SRE Challenge

DevOps Model Proposal

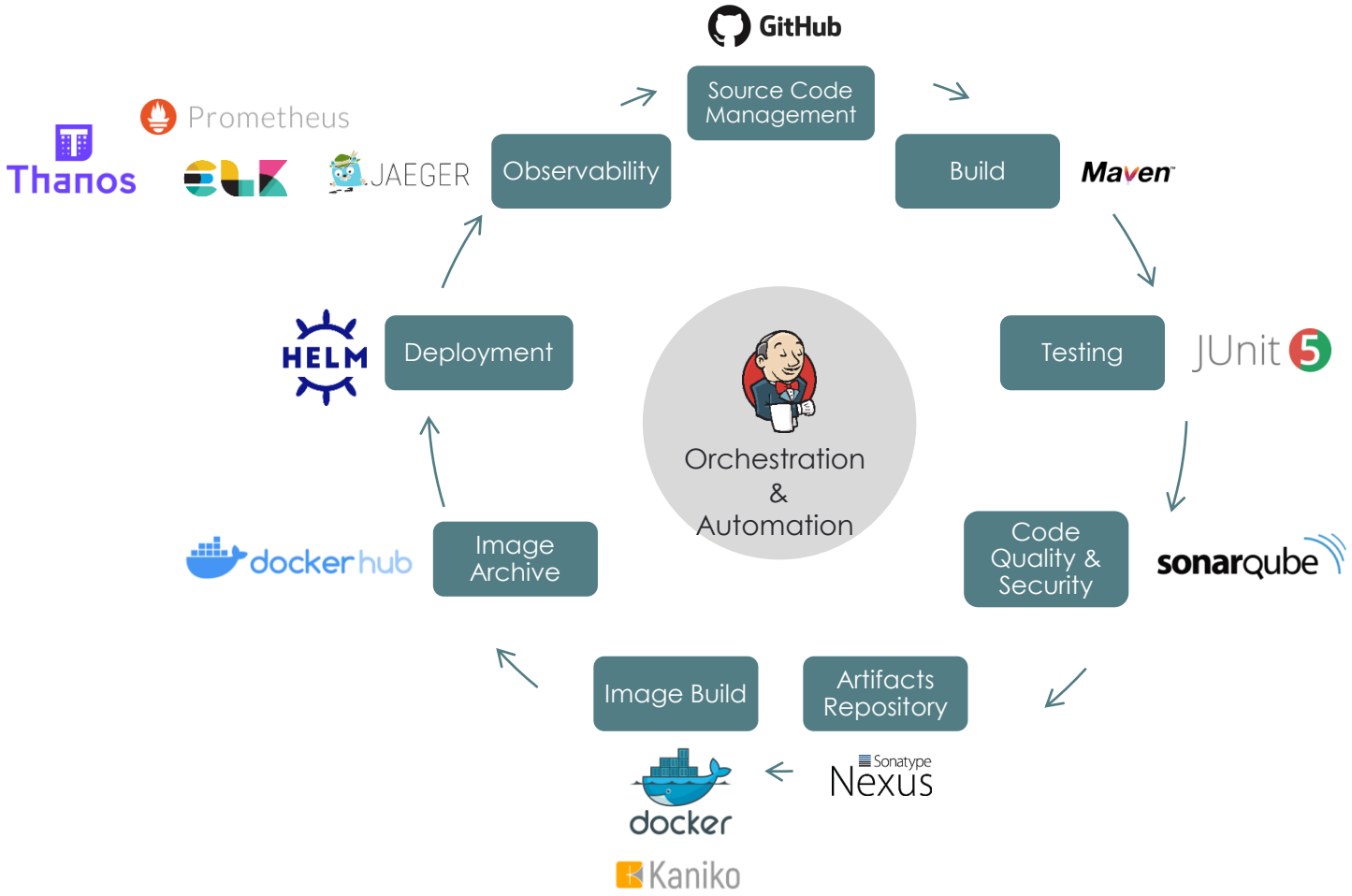
Alberto Azuara García

01. Introduction
Description

- This document explains the DevOps model proposed for the continuous development of applications, the deployment of applications, the architecture of the monitoring stack in HA and the application for collecting logs in Kubernetes clusters.
- First, it is detailed the Open-source tools used to implement this DevOps cycle. It is also detailed how to deploy infrastructure as code regardless of which cloud provider.
- Then, the model of branches used to share the code is proposed. Each of the steps performed by the pipeline that is included in the repository.
- The canary deployment model is explained to reduce errors and application downtime.
- Finally, the architecture of the monitoring stack in HA is proposed, using Thanos, Prometheus and Grafana. The logs collection system using ElasticSearch, Beat and Kibana stack.
- The repository can be found at the following link:
- https://github.com/alazuga/apiSampleJava

