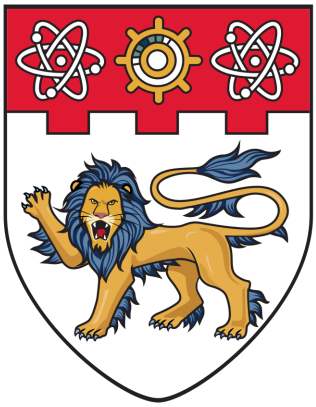

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Final Year Project

SCSE21-0566

GitOps in Kubernetes Clusters

Poh Kai Kiat

Supervisor: Associate Professor Chng Eng Siong

Examiner: Professor Wen Yonggang

2022

Submitted in Partial Fulfilment of the Requirements for the Degree of Bachelor of Computer Science of the Nanyang Technological University

Abstract

This project aims to create an end-to-end pipeline combining Git best practices, Continuous Integration and Continuous Deployment (CI/CD) and apply them to infrastructure automation, provisioning and monitoring. This is often known as GitOps.

GitOps incorporate the whole Git ecosystem such as pull requests and code reviews into infrastructure automation. By adopting GitOps, organizations can release and rollback features frequently and with ease.

The solution can be divided into 2 parts - the Continuous Integration pipeline and the Continuous Delivery pipeline. The CI pipeline mainly focuses on the usage of GitHub actions to automate the building and testing of the application code. On the hand, the CD pipeline focuses on ArgoCD and evaluates the rollout strategies that can be used.

This report will present the architecture diagram and the steps to implement the solution.

Acknowledgements

I would like to express my sincere gratitude to several individuals for supporting me throughout this project. The project would not have been possible without them.

Firstly, I want to thank Professor Chng Eng Siong for his guidance and advice on this project especially during the early stages of the project when I was uncertain about project requirements. He was always willing to listen and advice on the project. I also want to thank Professor Chng for the opportunity to work on this project.

Next, I am also grateful to my mentor Research Engineer Vu Thi Ly for her constant guidance and support for the project despite her busy schedule. She helped ensure that my progress for the project was smooth through the regular meetings.

Lastly, I would like to thank my friends and family for the support they have provided throughout the entire project

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Automatic speech recognition (ASR) is a technology that translates spoken languages into text. The ASR is based on an open source software called Kaldi that provides a speech recognition system using finite-state transducers [1]. A dedicated team of researchers from NTU Speechlab has developed speech recognition models for the ASR system that can transcribe speech containing a mixture of languages like English and Chinese which is useful in Singapore's bilingual context [2]. This system can be used in call centers for transcribing subtitles for videos as well as for real-time transcription.

1.2 Objectives and Aims

The goal of this project was to improve the continuous integration and continuous delivery (CI/CD) pipeline of the system using modern-day best practices like GitOps. This project aimed to create an end-to-end pipeline unifying best practices from Git deployment, monitoring and management

of containerized clusters and applications. This will ensure a better experience for developers working on operations and application development.

The project focused greatly on CI/CD, which involves building automation in the building, testing and deployment of an application to ensure that the application can be delivered to customers promptly. As releasing software can potentially be a painstaking process that might involve manual integration, changing configuration files as well as integration testing, a good CI/CD pipeline enables the team to release more features without compromising on quality and additional intervention.

Furthermore, this project was implemented based on the principles of GitOps, which is an operational framework that takes best practices used for software development, such as version control, collaboration, compliance, and CI/CD tooling, and apply them to infrastructure automation [3]. In the past, many organizations have been plagued with bottlenecks when it came to handling cloud infrastructure. GitOps can help to simplify the deployment process by relying on a single source of truth, which is the GitHub code repository, to define the infrastructure running the application.

1.3 Scope

The scope of the project was to implement a CI/CD pipeline using GitOps principles. The project utilized Google Cloud Platform (GCP) as the cloud service provider.

The solution was divided into 2 sections, mainly the CI pipeline as well as the CD pipeline. The CI pipeline was implemented using GitHub Actions. Meanwhile, the CD pipeline was implemented using Argo CD and Argo Rollouts. They are both open-source tools that can deliver infrastructure updates to Kubernetes clusters in GCP and provide enhanced deployment

capabilities such as blue-green and canary deployment. Furthermore, the solution also involved the usage of monitoring tools like Prometheus and Grafana to analyze the results of the deployment for the application.

The solution also utilizes Helm to install the Kubernetes application as well as Terraform, an open source tool created by HashiCorp which enables the provision of infrastructure as code.

1.4 Report Organisation

There are 5 chapters in this report, each chapter explaining different areas as explained below

Chapter 1: Introduction and overview of the project

Chapter 2: Summary of technologies used

Chapter 3: Analysis and design of proposed solution

Chapter 4: Detailed implementation of the solution

Chapter 5: Conclusion and suggestions for possible improvements

Chapter 2

Literature Review

This project was deployed using containerization technology with Google Cloud Platform as the main cloud provider. The main continuous integration and continuous delivery (CI/CD) tool that was used in this project are GitHub Actions [4] and ArgoCD [5]. Furthermore, the project utilized tools like Terraform [6] and Helm [7] which are infrastructure as code software tools to manage and provision infrastructures reliably. Lastly, to gain insights into our CI/CD pipelines, we use monitoring tools such as Prometheus [8] and Grafana [9].

2.1 Containerization

Containerization is the packaging of software code with its dependencies such as its binaries, libraries, configurations and framework into an isolated "container" [10]. The packaged container is abstracted away from the host operating system. Hence, the containerized application can run in any environment or infrastructure.

Traditionally, before containerization, virtualization was widely used by most organizations. It is a technique where developers can run multiple

virtual machines (VM) on a single server's CPU [10]. Each virtual machine has its operating system, on top of the virtualized hardware hence there will be overhead and poor performance.

Figure 2.1 depicts the architectural differences between containers and virtual machines [11].

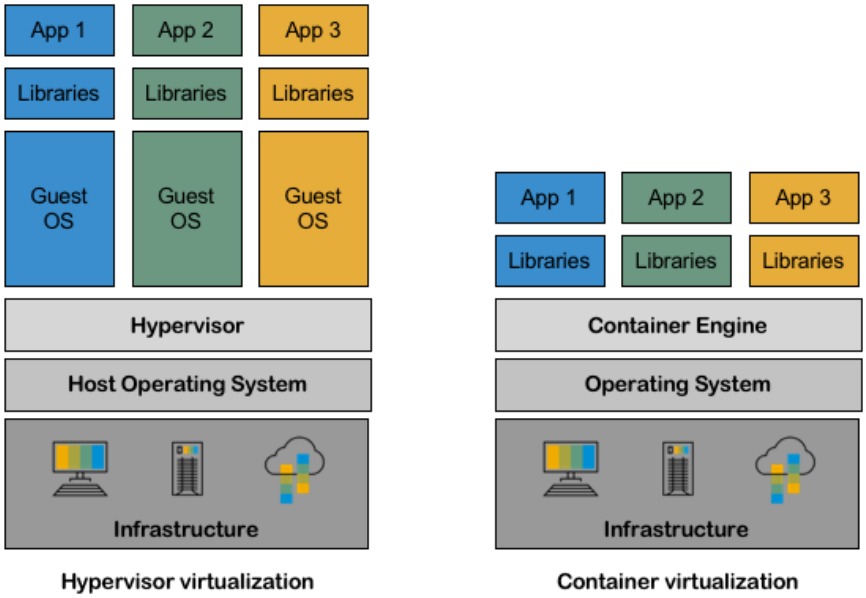

Figure 2.1: Architecture diagrams of Container and Virtual Machines

Containerization technology has become increasingly popular among developers as it provides benefits such as [12]:

1. **Faster delivery:** Containerization allows developers to divide their applications into discrete parts. This ensures that code changes and upgrades can be performed independently from other containers.

2. **Improved security:** The isolation nature of containerization means that applications are running in their own self-contained environment. This means that if one application is compromised, other applications will still be secure.

3. **Portability:** Developers do not have to rebuild their containerized

application if they were to change the environment or underlying infrastructure. This is because containers are independent of the host operating system.

4. **Lightweight:** As seen from figure 2.1, containers do not have hypervisors, which is a software that enables multiple Guest OS to share the underlying system's resources, hence this makes containers lightweight and reduces their startup time.

2.1.1 Docker

Docker is an open-source tool for developing, shipping and running applications. It streamlines the development lifecycle by allowing developers to work in a standardized environment [13]. Docker applications are environment agnostic hence they can be run on any host machine. Docker manages the container provisioning and provides its own registry to allow developers to store and version Docker applications.

Furthermore, Docker facilitates Agile software development, which is getting increasingly popular in organizations [14]. Agile is an incremental and iterative approach to software and project management. Instead of releasing all software features in a single go, Agile aims to break requirements into smaller tasks. Agile allows the organization to release new features every other sprint cycle which is typically 2 - 3 working weeks.

By using Docker, developers can adjust to the ever-changing requirements of an application as they can modify the Dockerfile easily. The Dockerfile is used to build an image, which is a set of instructions that Docker containers can use to execute code. Each Docker image has a SHA256 code and can be versioned, this also facilitates the rolling back of features.

2.1.2 Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads. [15]. There are several features of Kubernetes that makes it popular among developers, namely

1. **Automated rollouts and rollbacks:** Kubernetes updates the application with zero downtime by incrementally replacing older pods with newer ones. Kubernetes also offers a mechanism to revert and rollback any changes that break the application.

2. **Self-healing:** Kubernetes will redeploy any components to their desired state when it goes down. Self-healing allows applications to be available 24/7 and greater availability across the system. Kubernetes does this by constantly probing its resources to check if they are running in the desired state.

3. **Horizontal scaling:** Kubernetes automatically upgrades a resource by allocating more CPU resources to match demand. Likewise, it can scale down a resource when there is a lack of demand. This ensures that a Kubernetes cluster can effectively handle a rise in traffic as well as reduce resource wastage when there is no demand.

4. **Cloud Agnostic:** Kubernetes can run on various platform such as Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), or even on-premises. This ensures that there is no vendor lock-in.

