://tekton.dev/, accessed: 2022-06-14.
[34] The ArgoCD Authors, “Git Webhook Configuration,” https://argo-cd.readthedocs.io/en/stable/operator-manual/webhook/, accessed: 2022-06-14.
[35] The ArgoCD Developers, “Cluster bootstrapping,” https://github.com/argoproj/argo-cd/blob/master/docs/operator-manual/cluster-bootstrapping.md, 2022.
[36] ArgoCD, *Best Practices*, ArgoCD, https://argo-cd.readthedocs.io/en/stable/user-guide/best_practices/.
[37] The Helm Authors, “The Package Manager for Kubernetes,” https://helm.sh/, accessed: 2022-06-14.
[38] The Prometheus Authors, “What is Prometheus?” https://prometheus.io/docs/introduction/overview/, accessed: 2022-06-05.
[39] The Cloud Native Computing Foundation, “Graduated and Incubating Projects,” https://www.cncf.io/projects/, accessed: 2022-06-05.
[40] The Kustomize Authors, “Kubernetes Native Configuration Management,” https://kustomize.io/, accessed: 2022-06-14.
[41] The Kubernetes Authors, “Kustomization of a Helm Chart,” https://github.com/kubernetes-sigs/kustomize/blob/master/examples/chart.md, accessed: 2022-06-14.
[42] GitLab, *GitLab Runner*, The GitLab Developers, https://docs.gitlab.com/runner/.
[43] ——, *Executors*, The GitLab Developers, https://docs.gitlab.com/runner/executors/.
[44] The Rancher Community, *K3s - Lightweight Kubernetes*, Rancher, https://rancher.com/docs/k3s/latest/en/.