02. Global Solution

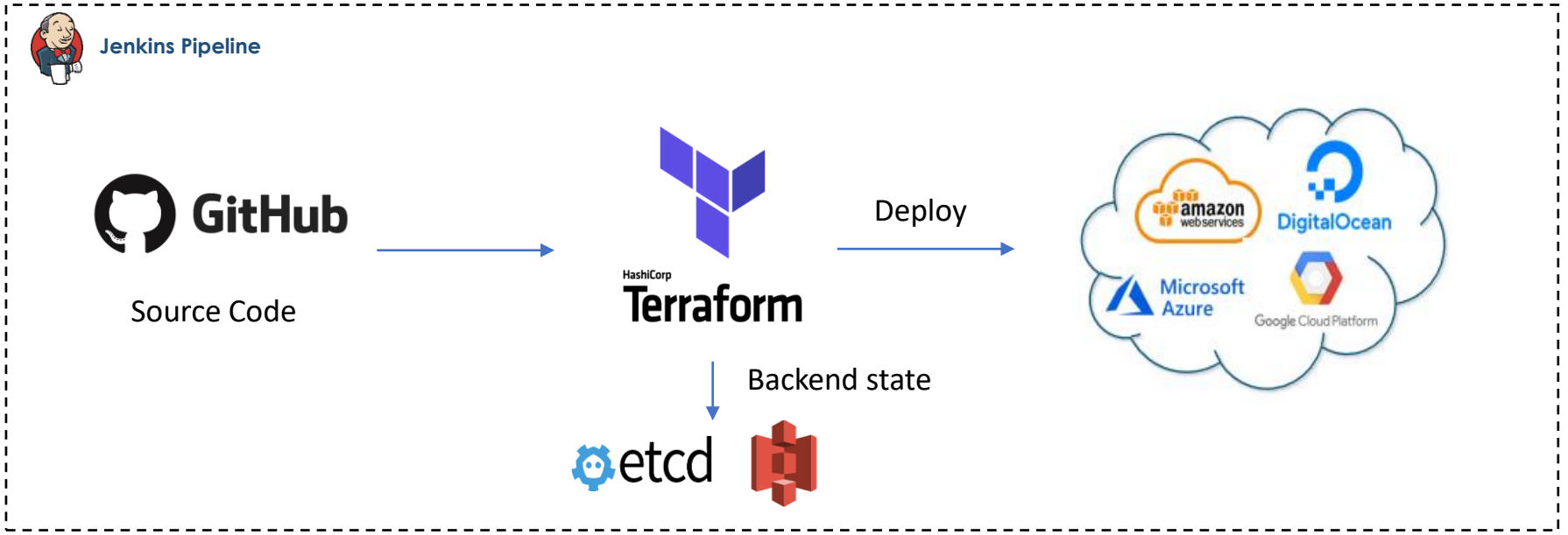
Functional overview



DevOps cycle proposed for the application, the tools for compilation, testing and deployment and monitoring container-native applications.