As the solution focused more on rollouts and rollbacks, the following paragraph will explain more about it. The default rollout strategy used in Kubernetes is rolling deployment. Below is a manifest file that defines a pod with a rolling update strategy:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rollout-example-app
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2 #default value is 1
      maxUnavailable: 0 #default value is 1
```

When using the rolling update strategy, 2 parameters can be defined:

1. **maxSurge:** Specifies the number of pods that can be created above the desired amount of pods during an update
2. **maxUnavailable:** Specifies the maximum number of pods unavailable during the rollout

In an actual deployment manifest, at least one of the parameters mentioned above must be greater than 1. During a rollout, Kubernetes scale down pods in the older version and create pods for the newer version according to the values defined in `maxSurge` and `maxUnavailable`.

When Kubernetes is deployed, there will be a cluster. Figure 2.2 shows the components of a Kubernetes cluster [11].

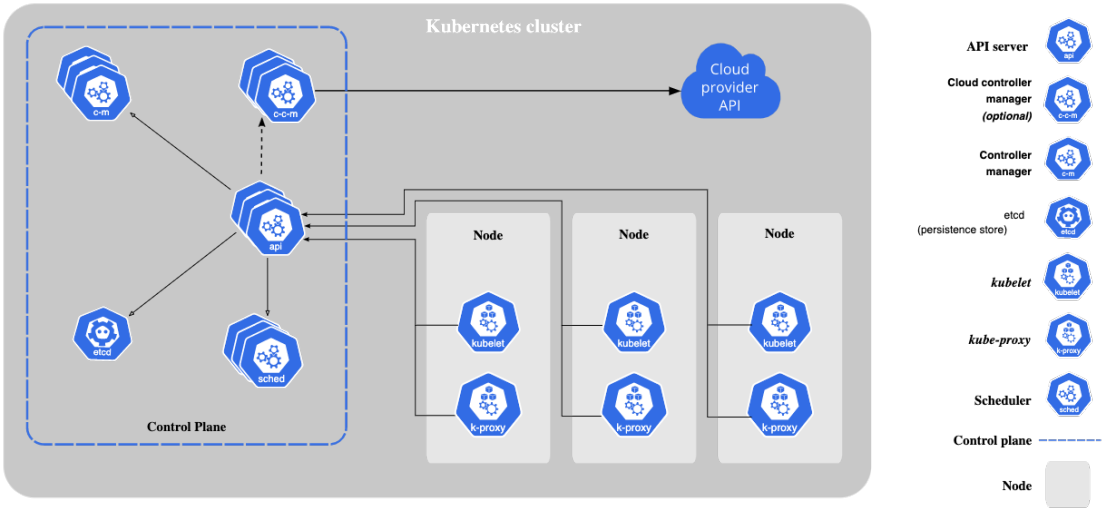

Figure 2.2: Components of a Kubernetes Cluster

The following few points will briefly describe the core concepts in Kubernetes.

1. **Pod:** Pods are the smallest deployable units of computing that Kubernetes can create and manage. A pod is a group of one or more containers comprising of shared network and storage resource [16]. Each pod has its Internet Protocol (IP) address, when it dies, a new IP is assigned to the Pod.

2. **Node:** A node can be a physical machine or a virtual machine depending on the cluster [17]. Kubernetes group multiple pods to form a node. A node contains the essential services to run a pod.

3. **Control Plane:** The control plane make top level decisions about the cluster [18]. It manages the Kubernetes cluster and workloads running on them and it includes components such as the kube-apiserver, etcd and kube-scheduler.

4. **Deployment:** A Deployment can be used to update and create Pods

and ReplicaSets [19].In short, a deployment manages a pod. In the deployment manifests file, developers can define the exact specifications of a Pod, such as its environment variables, which ports to expose or what volume to mount to the Pod.

5. **Service:** Kubernetes gives pods their Internet Protocol (IP) addresses and Domain Name System (DNS) name [20]. It provides an abstract way to load balance network requests across pods. Whenever a pod dies, it gets assigned a new IP address, however, the Service is not affected as Kubernetes identifies its pods via the selector label.

6. **Secrets:** Kubernetes secrets is sensitive data that is accessed by Kubernetes resources. It includes credentials, API tokens or a key [21]. Kubernetes secrets are stored in the cluster's in-built data store, etcd.

7. **Persistent Volume:** Persistent Volume (PV) refers to a storage in the cluster that the administrator has provisioned. The PV follows the lifecycle of the cluster, it is stateful in nature. It provides a storage/ file system for Kubernetes Pods to access to.

8. **Persistent Volume Claim:** It is a request to provision the PV with a certain configuration and type.

2.2 GitOps

GitOps is an operational framework that combines DevOps best practices used for application development such as version control, collaboration, and CI/CD and applies them to infrastructure automation [3].

GitOps combine several practices. They are explained in the following few bulleted points.

1. **Infrastructure as Code (IaC):** IaC is defined as managing and pro-

visioning infrastructure through code instead of manual process [22]. IaC ensures that it is easier to edit and share configuration files within the development team. Also, configuration files are declarative in nature instead of being imperative. The desired state of the infrastructure provisioned by IaC tools should correspond to the one stored in the Git repository. Furthermore, as most IaC tools use version control, it ensures that any changes made to the configurations are trackable.

2. **Git best practices:** Part of Git best practices includes creating a pull request (PR) to make changes to a repository. A pull request notifies other developers working on the same repository to review and approve the changes before merging into the main branch.

3. **Continuous Integration/ Continuous Delivery (CI/CD):** GitOps integrates a CI/CD pipeline into the git repository. Whenever there is a change in the repository, an automatic CI/CD pipeline will be triggered to update the existing infrastructure. A CI pipeline helps to create a standard flow to test, package and build applications whenever a developer commits his code [23]. Meanwhile, the CD pipeline automates code deployment to their respective environments, such as production, staging and development.

The implemented solution consists of several IaC tools such as Argo CD, Argo Rollouts, Terraform and Helm.

2.2.1 Argo CD

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes [5]. Argo CD uses a git repository as a source of truth for the desired state of the Kubernetes environment. The manifests can be defined as YAML

files, ksonnet/jsonnet applications or Helm packages. Argo CD repeatedly compares the state of the application and the desired state specified in the git repository, a sync operation will be performed whenver there is a difference.

Figure 2.3 shows a high level overview of the architecture of Argo CD [24].

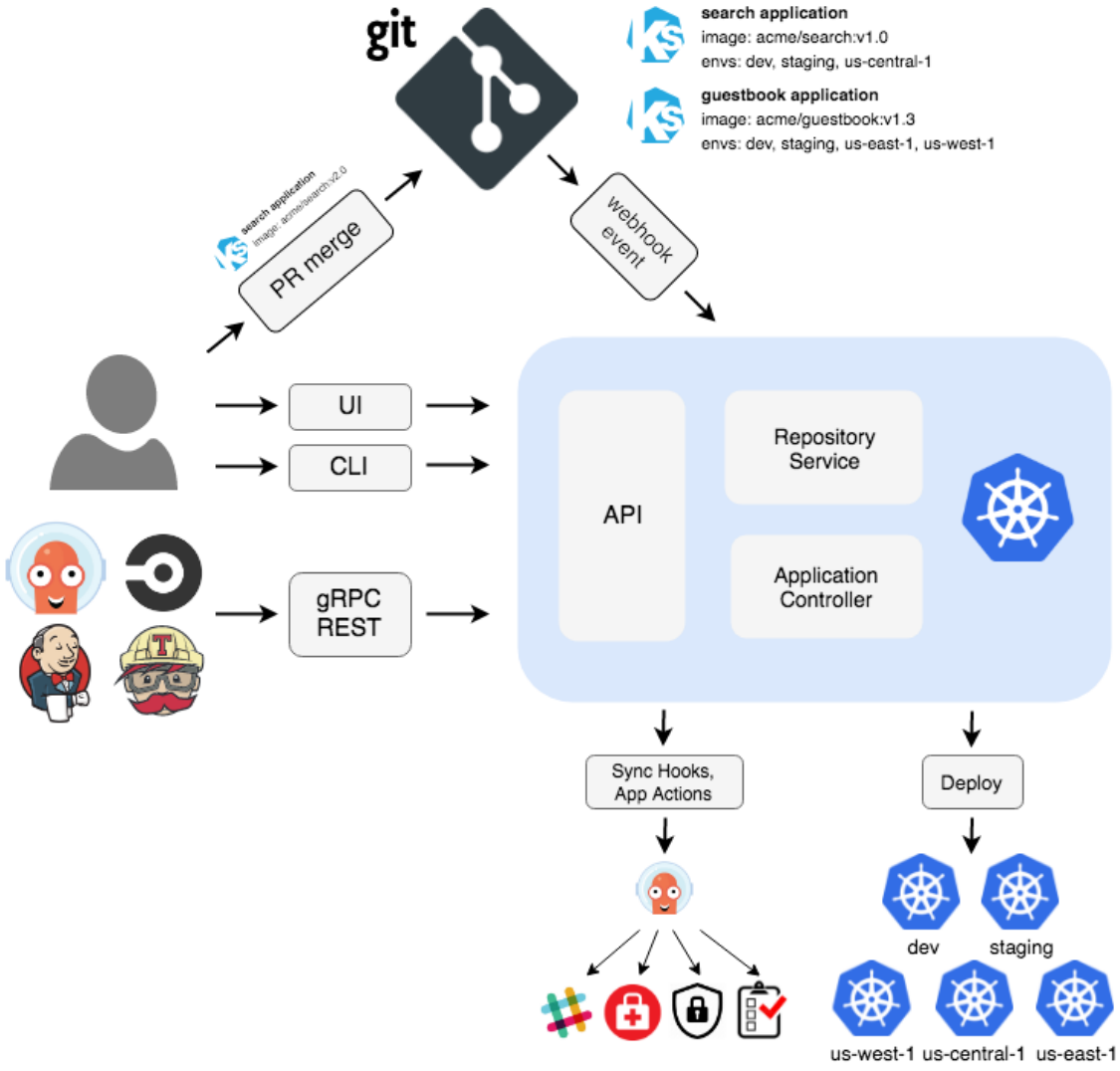

Figure 2.3: Argo CD Architecture

As seen from figure 2.3, there are 3 main components, specifically:

1. **API server:** A HTTP/gRPC server that exposes the endpoints to the web client. It serves several purposes including managing the

application and credentials for the cluster and repository as well as acting as the listener/forwarder for webhook events by GitHub.

2. **Repository Server:** The repository server serves as a cache for the git repository container the Kubernetes manifests. It generates the manifests on demand.

3. **Application Controller:** The application controller's role is to monitor the existing state of the application and the desired state of the application as defined by the manifests files in GitHub. Whenever the application is out of sync, restorative actions will be taken.

There are many features in Argo CD which make it a suitable tool for this project [25], namely:

1. Support for a variety templating tools such as Kustomize and Helm.

2. Web user interface for real-time update and visualization of application activity.

3. Support for multicluster which means that operators can use a single Argo CD instance for production and non-production environments.

4. Ability to define own role-based access control (RBAC) policies to enable restriction of access to Argo CD resources, this allows multiple teams to access Argo CD but with different level of scopes and permissions.

5. Support for alerting tools such as Slack, Teams, Emails and Telegram. An alert can be sent to users whenever the application becomes out of sync. Also, the alerting template is highly customizable.

ArgoCD uses 2 kind of deployment approaches - **Pull based** deployment & **Push based** deployment.

Figure 2.4 shows a comparison between a pull and push based deployment [26].

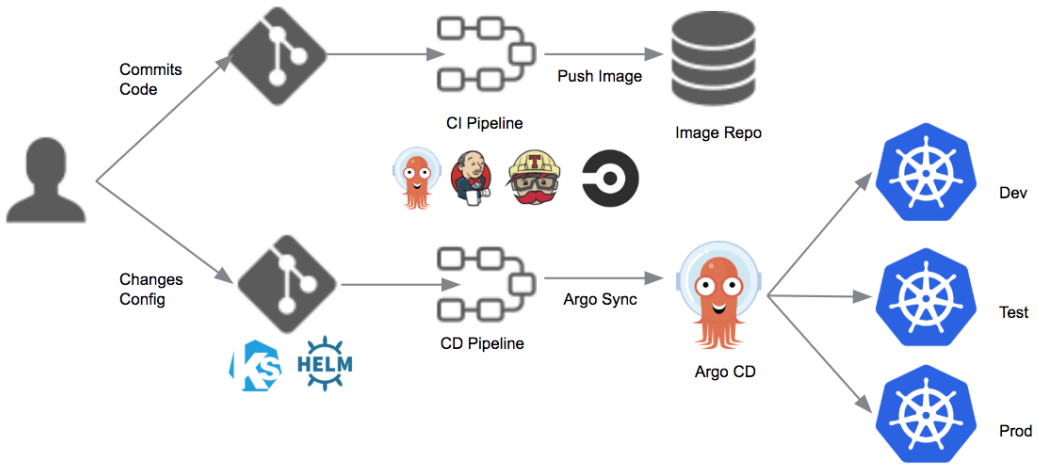

Figure 2.4: Pull vs Push deployment approach

For the pull based deployment, ArgoCD watches for a mismatch between the desired and current state of the infrastructure. Whereas, in the push based model, the ArgoCD pipeline is triggered whenever there is a merge or code commit.

There are some advantages in a Pull based deployment, for example, there is enhanced security as developers do not have to expose the credentials in the GitHub repository or CI pipeline [27]. Developers can create service accounts or configure role-based access (RBAC) configurations in the Kubernetes cluster itself. Also, by separating the CI and CD pipelines, there is less coupling between the 2 pipelines, giving each pipeline a single responsibility to perform.

On the other hand, there are also benefits in a Push based deployment [28]. A Push based deployment is more optimized than a pull based deployment since there is no need for a Kubernetes resource to continually poll for the state of the desired and current of the infrastructure. If there are multiple Kubernetes clusters, the Kubernetes agent have to polling and checking

the state of the infrastructure. This will result in an overhead of network requests which translate to more costs and delays. All in all, developers have to consider the pros and cons of a pull and push based deployment and choose the style which suits the organization.

2.2.2 Argo Rollouts

Argo Rollouts is a Kubernetes controller and set of custom resource definitions (CRDs) that provide advanced deployment capabilities such as blue-green and canary features to Kubernetes [29]. By default, Kubernetes provide `RollingUpdate` capabilities during an update. However, there are certain restraints such as the lack of control of how fast the rollout is carried out, the inability to manage traffic flow and lastly metrics cannot be obtained during a rollout. Argo Rollouts provides a user interface and a command line interface to manage rollouts that allow developers to perform operations such as promoting, restarting and aborting a rollout.

Figure 2.5 shows a high level overview of the architecture of Argo Rollouts [30].

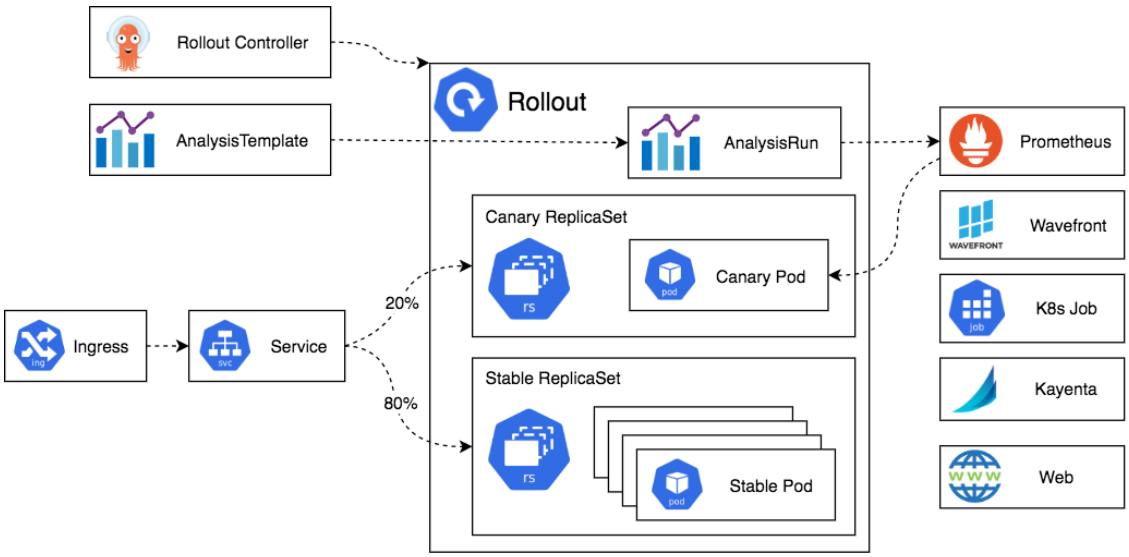

Figure 2.5: Argo Rollouts Architecture

Argo Rollouts comprises of 3 main components - Rollouts resource, Ingress/ Service, ReplicaSets for old and new versions.

1. **Rollouts resource:** The rollouts resource is a Kubernetes resource managed by Argo Rollouts, it provides the functionality to manage and control deployments. It can be used to alter rollouts such as stopping it, increasing the number of Pods and adding certain limits.

2. **Ingress/ Service:** It manages how external traffic is directed into the Kubernetes cluster. Argo Rollouts uses the Kubernetes Service resource, with some additional add-ons, which allows traffic to be directed to the older or newer version of the application.

3. **ReplicaSets:** Argo Rollouts maintain a stable set of Pods for different deployment versions.

Argo Rollouts uses 2 kind of rollout strategy - canary & blue green rollout.

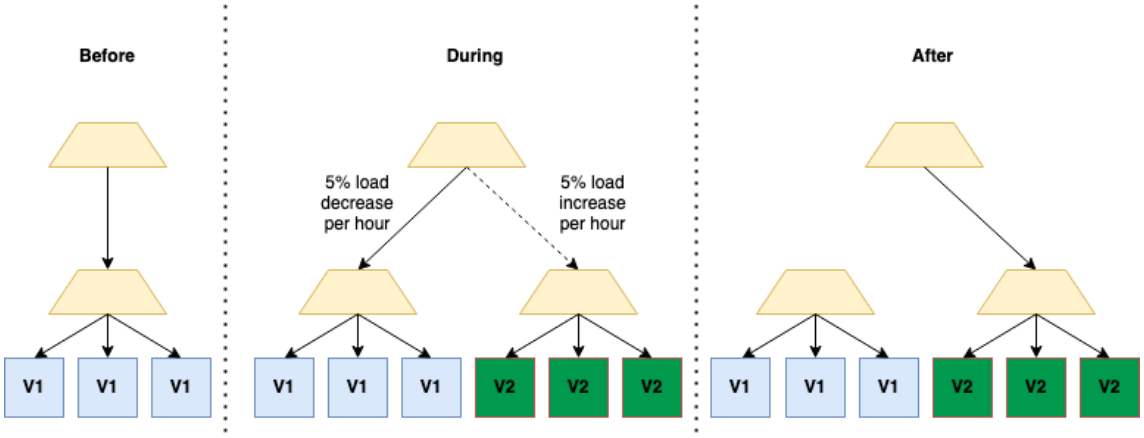

Figure 2.6: Canary Rollout

As seen from figure 2.7 canary rollout is a release strategy where a new version of an application is released to the production traffic in small percentage [31]. Canary release is a powerful technique as it allows developers to increase the amount of traffic to the new application after it passes functional and non-functional tests or perform an rollback if the release is substandard. A canary release results in gradual changes in the production environment, this gives developers more leeway to experiment with new features.

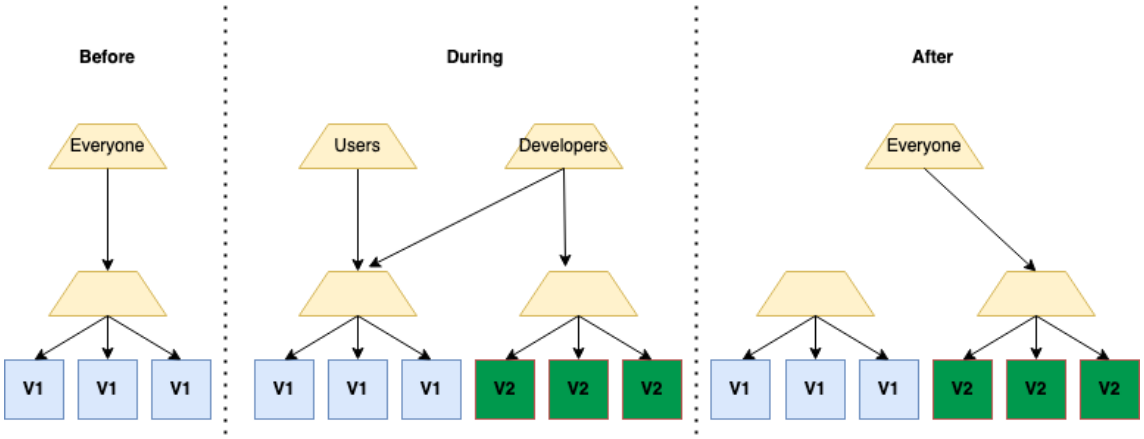

Figure 2.7: Blue Green Rollout

On the other hand, as seen from figure 2.7, a blue green deployment allows more than 1 version of the application to be running at the same time [32]. The original version of the application is often referred as blue environ-

ment whereas the preview version is called the green environment. During this time, developers can test the preview environment before rolling it out as the blue environment (original version). Blue Green deployment results in zero downtime and gives developers ample time to conduct testing without affecting the end users.

Argo CD checks the health of Argo Rollouts via Argo CD's Lua health check to determines the state of the Argo Rollouts [33]. In addition, Argo CD provides a service to alter the state of Argo Rollouts, for example, to pause a rollout. As a result, we can combine Argo Rollouts and Argo CD to implement an end-to-end pipeline which follows GitOps best practices.

2.2.3 Argo Analysis

ArgoCD analysis allows developers to execute Prometheus queries while performing a canary/blue-green rollout. This feature is not available if we use the default rolling update strategy. Other than Prometheus, developers can also integrate ArgoCD with DataDog, NewRelic, Wavefront etc. By analyzing our rollouts, we collect information such as the percentage of failed and successful requests. This can help to determine if we should proceed with the deployment or abort it.

2.2.4 Argo CD Image Updater

Argo CD Image Uploader is a plugin provided by Argo CD which scans for latest container images and automatically updates the Kubernetes Cluster [34] by updating the deployment repository with the new image number. The image uploader will use an additional file called `.argocd-source` to keep track of the image number. This tool is useful if the team wants to use a pull based workflow.

2.2.5 Terraform

Terraform is an open-source tool developed by HashiCorp. It allows developers to define cloud and on-prem resources in a human-readable configuration file that can be version, reuse and share across teams [6]. Using Terraform, we can provision and make changes to the deployed infrastructure, for instance, adding a new compute instance. One reason why Terraform is popular among developers is that it supports a large number of resources such as Amazon Web Services (AWS), Google Cloud Platform (GCP) and Kubernetes [35].

The figure 2.8 below shows how Terraform works [36].

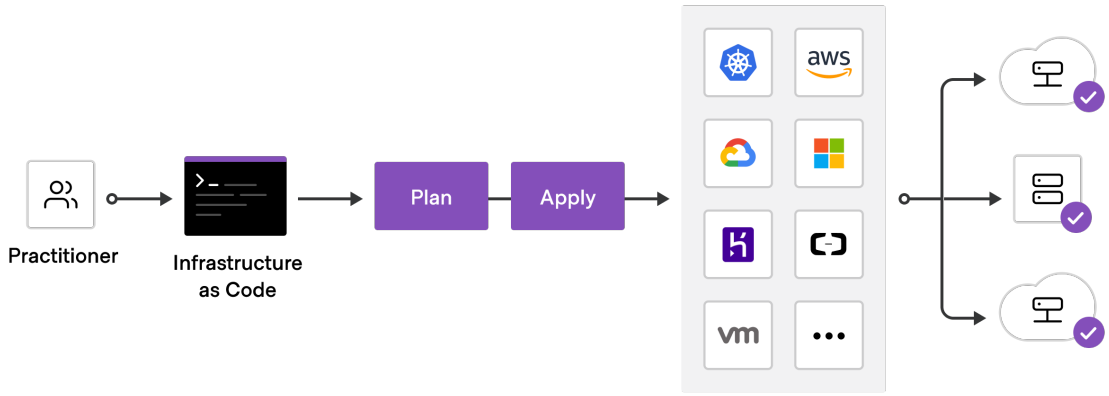

Figure 2.8: Terraform Workflow

Developers define the state of the desired architecture in a file called the state file. Terraform then takes in the input and determines what are the steps to be taken by comparing it with the existing architecture.

Terraform allows developers to provision resources within a few minutes, which greatly reduces the need for traditional click-ops deployment methods that are error-prone and take a long time. Also, Terraform will come in handy when the application needs to be deployed in a multi-cloud environment such as in Azure, Google Cloud Platform and so on. Terraform can save developers a lot of time as configuration files can be reused.

2.2.6 Helm

Helm is a package manager for Kubernetes [7]. Helm aims to simplify how developers manage Kubernetes manifests files by using Helm charts. A chart is a bundle of files that describe a set of Kubernetes resources [37]. Helm keeps a history of deployed charts which allows developers to rollback any changes. There is also a public repository where Helm charts can be shared and stored.

A Helm chart can be customized for deploying to different environment

in Kubernetes as such developers do not need additional manifests files to deploy an application to a separate environment. Futhermore, Helm allows developers to add variables and functions inside the template files which works well for scalable applications with ever-changing features and requirements.

There are many alternatives to Helm such as Kustomize and jsonnet. Unlike Helm, Kustomize is more of an overlay engine rather than a templating engine, it is less powerful as developers do not have access to complicated operations such as conditionals, range and functions. Furthermore, Helm is an established tool as there is already a sheer number of Helm charts readily available online.

2.2.7 GitHub Actions

GitHub action is a feature available on GitHub which integrates a continuous integration (CI) pipeline into the git repository. It allows developers to create their own workflows to test their pull requests or to build their applications [4].

GitHub Actions has their own market place, where developers can find common pipelines created by others or even share their pipelines. Futhermore, GitHub Actions is easy to setup as it is integrated into all GitHub repositories. There is also no need to host or maintain any CI servers to run the pipeline as the runners are maintained by GitHub.

However, the downside of using GitHub Actions is that the repository is tied to a single source code management system. As a rather new product, there are also certain usage limits when using runners maintained by GitHub, namely: [38]

1. **Job execution time:** Jobs exceeding 6 hours will be terminated au-

tomatically.

2. **Concurrent jobs:** The maximum number of concurrent jobs that can be run on a free and enterprise account is 5 and 50 respectively.
3. **API limits:** There is a maximum of 1000 API requests that can be invoked per repository per hour.

2.3 Monitoring

The implemented solution uses monitoring tools like Prometheus and Grafana. Monitoring is the practice of understanding how software components run in a remote environment [39]. Monitoring important as it allows developers to optimize and debug existing programs. A key advantage of monitoring is that developers can predict future system-level changes with collected data.

2.3.1 Prometheus

Prometheus is an open-source monitoring and alerting tool. It collects and aggregate metrics as time series data. Metrics is a numeric measurement of what the user wants to measure over a time range [8]. Prometheus uses a pull-based mechanism to retrieve the metrics and store them as time series data. The data collected can be visualized and analyzed to improve the existing application.

Figure 2.9 shows a high level overview of the architecture of Prometheus [40].

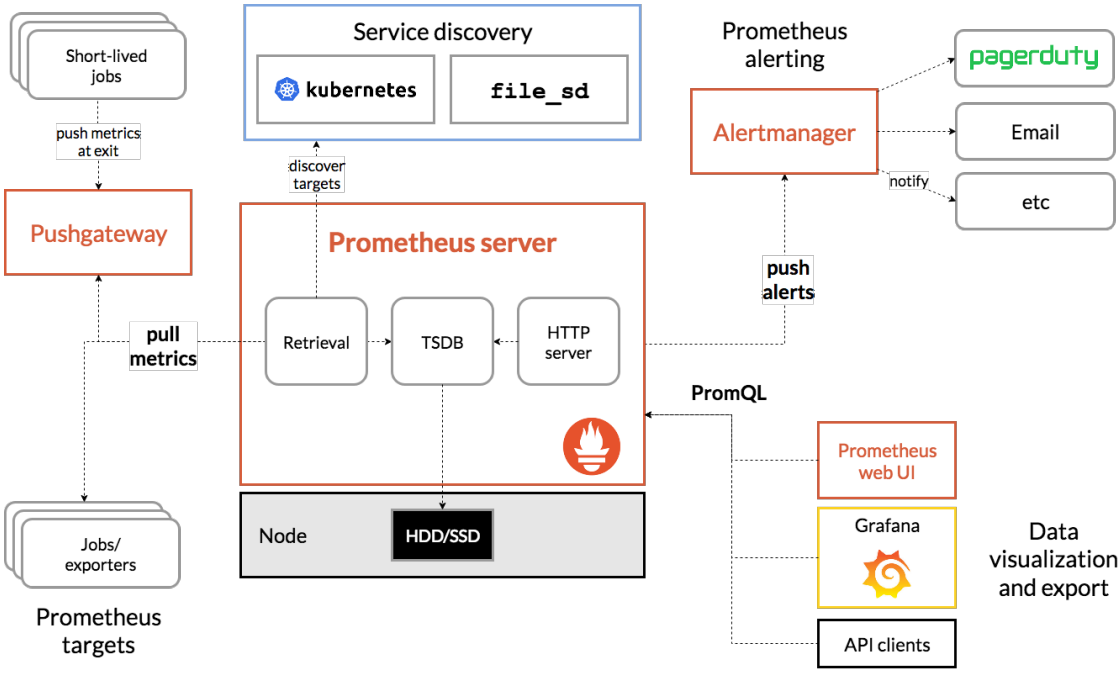

Figure 2.9: Prometheus Architecture

Prometheus comprises of 3 main components - Prometheus Server, Service discovery, Prometheus Alerting

1. **Prometheus Server:** The Prometheus server is responsible for pulling metrics from the targets or via the Pushgateway for short-lived jobs. It is also in charge of storing the metrics as a time series data and executing PromQL query, which is a query language written for Prometheus.

2. **Prometheus Exporter:** Exporters are third-party tools that help scrap metrics for Prometheus when it is not possible to extract metrics from the target application. There are different kinds of exporters that export different data such as CPU usage, network usage, etc.

3. **Service discovery:** Prometheus service discovery is a technique of

finding endpoints to scrape for metric [41].

4. **Prometheus alerting::** The Prometheus alerting handles the alert sent by the Prometheus Server. It groups the signals and routes them to the correct receiver such as email, PageDuty, etc.

2.3.2 Grafana

Grafana is an open-source tool for visualization and analytics. Grafana is used to visualize time-series data by using various dashboards which help developers study and understand their applications better. Grafana can be used in conjunction with different data sources such as Prometheus as there is already in-build support in both Grafana and Prometheus.

2.4 Cloud Computing

Cloud Computing provides a pay-as-you-go pricing policy, instead of buying and owning physical servers and data centers, organizations can utilized remote data centers owned by cloud providers. There are many advantages for organizations to switching to cloud computing, namely :

1. **Costs:** Cloud computing eliminates the cost to set up a physical data center, hardware and other hidden costs. Relying on a pay-as-you-go pricing policy from cloud providers, organizations have the control to expand or shrink their business according to their demands, resulting in less wastage of resources.

2. **Performance:** As cloud providers have data centers available across the globe, this results in less latency in network requests for the applications. As a business grows, cloud providers can scale to meet the demand.

3. **Disaster Recovery:** Cloud Providers maintain backup for data in their data centers in the event of cyber attacks, natural disasters or power outages. This ensures a certain level of reliability for businesses and allows businesses to minimize loss during an unfortunate incident.

2.4.1 Google Cloud Platform (GCP)

Google Cloud Platform is a variety of services offered by Google. This includes Google Compute, Google Networking and Google Storage.

2.4.2 Google Kubernetes Engine (GKE)

Google Kubernetes Engine (GKE) allows developers to manage, deploy and scale their containerized applications with ease [42]. A GKE cluster comprises of several components such as nodes, control plane and services provided by Google Cloud Platform (i.e VPC networking, Cloud monitoring, Load Balancer).

2.4.3 Google Cloud Storage (GCS)

Google Cloud Storage provides worldwide storage and reliable access to the data stored [43]. GCS primarily stores binary large objects (blob), which includes .mp4, .pdf, or images. The purpose of GCS in this project is to store the state file for Terraform.

2.4.4 Google Filestore

Filestore are Network File System (NFS) file servers managed by Google Cloud [44]. They can be mounted on virtual machines (VM) instances or

the Google Kubernetes Cluster Engine. The rationale of the filestore is to store the models for the worker pods to use.

Chapter 3

Analysis and Design Approach

This chapter aims to briefly explain the benefits of the proposed solution and provide a brief description of the system architecture.

3.1 Traditional approach

Before examining the benefits of the proposed solution, we need to the traditional approach to DevOps in Kubernetes application.

A common CI/CD pipeline is to

1. A developer commits to the Git Repository.
2. The CI pipeline runs automated tests, the pipeline fails when there are errors.
3. The CI pipeline builds the application and pushes it to an image registry.
4. An automated script or manual steps to update the Kubernetes De-

ployment file and apply the changes to the Kubernetes Cluster.

In this approach, the CI is the core component of the pipeline. Besides that, Step 4 is error-prone and not scalable especially if the application is huge. The script has to be modified whenever a new feature is added to the application. The figure below (Figure 3.1) shows a traditional pipeline.

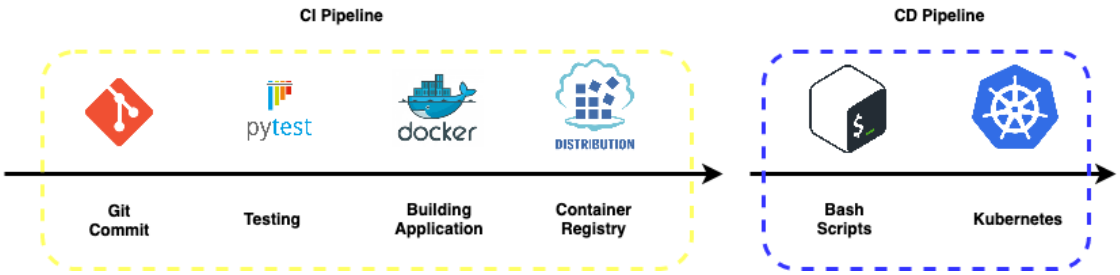

Figure 3.1: A tradition CI/CD pipeline

3.2 Benefits of proposed solution

The proposed solution aims to create an end-to-end CI/CD which solves the issues mentioned above. It relies on GitOps as a core principle and uses tools such as GitHub Actions and ArgoCD to accomplish it. Other than that, to make the solution more robust, we have added alerting and monitoring tools as well as incorporated different rollout strategies.

3.3 Benefits of GitOps

The solution uses Git as the central component. The configuration files are stored in Git. A simplified workflow can be

1. A developer commits to the Git Repository.
2. The CI pipeline runs automated tests and format the source code, the pipeline fails when any there are errors.

3. The CI pipeline builds the application and pushes it to an image registry.
4. The GitOps agent detects a change in the image registry and updates YAML files in the deployment repository.
5. An automatic workflow is triggered which will update the Kubernetes cluster with the relevant changes.

The figure below (Figure 3.2) shows a proposed GitOps pipeline.

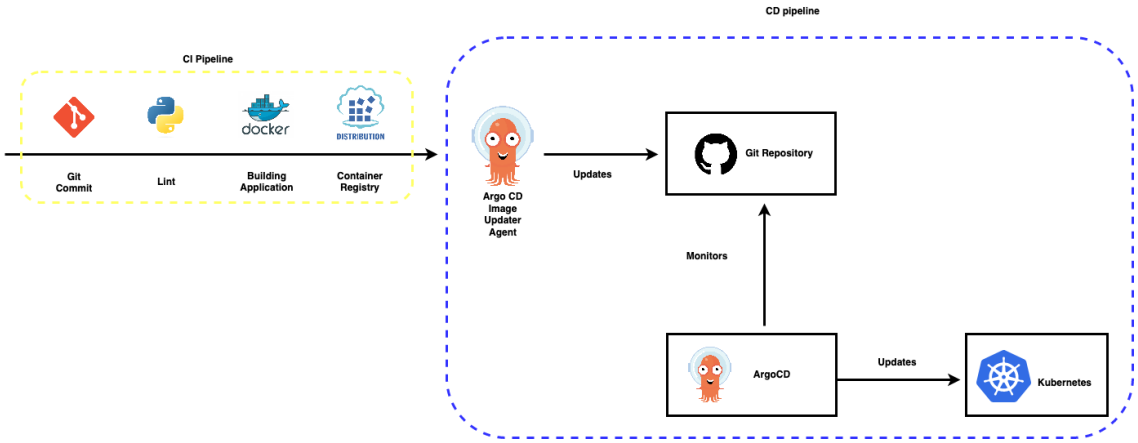

Figure 3.2: Proposed GitOps pipeline

There are numerous benefits to adopting a GitOps approach, for instance

3.3.1 Git as a single source of truth

As the YAML files for Kubernetes are stored in Git and there is a GitOps agent which automatically syncs the desired and actual state of the application, the YAML files in Git will always describe the actual state of the application. Using Git also ensures that the configuration files are versioned and there is a trackable history which helps in debugging.

3.3.2 Better developer experience

As most engineers are proficient in Git, the learning curve to manage the deployment workflow is less steep. For instance, developers can run `git log` to view the commits made to the repository, alternatively, they can use `git revert` to rollback certain changes. When developers are familiar with the tools they used, it will usually lead to higher productivity.

3.3.3 Git best practices

By using GitOps, developers can include practices such as merge requests. A merge request takes place when local changes are applied to the main branch and it usually requires a team member to review the code changes. This ensures better code quality and also ensures that the team knows the changes.

3.4 Benefits of GitHub Actions

We are using GitHub Actions for our CI pipeline. The CI pipeline is responsible for testing and building the application. GitHub Action can be easily integrated as we are storing the code in GitHub and it is easier to navigate around the GitHub user interface (Refer to Figure 3.3). The CI pipeline is defined as a YAMl file and is usually stored in the `.workflows` directory.

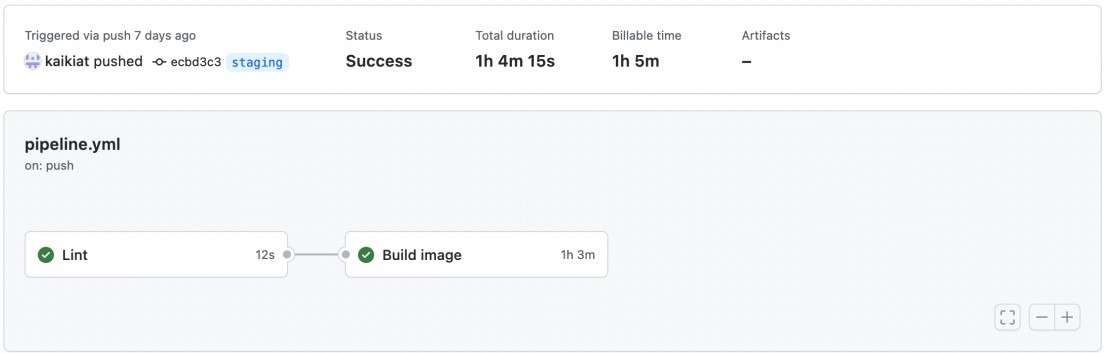

Figure 3.3: An example of GitHub Action UI

There are other alternatives in the market such as GitLab CI and Jenkins. GitHub Actions and GitLab CI are similar but they are hosted on different platforms. On the other hand, in the case of Jenkins, this tool might not be a suitable choice for the project as Jenkins instances are self-hosted. It

requires more niche knowledge to maintain Jenkins instances and it has an outdated user interface. The table below shows some of the key differences between the popular CI tools.

A comparison between CI tools			