03. IaC

Infrastructure as a Code



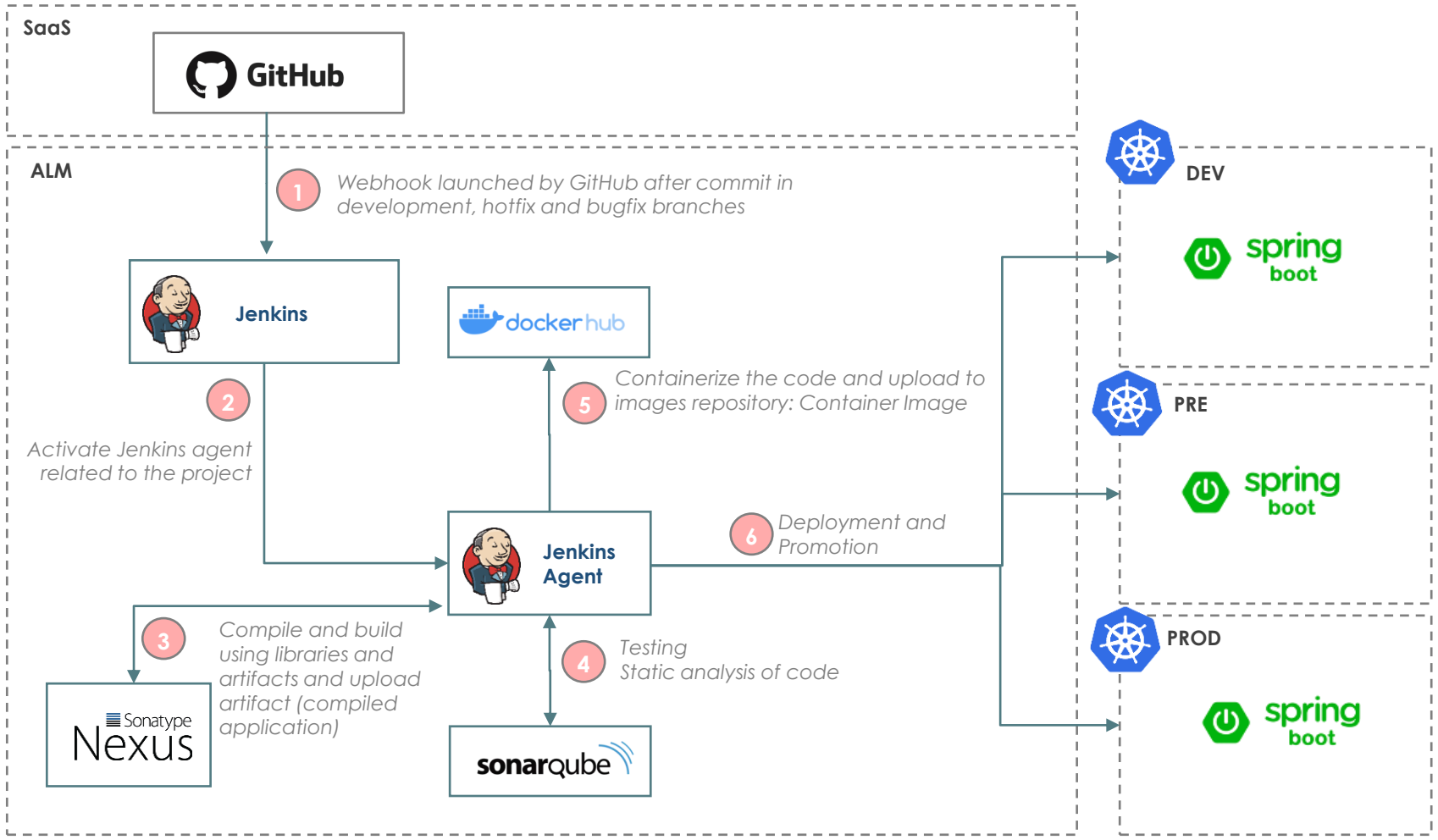
In this proposal, I used Terraform for the deployment of the infrastructure.

The repository follows the all-in-code philosophy and any desired changes to the infrastructure are repositioned and traced. Any change to that code triggers a deployment pipeline that deploy these changes automatically to cloud provider.