	GitHub Actions	Gitlab CI	Jenkins
Price	$7 per user/mnth	$19 per user/mnth	Pay as you use
Maintenance	Easy	Easy	Hard
Self hosting	No	No	Yes

Figure 3.4: A comparison between CI tools

3.5 Benefits of ArgoCD

3.5.1 Providing automation in managing Kubernetes resources

Before the adoption of ArgoCD, it was relatively manual to make changes to an existing Kubernetes cluster. For instance, to increase the number of replicas in a cluster, a developer has to edit the configuration files and apply the changes using Kubernetes's graphic user interface (GUI) or command line tool (CLI). By using ArgoCD, which has an inbuilt self-healing mechanism, it will monitor the desired and the current state of the cluster and automatically update the cluster when the configuration files in Git change.

3.5.2 Improved developer experience

ArgoCD has a dashboard that provides an overview of all tracked applications and health. The state and health of all child deployed and managed by ArgoCD can be monitored as well. Having an easily accessible overview of all the deployed services allows developers to quickly debug and fix issues during outages. Figure 3.5 shows an example of the user interface. At

a glance, it shows the Kubernetes components required to create the application, such as the Kubernetes services, secrets, deployment and secrets etc. The are other functions in the website, for instance, a user can sync and rollback a deployment.

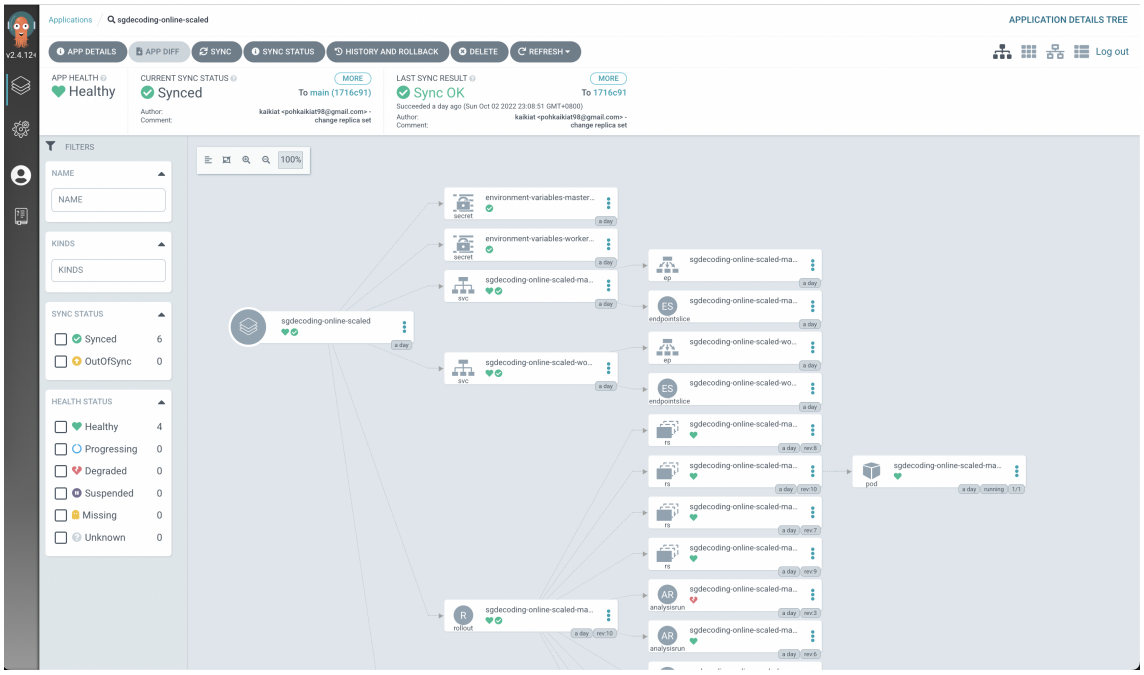

Figure 3.5: User interface of Argo CD

The user interface also allows users to view the manifest of a Kubernetes resource (Refer to Figure 3.6). This is useful for developers to inspect the application deeper without having to look at the Git repository.

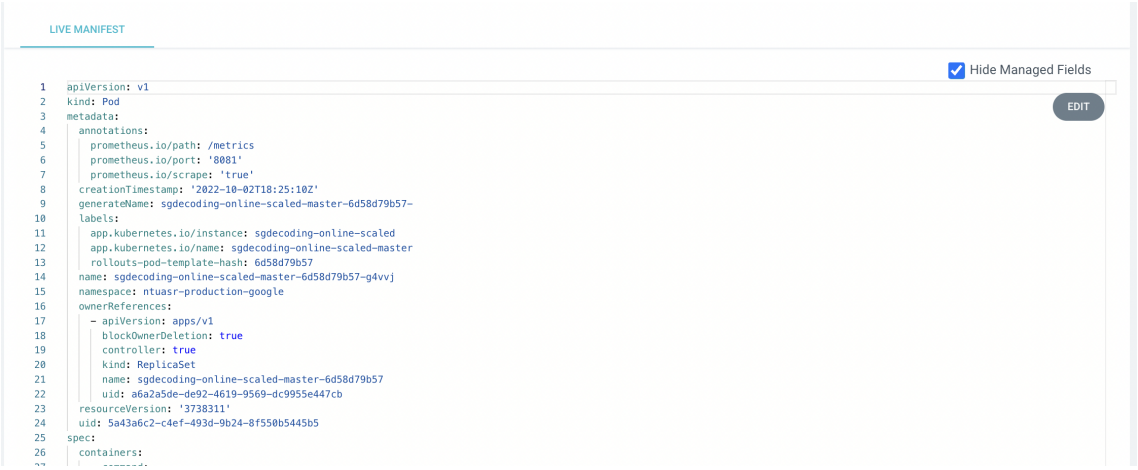

Figure 3.6: A live manifests of a Pod

3.5.3 Clean disaster recovery strategy

As the configuration files are stored decoratively in GitHub and GitHub represents the source of truth, whenever a disaster struck, the entire Kubernetes Cluster can be recreated quickly.

3.5.4 Better separation of CI and CD

In the GitOps pattern, Argo CD is completely responsible for deployments. This frees up the GitHub Action pipeline, whereby it will be solely responsible for building and testing applications.

3.5.5 High Availability

Most of the components in ArgoCD is largely stateless. Developers can increase the number of replicas of controllers depending on their needs.

Furthermore, ArgoCD supports leader election which helps it achieve high availability.

3.5.6 ArgoCD vs FluxCD

Another tool that we can use FluxCD which is another open-source tool. There are some differences between these 2 tools. Firstly, developers can only sync one Git repository to one FluxCD instance [45], but in ArgoCD, you can have multiple Git repositories. Next, FluxCD does not have a robust web user interface (UI) ecosystem. The existing project at `https://github.com/fluxcd/webui` is no longer maintained by the team, except of a project which is maintained by a third party called Weaveworks. On the other hand, FluxCD does have some features that ArgoCD does not, for instance, FluxCD can scan the image registry for changes to redeploy the cluster again. This feature is still under active development in Argo CD as of v.0.12.0.

Below are some of the key differences between the 2 technologies

A comparison between K8s Deployment tool		
	ArgoCD	Flux CD
Multi Git repositories	Yes	No
Good UI	Good	Average
GitHub Stars	10.6k	6.9k

Figure 3.7: A comparison between K8s Deployment tool

3.6 Improved Rollout Strategies

By default, Kubernetes only offer the `rolling update strategy`, which updates pod one by one and ensures zero downtime. However, there are a few limitations with this strategy.

1. Unable to control the duration of the rollout.

2. Inability to control the traffic flow of requests to the new application.
3. Cannot abort or rollback the deployment.
4. Cannot query external metrics such as Prometheus query to verify and test a deployment.

Hence, to make the solution even more-well rounded, we relied on Argo Rollouts to implement strategies such as canary and blue green deployment and also used tools like Argo Analysis to examine a deployment.

3.6.1 Benefits of Argo Rollouts

As we are using ArgoCD, it is easy to integrate Argo Rollouts into our existing solution. It provides a user interface (Refer to 3.8) for engineers to see the overview of the rollout.

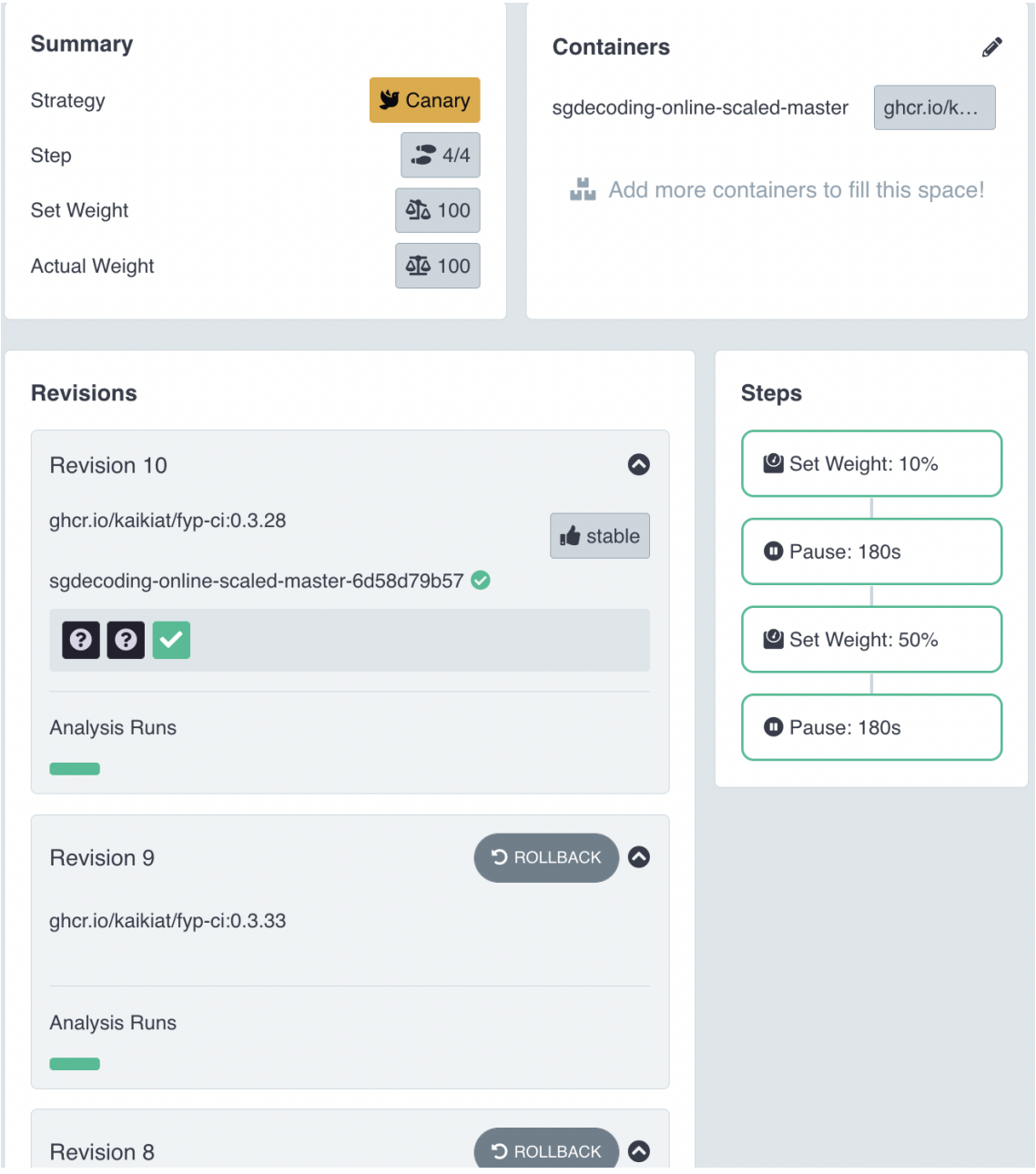

Figure 3.8: User Interface of Argo Rollouts

Engineers can view which stage the rollout is at and rollback to any version when needed.

3.6.2 Benefits of Canary Deployment

Canary Rollout is a deployment strategy where a new version of the application is released to the production environment incrementally. Canary Rollouts result in zero downtime and reduce the risk of a large number of users being impacted negatively due to a rollout. In this project, canary rollout is a good strategy since the ASR application might be deployed in call centers where there must be minimal downtime when upgrading the application.

3.6.3 Benefits of Blue Green Deployment

Blue Green deployment is another rollout strategy that can be used. This strategy allows 2 versions of the application to be running at the same time. Typically only the preview (green) version is visible to developers which allows them to run integration testing before switching traffic and releasing it to users. This strategy can be beneficial when it is required to have a fully working version running in production, for instance, when the ASR application is used in the live transcription of important events such as National Day Parade etc.

3.6.4 Benefits of Argo Analysis

In this project, the following Prometheus query will be used

$$\frac{\text{sum(number of request reject total)}}{\text{sum(number of request receive by master total)}} <= 0.10$$

This Analysis run checks that the number of failed requests received by the master service is less than 10 percent. Values higher than that may indicate that something is wrong with the new application and the rollout will be

halted.

3.7 Benefits of using Grafana Prometheus

In our proposed solution, Prometheus will be collecting data from different components such as ArgoCD, Argo Rollouts and the NTU ASR application. The data collected can be related to the amount of resources used, up time of service and network traffic

In this project, Prometheus will be used to collect metrics from 3 different services namely:

1. **ArgoCD** - Metrics related to the status of ArgoCD.
2. **ArgoRollouts** - Metrics related to the status of the rollouts as well as the resource used to manage the rollout.
3. **ASR Application** - Metrics related to the network traffic and the amount of resource used.

The metrics collected can be collected and displayed on Grafana. For instance, Figure 3.9 shows a dashboard for ArgoCD metrics whereas Figure 3.10 shows a dashboard for Argo Rollouts metrics. These information can be analyzed during outages, for instance, when users experience throttling of services, it could be due to the higher CPU usage of the worker pod in the ASR application.

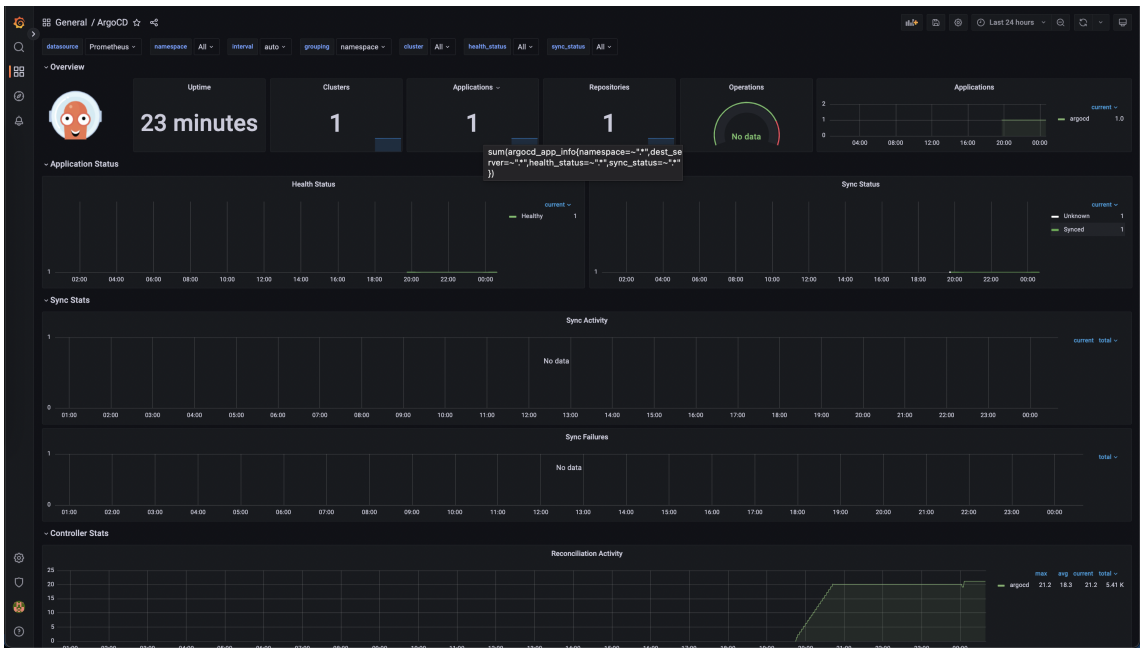

Figure 3.9: Argo CD Dashboard

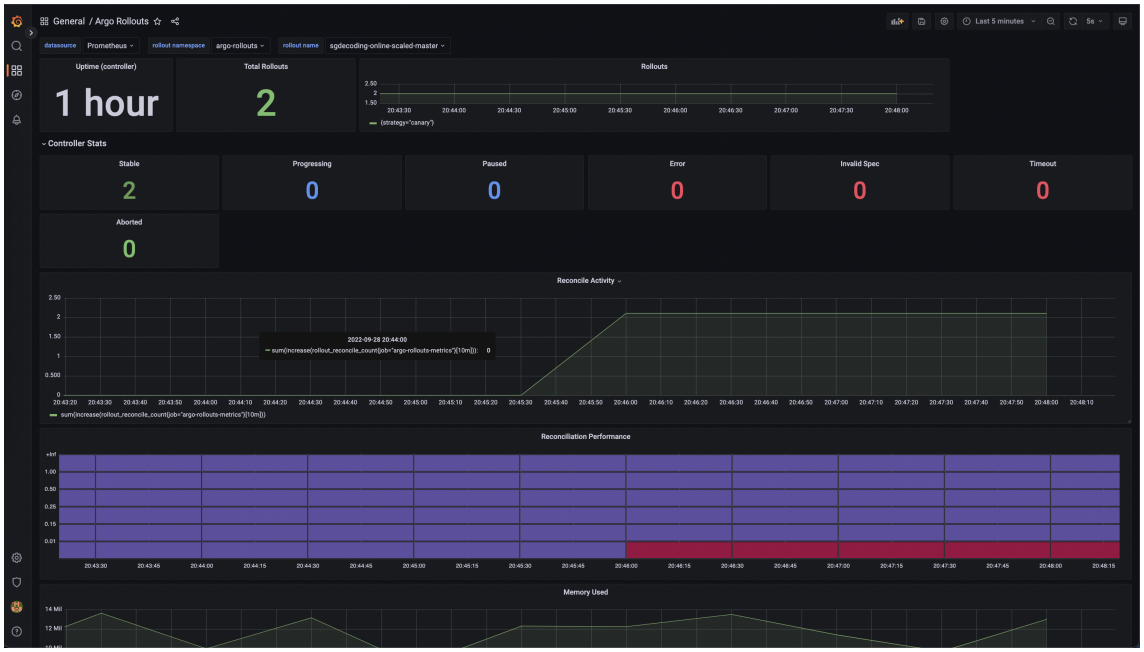

Figure 3.10: Argo Rollout Dashbboard

3.8 Proposed Workflow

This chapter will describe the steps of the solution. Furthermore, it will discuss briefly on the system architecture of the proposed .

3.8.1 Continuous Integration Workflow

Figure 3.11 shows a high-level overview of the continuous integration pipeline. The CI pipeline is triggered whenever a developer commits a change.

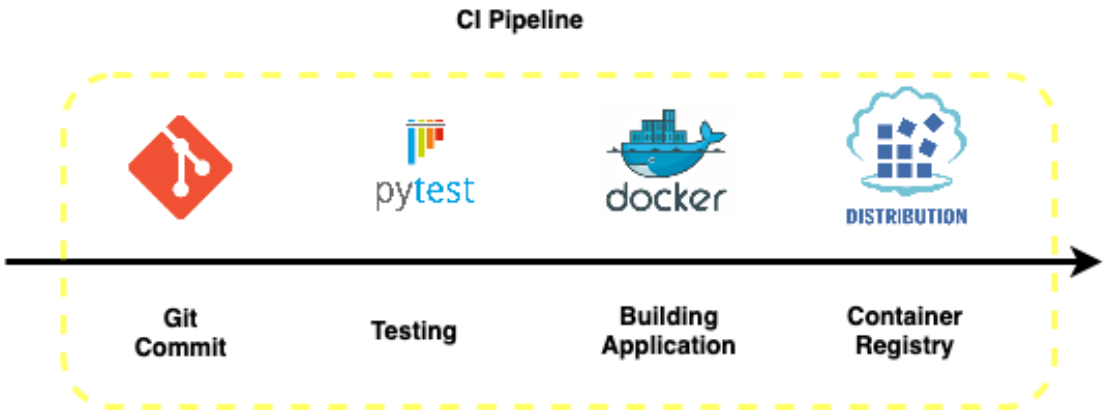

Figure 3.11: Continuous Integration Pipeline

The CI pipeline comprises of different stages. Each stage must be completed before the next stage can proceed. Also, the pipeline can be restarted or paused if needed. Each stage will be responsible for the following actions.

1. **Lint:** Analyse the code using Pylint and check for errors and code smell.
2. **Build:** Builds the ASR image and stores in the GitHub Container Registry.
3. **Push to GitHub Container Registry (ghcr):** After tagging the

ASR image with a new version number, the image will be uploaded to GHCR. In this project, semantic versioning (semver) strategy is used.

4. **Argo CD Image Updater:** The proposed solution uses an open source tool called Argo Image Updater. It will monitor GitHub registry for newer images [34]. After that, it will create a pull request at the deployment repository with the new version number

3.8.2 Continuous Delivery Workflow

The CD pipeline is a continuation of the CI pipeline. The main purpose of the CD pipeline is to rollout the new Kubernetes configurations according to the manifest file. Figure 3.12 shows a high-level overview of the continuous delivery pipeline.

The CD pipeline can be divided into 3 stages, namely

1. **Stage 1 - Monitoring:** In this stage, ArgoCD will monitor for any differences between the existing and desired state of the Kubernetes Cluster defined in the GitHub repository.

2. **Stage 2 - Deployment:** In this stage, Argo Rollouts will deploy the changes to the existing Kubernetes Cluster. This can be accomplished either by using a canary rollout or blue-green rollout. In figure 3.12, canary rollout is used. ArgoCD can perform analysis to gain metrics and insights from a rollout which can be carried out during or after a rollout. The metrics can determine whether a rollout should be aborted.

3. **Stage 3 - Alerting:** After the new application is deployed, ArgoCD will send notifications to the users. In the proposed solution, ArgoCD is integrated with Slack and Gmail.

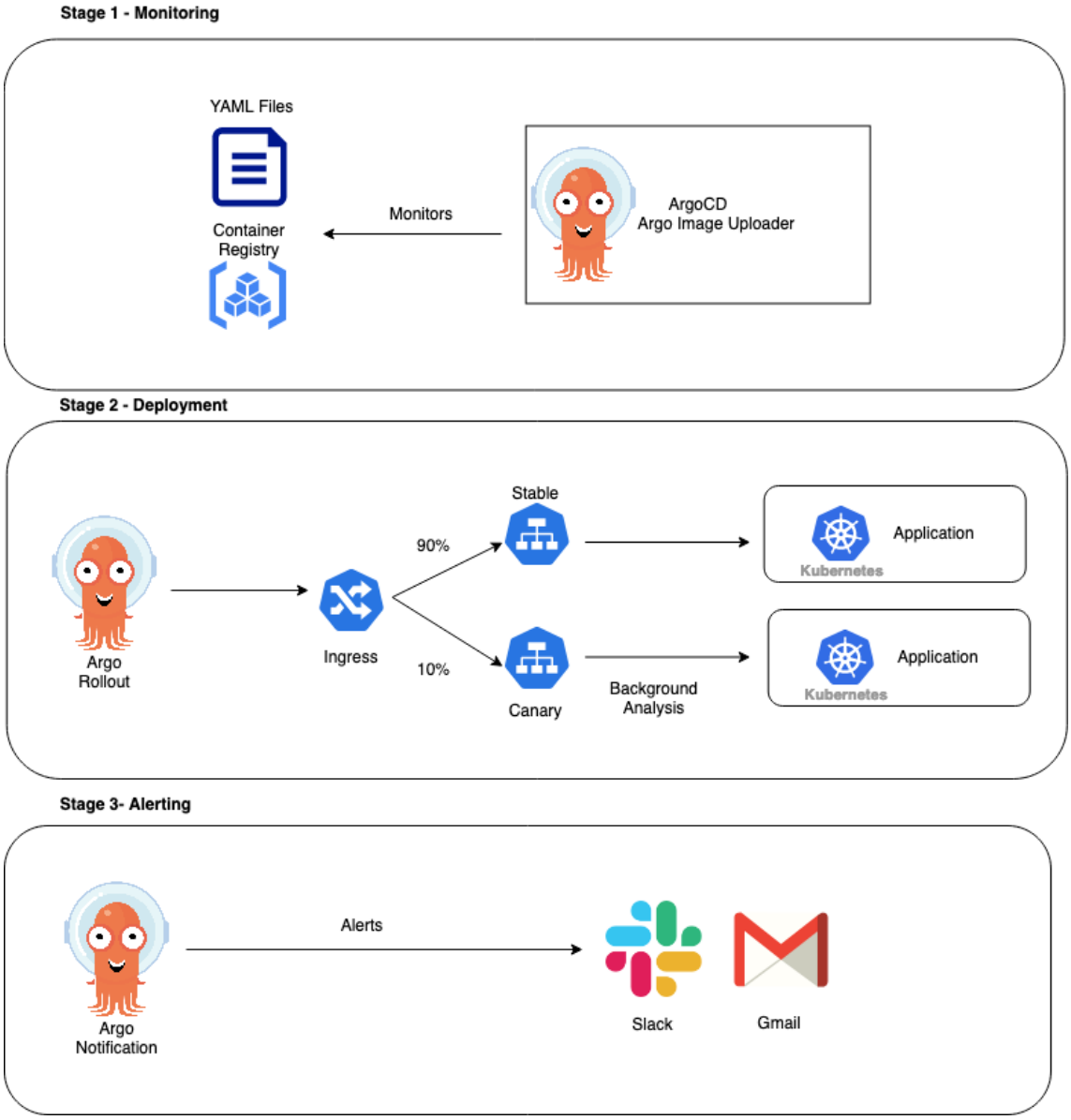

Figure 3.12: Continuous Delivery Pipeline

3.8.3 Monitoring Workflow

In the proposed solution, there are tools to monitor the application. Different kinds of data can be collected. Developer such as resource utilization, application's status and network traffic etc. The purpose of the monitoring solution is to provide more insights during canary/blue-green deployment.

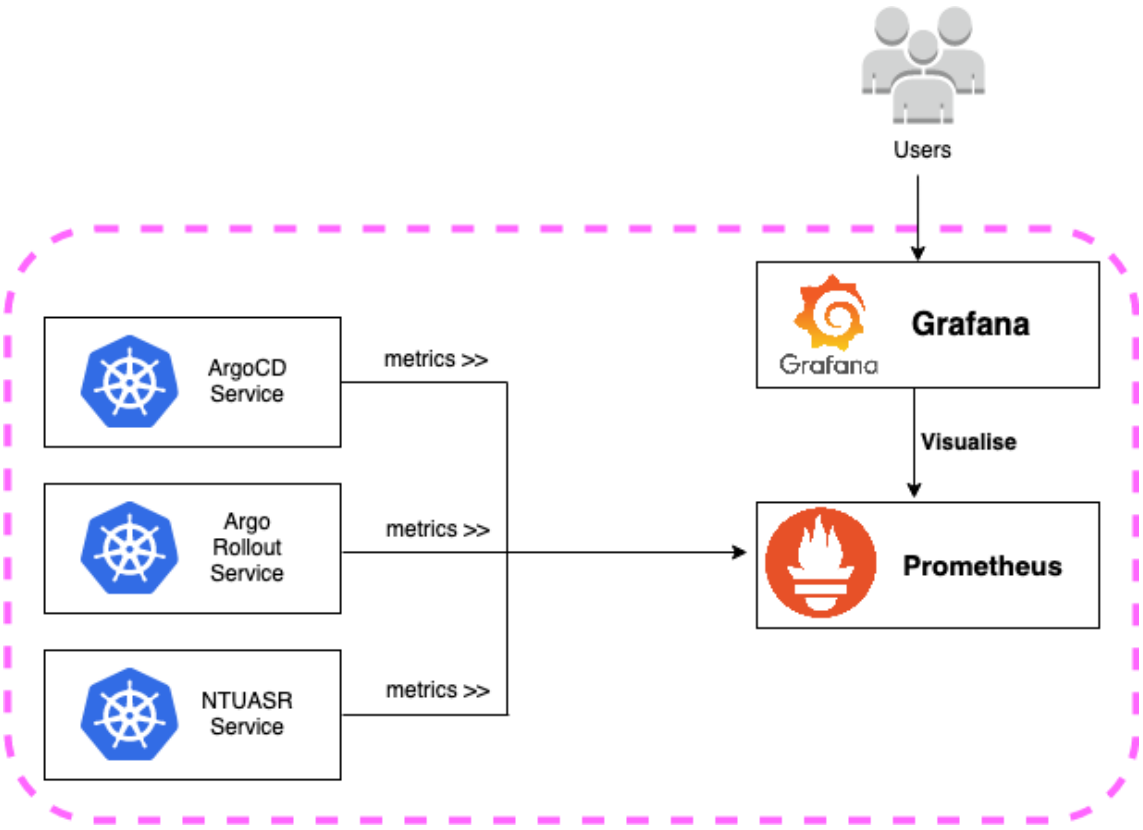

Figure 3.13: Monitoring Architecture

Figure 3.13 shows a high-level overview of the monitoring architecture. Grafana was utilized as the user interface for developers to visualize the results, on the other hand, Prometheus was used to collect metrics from the Kubernetes Cluster. Each service inside Kubernetes will export its own metrics.

3.8.4 High Level Overview of System Architecture

Figure 3.14 shows a high-level overview of the overall system architecture.

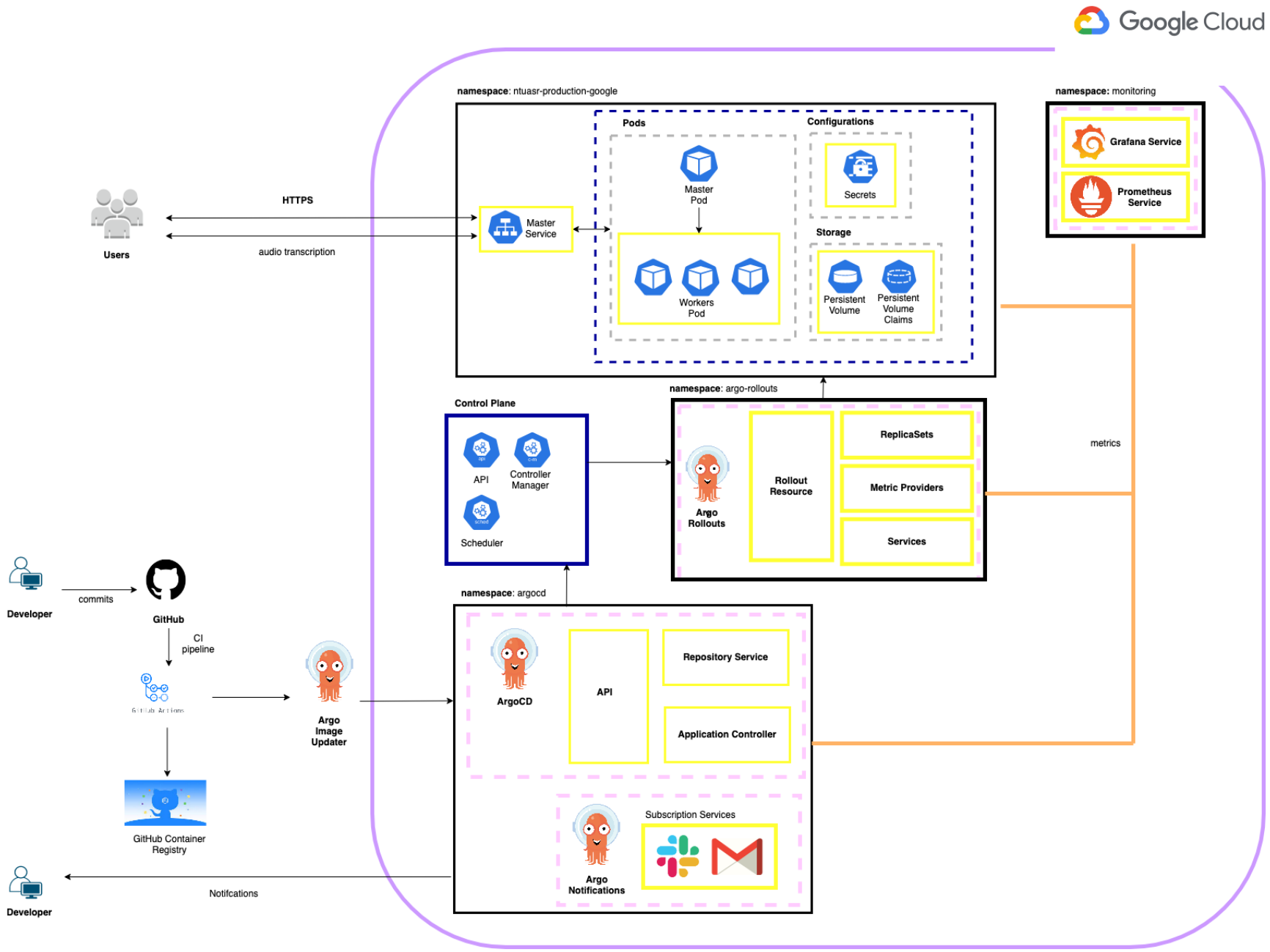

Figure 3.14: Architecture diagram of proposed solution

Chapter 4

Detailed Implementation

In this chapter, I will elaborate on the steps taken to implement the proposed solution in depth.

4.1 Initial Setup

Before setting up ArgoCD and its components, we need to set up Google Cloud Infrastructure with the help of Terraform as well as a few add-ons such as a SMTP server and a Slack Application. A detailed explanation will be in the next few sections

4.1.1 Google Cloud Platform

In the proposed solution, we will be using Google as the cloud provider. First, ensure that Google Cloud CLI is installed. Next, login into the Google Cloud Console using `gcloud auth login`. After that, create a project either via the web interface or through the command line, i.e `gcloud projects create project-name`.

In the project, we will need the following google cloud services, namely

1. **file.googleapis.com:** Network filestore for ASR models
2. **compute.googleapis.com:** Google Kubernetes Engine (GKE)

In addition, we also need to create a service account for Terraform to make authorized changes in Google Cloud. Lastly, we need to create a Google Cloud Storage bucket to store the state file for Terraform, this will serve as the remote backend. By using a remote backend, it allows the team to collaborate on infrastructure changes.

4.1.2 Terraform

To automate the provisioning of infrastructure, we use Terraform. In the proposed solution, there are several components that we need to provision

1. **Google Container Cluster** - A cluster can contain one or more node pools.
2. **Google Container Node Pool** - A group of ec2 instances for Google Cloud to run Kubernetes.
3. **Google Filestore instance** - The filestore instance is used to store the ASR model, it is recommended to use the lowest storage size of `1024GB` to save cost.

In addition, the Terraform file should define the destination of the Terraform state file in Google Cloud Storage. The file is stored in a bucket named `tf-state-prod-14`, inside the path `terraform/state` as seen from the code snippet below

```
terraform {
  backend "gcs" {
    bucket  = "tf-state-prod-14"
    prefix  = "terraform/state"
```

```
  }
}
```

Finally, we need to run the following Terraform commands to provision the required components. It might take a few minutes.

1. `terraform init` - Used to set the Terraform working directory
2. `terraform validate` - Ensure the configuration files are valid
3. `terraform plan` - Display the changes from applying the configuration files
4. `terraform apply` - Update or create the relevant infrastruc- ture

4.1.3 Uploading models

A network filestore is a distributed file system that can be used to store and share data. We will be using Google Filestore to store the ASR model. First ssh into one of the VMs from the node pool, then create a mount directory using `mkdir mnt`. Add executable permissions to the directory using `chmod go+rw mnt`. Next, mount the filestore into the VM using `sudo mount <filstore ip>:/<filestore path> <mount directory>`. Lastly, upload the model to the VM using

```
gcloud compute scp path/to/model <VM_ID>:<output_from_pwd> --
   project=<PROJECT_ID> --zone=asia-southeast1-a--recurse
```

4.1.4 Google SMTP Server

This section will briefly run through the steps required to set up a SMTP server as the documentation is available online. The SMTP is required to notify the user when ArgoCD detects important changes in the Kubernetes Cluster. Setting up a SMTP server requires an administrator account and

SMTP relay service to be set up. The relay service allows email to be sent securely within and outside of the organization.

4.1.5 Slack Application

This section will also briefly run through the steps needed to set up a Slack application bot as the instructions are available online.

1. Create a new Slack Application.
2. Under OAuth Permissions, add chat:write:bot and chat:write.customize permissions.
3. Add the bot to your channel and save the OAuth token generated from step 2, this token will be used for ArgoCD Notifications.

4.2 Continuous Integration Setup

In the proposed solution, we will use separate Git repositories for the continuous integration pipeline and the continuous deployment pipeline. This is to separate Kubernetes configuration files from the application code. In a corporate setting, an application developer will not have write permission to the deployment code while the DevOps engineer will not have write permission to the application code.

Before setting up the CI workflow, you will need a personal access token (PAT) with minimally the `repo` scope. The PAT token is an alternative to using passwords for authentication to GitHub [46].

We are using GitHub Actions for our CI workflow, the workflow is defined in a YAML file which is located in the `.github/workflow` path of the repository. Whenever a developer commits a change, the CI pipeline will be executed in a GitHub runner which is a machine that runs the workflow.

The workflow file consists of several components such as

1. **Job** - A set of steps that will be executed on the GitHub runner.
2. **Step** - A bash command or custom actions which are popular bash commands that are available in the GitHub Action marketplace.

Both jobs and steps can be executed in order or in parallel. The output from a job can be stored and retrieved in subsequent steps.

In the proposed solution, the pipeline can be found in the `pipeline.yml` file and is divided into several jobs.

1. **Lint** - Since the ASR application (master and worker) is mainly written in Python, `pylint` is used for static code analysis. Install the module using `pip install pylint`, next, the pipeline will run the command `pylint $(git ls-files '*.py') --fail-under=7` that will lint all Python files and exit the pipeline with an error when the score is below 7.
2. **Build image** - To build the image, we will use Docker build command. In addition, since we are using semantic versioning, we need to tag the image with the new version number. The version number is stored in the `package.json` file and accessed through a third-party plugin called actions/checkout@v2. Once a new image is pushed to GitHub the version number in `package.json` changes. The version number can be interpreted as MAJOR.MINOR.PATCH A minor change such as bug fixes will increment the patch number, the addition of new features will change the minor number and lastly breaking changes will affect the major number.
3. **Push Image** - Using a third-party plugin, docker/login-action@v2, the pipeline will push the image to GitHub Container Registry.

4. **Pull request (Optional Step)** - If Argo Image Uploader is used in the project this step can be omitted as it will update the CD repository automatically. Otherwise, we need to update the image tag inside the CD repository to trigger the ArgoCD pipeline. This can be achieved via the `sed` command by replacing the previous image number of the new image number. The code snippet below shows how to perform this operation:

```
sed -i "/  tag: ./c\  tag: ${{ needs.build.outputs.new-tag }}"
    canary/sgdecoding-online-scaled/values.yaml

```

The CI pipeline takes about 1hr to complete. The main reason why it is so slow is that building the image takes up a lot of time. At the end of the pipeline, a new tag number will be added to the `package.json` and a new image will be uploaded into the registry.

The two screenshots below shows the new tag number (figure 4.1) and the new image (figure 4.2).

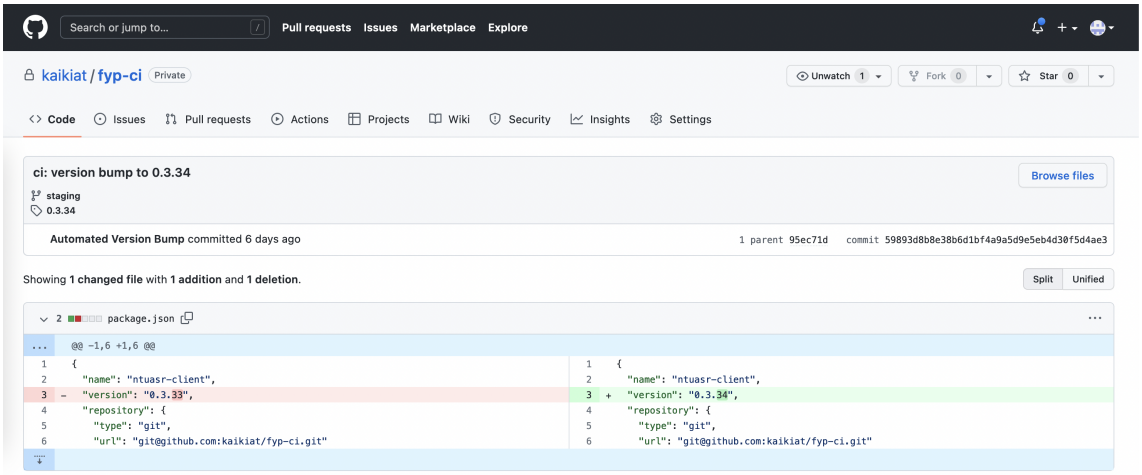

Figure 4.1: A GitHub commit showing an update in image version

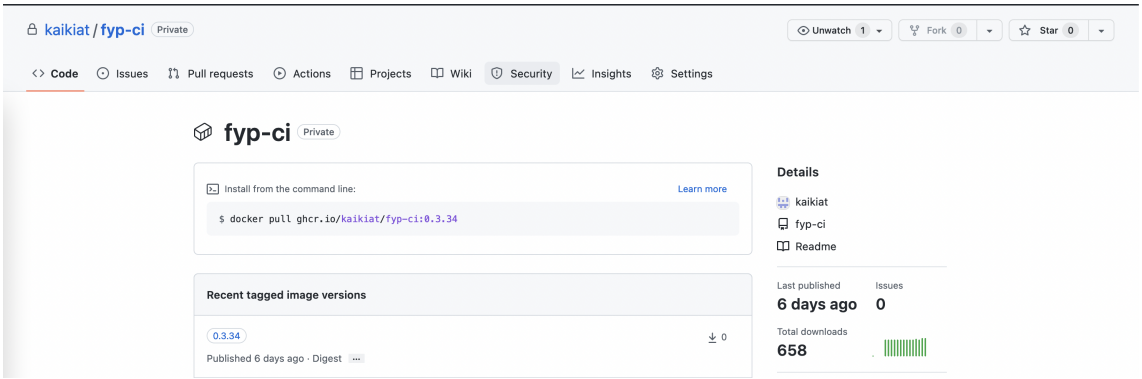

Figure 4.2: A new Docker image of 0.3.34

In addition, GitHub provides users with an interface to view the log. As seen from the figure 3.11 below, the interface shows the steps taken, the status of the step as well as provides an option to re-run the job (top-right corner)

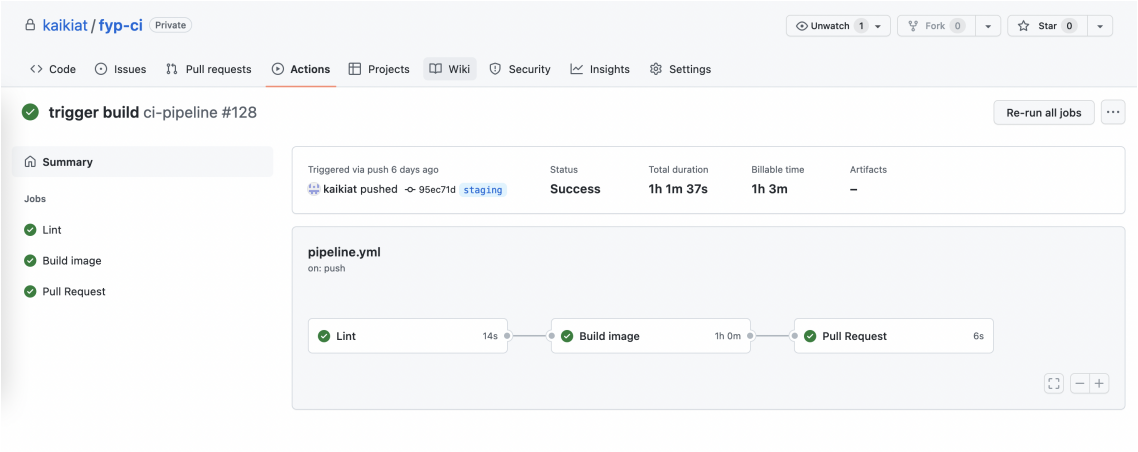

Figure 4.3: GitHub Action User Interface

4.3 Continuous Deployment Setup

4.3.1 Project Structure

The deployment repository (fyp-cd repository) should resemble the diagram below

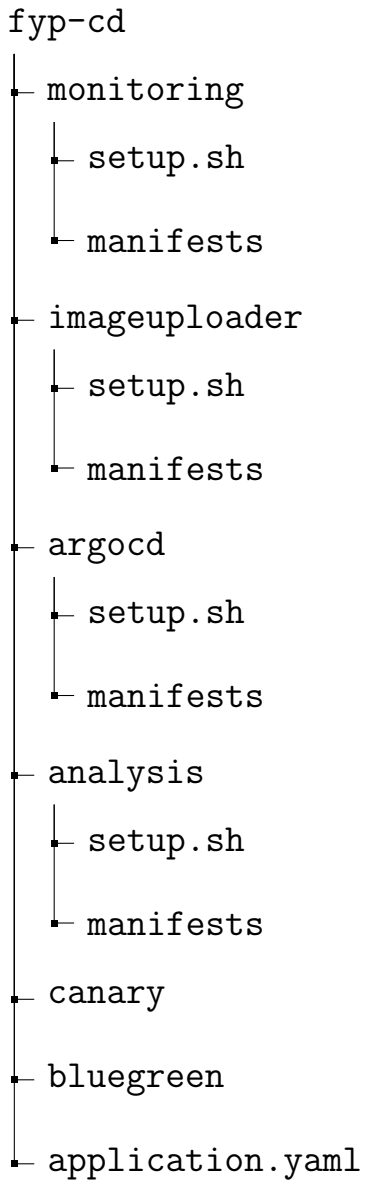

Figure 4.4: Deployment Repository Structure

Each folder and file serve a different purpose which I will further elaborate in the next few points

1. `monitoring` : Contains resources related to Grafana and Prometheus.

2. `imageuploader` : Contains resources related to Argo Image Uploader.
3. `argocd` : Contains resources related to ArgoCD, Argo Notifications, Argo Rollouts and Argo Image Uploader.
4. `analysis` : Contains resources for analyzing rollouts.
5. `canary` : Contains resources for canary rollout.
6. `bluegreen` : Contains resources for blue green rollout.
7. `application.yaml` : This file defines the ArgoCD application.

4.3.2 Argo CD

We will set up ArgoCD in the same Kubernetes cluster. All resources related to ArgoCD will be in the namespace: `argocd`. To set up ArgoCD, we need to install some custom resources provided by ArgoCD, in this project, we are using the default values. These values can be changed if you require more resources etc. Install ArgoCD's resources using the command below

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/
    argoproj/argo-cd/stable/manifests/install.yaml
```

Next, port forward the argocd-server service and you can login either through the command line or via the user interface. The default username is `admin` while the password can be obtained via decoding the argocd-initial-admin-secret initial secret using

```
kubectl -n argocd get secret argocd-initial-admin-secret -o
    jsonpath="{.data.password}"
```

At this stage, ArgoCD is still not watching any Git repository, to enable this, the owner of the repository have to bind them together using the com-

mand below

```
argocd repo add GITHUB_SSH_URL --ssh-private-key-path /path/to/ssh/
    key
```

To verify that the Git repository has been added, run `argocd repo list`. Next, configure `application.yaml` with the following

1. `spec.repoURL` : Add the url of the Git repository.
2. `spec.path` : Add the path of the helm chart for ArgoCD to watch.
3. `spec.destination.namespace` : The namespace of the asr application.

After that run `kubectl apply -f application.yaml`.

4.3.3 Argo Rollouts

Argo Rollouts is installed in a separate namespace. Helm has provided a community helm chart for Argo Rollouts.

```
    helm install argo-rollouts monitoring/argo_rollouts
```

To verify that Argo Rollouts is working, run `kubectl argo rollouts dashboard`. Then visit http://localhost:3100/rollouts to view the user interface.

4.3.4 Argo Notifications

To install Argo Notifications, make sure you have a SMTP Server or Slack application configured. The steps required provisioned them can be found in earlier chapters.

Run `kubectl apply -n argocd -f argo/manifests/config.yaml` to install the configuration map for Argo Notification. This file contains the API OAuth token for Slack and the credentials for Google SMTP server.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-notifications-cm
data:
  service.slack: |
    token: <your-slack-token>
  service.email.gmail: |
    username: kaikiat@nonscriberabbit.com
    password: <your-smtp-password>
    host: smtp.gmail.com
    port: 465
    from: kaikiat@nonscriberabbit.com
  template.app-deployed: |
    ...
  template.app-health-degraded: |
    ...
```

Figure 4.5: Configuration Map for Argo Notifications

In the configuration map, add the respective token in the date section (refer to the snippet below)

The content of the message can be message can be modified in the `template` section. An email message will be sent whenever ArgoCD syncs the application, likewise, users will be notified when the application degrades. The screenshot below 4.6 show how the email will look like.

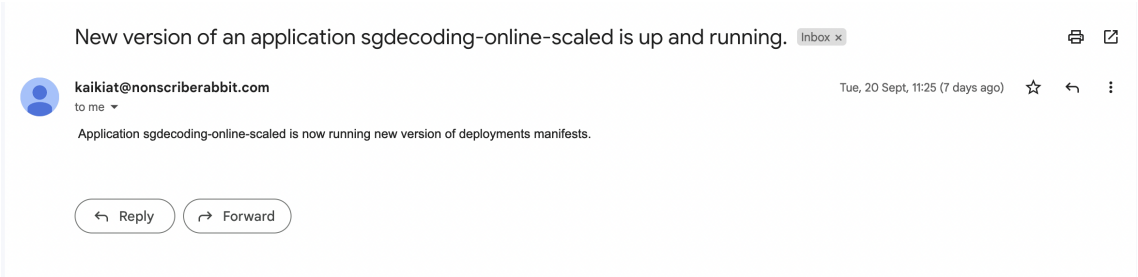

Figure 4.6: A Gmail Notification Sample

4.3.5 Argo CD Image Uploader

Argo CD Image Uploader can be installed in the `argocd` namespace. To use this plugin, we need install the custom Kubernetes resource

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj-labs/argocd-image-updater/stable/manifests/install.yaml
```

In addition, we need to add the GitHub personal access token (PAT) as Kubernetes Secrets. This to allow Kubernetes to retrieve the image from the container registry and update the Git repository whenever the version number changes. Besides that, we need to modify ArgoCD's manifest. An example can be seen at Figure 4.7.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  annotations:
    argocd-image-updater.argoproj.io/image-list: ntuasr=ghcr.io/kaikiat/fyp-cd
    argocd-image-updater.argoproj.io/write-back-method: git
```

Figure 4.7: Application Manifest for ArgoCD

The entire code to set up Argo CD Image Uploader can be found in Appendix A).

4.3.6 Prometheus and Grafana

We can install Prometheus and Grafana using Helm Charts (Refer to Appendix B). Next port forward the service using the command below

```
kubectl port-forward service/prometheus-kube-prometheus-prometheus 9090 -n prometheus
kubectl port-forward deployment/prometheus-grafana 3000 -n prometheus
```

Next, go to Dashboard and upload argorollout-dashboard.json and rgocd-dashboard.json.

4.3.7 Argo CD Analysis

Run `kubectl apply -f analysis.yaml` to install the analysis template. This Analysis template checks that the number of failed request received by the master service is less than 5 percent.

The following code snippet will be used (Figure 4.8)

```yaml
apiVersion: argoproj.io/v1alpha1
kind: AnalysisTemplate
metadata:
  name: analyse-request
spec:
  metrics:
  - name: analyse-request
    interval: 30s
    successCondition: result[0] < 0.1 || isNaN(result[0])
    failureLimit: 3
    provider:
      prometheus:
        address: http://34.124.209.46:9090
        query: |
          sum(number_of_request_reject_total{service="sgdecoding-
    online-scaled-master"})/sum(
    number_of_request_receive_by_master_total{service="sgdecoding-
    online-scaled-master"})
```

Figure 4.8: Analysis template

4.3.8 Canary Deployment

Implementing canary rollout can be done by modifying the `deployment` manifest.

The code snippet below shows an example of how we can implement canary rollout. There are 2 stages in the rollout. First, 50 percent of the new traffic will be directed to the new application. The analysis run (Refer to Figure 4.8) will take place. The step will last for 200 seconds. When the analysis run succeeds, the next stage of the rollout will take place. 100 percent of the incoming traffic be directed the new application.

```
spec:
  strategy:
    canary: #Indicates that the rollout should use the Canary
    strategy
      maxSurge: "25%" # The maximum number of replicas the rollout
    can create
      maxUnavailable: 0 # The maximum number of pods that can be
    unavailable during the update.
      analysis:
        templates:
        - templateName: analyse-request
        startingStep: 1
      steps:
      - setWeight: 50
      - pause:
          duration: 200s
      - setWeight: 100
      - pause:
          duration: 200s
```

Figure 4.9: Deployment file for Canary Rollout

During the canary rollout, we can track how many requests went to the new or older application by viewing the logs of the pods (Refer to script in Appendix C). The table below shows the number of request that goes through the pods in the span of 400s. In the first 200s, the request is evenly split between the 2 pods. After that, all the request is directed to the newer pod.

Total number of requests each pod received		
	200s	400s
Older Pod	16	17
Newer Pod	16	32

Figure 4.10: A Table showing the number of request each pod received

4.3.9 Blue Green Deployment

To implement blue green rollout, the active service and the preview service needs to be defined in the strategy section. This also means that a preview service has to be created beforehand. Figure 4.11 show an example of how blue green deployment is used in this project.

```
spec:
  strategy:
    blueGreen:
      activeService: sgdecoding-online-scaled-worker-singaporecs
    -0519nnet3
      previewService: sgdecoding-online-scaled-worker-singaporecs
    -0519nnet3-preview
      autoPromotionEnabled: false
```

Figure 4.11: Deployment file for Blue Green Rollout

Whenever a change is detected in the GitHub repository, a blue green deployment will take place. During the rollout, the developer can specify which service to test. A sample script to test the blue and green service can be found in Appendix D). This script sends a audio sample to the ASR application every 10 seconds.

In Grafana, you can create a diagram to visualise the number of requests going to the blue/green service (Refer to the screenshot below). In the graph shown below, the blue line shows the number of requests going to the blue service while the yellow line shows the number of requests going to the green service.

Figure 4.12: A graph showing the number of request going to the each service

Chapter 5

Conclusion & Future Work

This chapter will wrap up the project by summarising what the project has achieved and proposing possible improvements to the project.

5.1 Conclusion

The proposed solution has introduced another approach to implement CI/CD pipelines in Kubernetes cluster. This approach ensures that developers can have an easier time managing Kubernetes Cluster with the help of ArgoCD. It also provides rollout strategies that are more suitable for the ASR application. Lastly, we incorporated GitOps to combine the best practices of Git, this ensure that the CD pipeline is easier to maintain as most engineers are already are familiar with Git.

The Git repositories used in this project can be found in

1. **CI repository:** https://github.com/kaikiat/fyp-ci
2. **CD repository:** https://github.com/kaikiat/fyp-cd

5.2 Future Work & Possible Improvements

5.2.1 Authentication & Authorization

In ArgoCD, it is possible to assign different roles and permissions to users, such as allowing them to view or edit certain projects [47]. This feature is essential if the project needs to scale. In addition, larger teams can integrate authentication schemes such as Single Sign On (SSO) or OpenID Connect (OIDC) to reduce the need to store passwords and result in better security.

5.2.2 Security

The Kubernetes secrets stored in the Kubernetes Cluster are not encrypted. This poses a security threat since anyone with access to the cluster can get the secret. A common tool that can be used is HashiCorp Vault to encrypt and secure secrets. Vault is an identity-based secrets and encryption management system. It encrypts secrets before storing them in persistent storage [48].

5.2.3 Promotion of releases between environment

GitOps does not address the promotion of releases between environments [49]. Supposed that the staging environment is fully tested and can be promoted to the production environment. However, GitOps does not provide a clear strategy to do so. Currently, to propagate changes from one environment to another, companies often

1. Use a single environment.
2. Create a pull request to the environment. For instance, after the changes in the staging environment are validated, create a pull re-

quest to the development environment.This means that different Git branches represents different environments and this will put more strain on the CI server to run tests on each branch whenever a change is made.

Bibliography

[1] Povey, Daniel, et al. *The Kaldi speech recognition toolkit*. 2011.

[2] Wong Cassandra. *Speech Recognition system can transcribe Singapore lingo in real time*. Sept. 2018. URL: https://sg.news.yahoo.com/speech-recognition-system-can-transcribe-singapore-lingo-real-time-131406725.html.

[3] Gitlab. *What is GitOps?* URL: https://about.gitlab.com/topics/gitops/.

[4] GitHub. *Understanding GitHub Actions*. URL: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#overview.

[5] Argo-CD. *What Is Argo CD?* URL: https://argo-cd.readthedocs.io/en/stable/#what-is-argo-cd.

[6] HashiCorp Terraform. *What is Terraform?* URL: https://www.terraform.io/intro#what-is-terraform.

[7] Helm. *What is Helm?* URL: https://helm.sh/.

[8] Prometheus. *What are metrics ?* URL: https://prometheus.io/docs/introduction/overview/#what-are-metrics.

[9] GrafanaLabs. *Introduction to Grafana*. URL: https://grafana.com/docs/grafana/latest/introduction/.

[10] Red Hat. *What is containerization?* Apr. 2021. URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization.

[11] Alexander Braun. *Cloud Native with Containers and Kubernetes – Part 2*. June 2019. URL: https://blogs.sap.com/wp-content/uploads/2018/06/Container_vs_VM.png.

[12] Ron Powell. *Benefits of containerization*. Sept. 2021. URL: https://circleci.com/blog/benefits-of-containerization/.

[13] docker docs. *Docker Overview*. URL: https://docs.docker.com/get-started/overview/.

[14] APMGInternational. *Why is Agile becoming so popular in project management?* July 2017. URL: https://apmg-international.com/article/why-agile-becoming-so-popular-project-management.

[15] kubernetes. *What is Kubernetes?* Apr. 2022. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[16] kubernetes. *Pods*. URL: https://kubernetes.io/docs/concepts/workloads/pods/.

[17] kubernetes. *Nodes*. URL: https://kubernetes.io/docs/concepts/architecture/nodes/.

[18] kubernetes. *Control Plane Components*. URL: https://kubernetes.io/docs/concepts/overview/components/#control-plane-components.

[19] kubernetes. *Deployments*. URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[20] kubernetes. *Service*. URL: https://kubernetes.io/docs/concepts/services-networking/service/.

[21] kubernetes. *Secrets*. URL: https://kubernetes.io/docs/concepts/configuration/secret/.

[22] Red Hat. *What is Infrastructure as Code (IaC)?* May 2022. URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac.

[23] Red Hat. *What is CI/CD?* May 2022. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[24] Argo-CD. *What Is Argo CD?* URL: https://argo-cd.readthedocs.io/en/stable/assets/argocd_architecture.png.

[25] Argo-CD. *Features*. URL: https://argo-cd.readthedocs.io/en/stable/#features.

[26] Matthias Nguyen. *Introducing Argo CD — Declarative Continuous Delivery for Kubernetes*. URL: https://miro.medium.com/max/1400/1*0MpcMgFb4hkcqXtflGSYNQ.png.

[27] Florian Beetz, Anja Kammer, Dr Simon Harrer. *Push-based vs. Pull-based Deployments*. May 2021. URL: https://www.gitops.tech/#push-based-vs-pull-based-deployments.

[28] William Chia. *Push vs. Pull in GitOps: Is There Really a Difference?* May 2021. URL: https://thenewstack.io/push-vs-pull-in-gitops-is-there-really-a-difference/.

[29] Argo Rollouts. *What is Argo Rollouts?* URL: https://argoproj.github.io/argo-rollouts/#what-is-argo-rollouts.

[30] Argo Rollouts. *Architecture*. URL: https://argoproj.github.io/argo-rollouts/architecture-assets/argo-rollout-architecture.png.

[31] Argo Rollouts. *Canary Deployment Strategy*. URL: https://argoproj.github.io/argo-rollouts/features/canary/#canary-deployment-strategy.

[32] Red Hat. *What is blue green deployment?* Aug. 2019. URL: https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment.

[33] Argo Rollouts. *How does Argo Rollouts integrate with Argo CD?* URL: https://argoproj.github.io/argo-rollouts/FAQ/#how-does-argo-rollouts-integrate-with-argo-cd.

[34] Argo CD Image Updater. *Argo CD Image Updater*. URL: https://argocd-image-updater.readthedocs.io/en/stable/#argo-cd-image-updater.

[35] HashiCorp Terraform. *How does Terraform work?* URL: https://www.terraform.io/intro#how-does-terraform-work.

[36] HashiCorp Terraform. *Standardize your deployment workflow*. URL: https://mktg-content-api-hashicorp.vercel.app/api/assets?product=tutorials&version=main&asset=public%2Fimg%2Fterraform%2Fterraform-iac.png.

[37] Helm. *Charts*. URL: https://helm.sh/docs/topics/charts/.

[38] GitHub. *Usage Limits*. URL: https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration#usage-limits.

[39] Open up the cloud. *Observability Monitoring: An Ultimate Guide.* URL: https://openupthecloud.com/observability-monitoring-ultimate-guide/.

[40] Prometheus. *Overview*. URL: https://prometheus.io/assets/architecture.png.

[41] sysdgig. *Enable Prometheus Native Service Discovery*. URL: https://docs.sysdig.com/en/docs/sysdig-monitor/monitoring-integrations/custom-integrations/collect-prometheus-metrics/enable-prometheus-native-service-discovery/.

[42] Google Cloud. *Google Kubernetes Engine*. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview.

[43] Google Cloud. *Cloud Storage documentation*. URL: https://cloud.google.com/storage/docs.

[44] Google Cloud. *Filestore documentation*. URL: https://cloud.google.com/filestore/docs.

[45] Flux. *Frequently asked questions*. URL: https://fluxcd.io/legacy/flux/faq/#does-it-work-only-with-one-git-repository.

[46] GitHub. *Creating a personal access token*. URL: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token.

[47] ArgoCD. *User management*. URL: https://argo-cd.readthedocs.io/en/stable/operator-manual/user-management/.

[48] HashiCorp Vault. *What is Vault ?* URL: https://www.vaultproject.io/docs/what-is-vault.

[49] Kostis Kapelonis. *The pains of GitOps 1.0*. Nov. 2020. URL: https://codefresh.io/blog/pains-gitops-1-0/.

Appendices

Annex A

Argo CD Image Updater

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/
    argoproj-labs/argocd-image-updater/stable/manifests/install.yaml

# Set debug level
kubectl patch configmap/argocd-image-updater-config \
  -n argocd \
  --type merge \
  -p '{"data":{"log.level":"debug"}}'

# Add config files
kubectl patch configmap/argocd-image-updater-config --patch-file
    image_uploader/argocd-image-updater-config.yaml -n argocd
kubectl apply -f image_uploader/secrets.yaml

# Restart image uploader deployment
kubectl -n argocd rollout restart deployment argocd-image-updater
```

Annex B

Prometheus and Grafana

```
kubectl create namespace prometheus
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install prometheus monitoring/manifests/kube-prometheus-stack --namespace prometheus

# Install service monitors
kubectl apply -f monitoring/manifests/service-monitor.yaml -n argocd
kubectl apply -f monitoring/manifests/service-monitor-ntuasr.yaml -n ntuasr-production-google
kubectl apply -f monitoring/manifests/service-monitor-ntuasr-preview.yaml -n ntuasr-production-google
```

Annex C

Canary Deployment Verification Script

```
# Original Master Pod (Ensure that there are no erroneous pods)
NAMESPACE=ntuasr-production-google && \
WORKER=$(kubectl get pods --sort-by=.metadata.creationTimestamp -o
    jsonpath="{.items[0].metadata.name}" -n $NAMESPACE) && \
kubectl logs $WORKER -f -n $NAMESPACE

# Preview Master Pod
NAMESPACE=ntuasr-production-google && \
WORKER=$(kubectl get pods --sort-by=.metadata.creationTimestamp -o
    jsonpath="{.items[2].metadata.name}" -n $NAMESPACE) && \
kubectl logs $WORKER -f -n $NAMESPACE
```

Annex D

Testing Blue Green Deployment

```
'''
    This script sends a audio file to the blue service

    To test the green service,
    Change `master` to `preview`
'''
import os
import logging
import subprocess
import time
from pathlib import Path

logging.basicConfig(level=logging.INFO, format='%(message)s')
logger = logging.getLogger(__file__)
logger.setLevel(logging.INFO)

def main():
    cmd = r"kubectl get svc sgdecoding-online-scaled-master -n
    ntuasr-production-google --output jsonpath='{.status.
    loadBalancer.ingress[0].ip}'"
    process = subprocess.Popen(cmd, stdout=subprocess.PIPE, shell =
    True)
```

```
    output, error = process.communicate()
    ip_address = output.decode('utf-8')
    logger.info(f'Ip Address : {ip_address}')

    min = 60
    duration = min * 4
    sleep_duration = min * 3
    logger.info('Sleeping for '+ str(sleep_duration) + ' seconds')
    time.sleep(sleep_duration)
    end = int(time.time()) + duration

    cmd = f"python3 client/client_3_ssl.py -u ws://{ip_address}/client/ws/speech -r 32000 -t abc --model='SingaporeCS_0519NNET3' client/audio/34.WAV"
    while int(time.time()) < end:
        process = subprocess.Popen(cmd, stdout=subprocess.PIPE, shell = True)
        output, error = process.communicate()
        logger.info(output)
        time.sleep(10) # test the new service less often

if __name__ == "__main__":
    main()
```