04. ALM definition

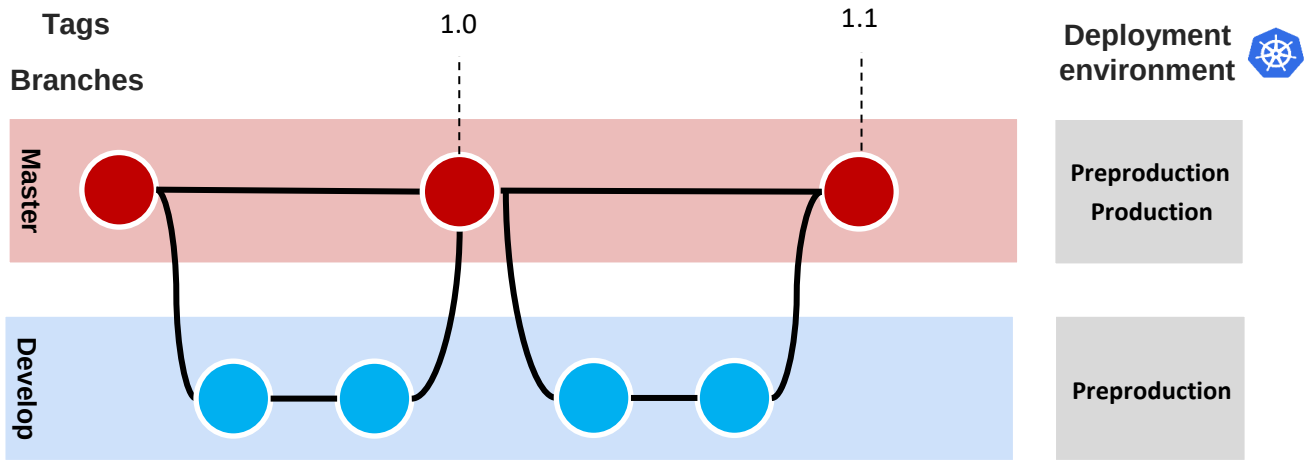
CI/CD flow

This slide describes the CI / CD flow with Jenkins for software development.



05. ALM definition

Branching and flow strategy



- All developers work in a single branch with open access to it. It is the develop branch and all jobs are directly integrated into it.
- The master branch enables the stabilization of the version.

Feature Branch Development is the branching model which be followed be developer teams, as contains many important advantages:

- Simple model that allows to maintain focus on development.
- Maximize continuous integration
- Code merges are simple
- Allows fast iterations
- Easy implementation in small teams

06. Local debug
CI/CD flow

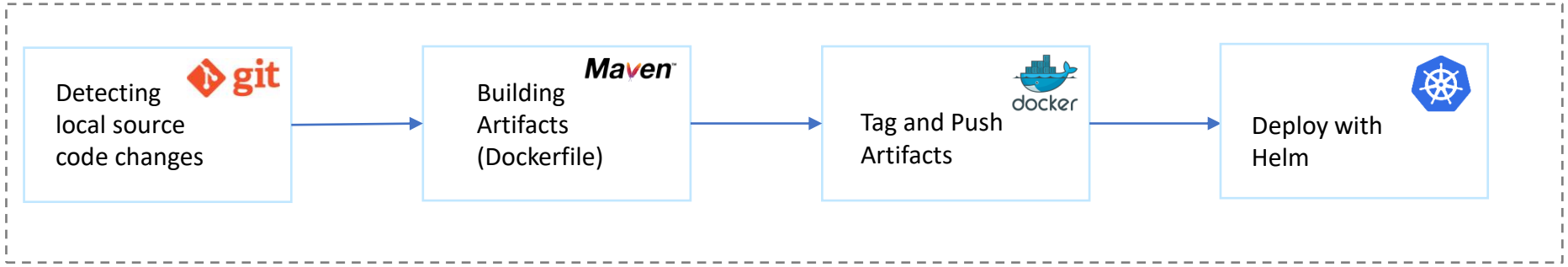
SKAFFOLD

For local development developers can use the Scaffold tool, this tool automates the deployment process with a simple pipeline defined in scaffold.yaml file.

In the Github repository we can find the *watch.sh* file, the developer can run the script and this script will automatically compile the Maven Project locally and deploy it using helm in the Kubernetes cluster configured by default on their computer.

Skaffold facilitates continuous development for Kubernetes applications. Skaffold handles the workflow for building, pushing and deploying the application. It also provides building blocks and describe customizations for a CI/CD pipeline.

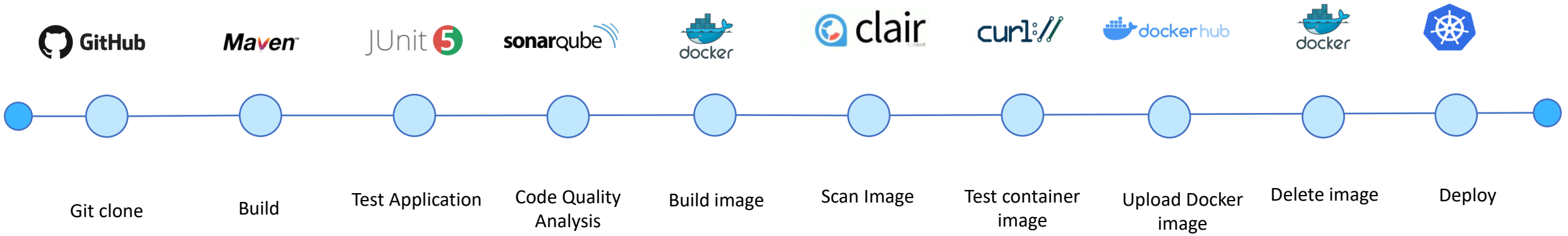
```
$ ./watch.sh
```



07. Example pipeline description

CI/CD flow

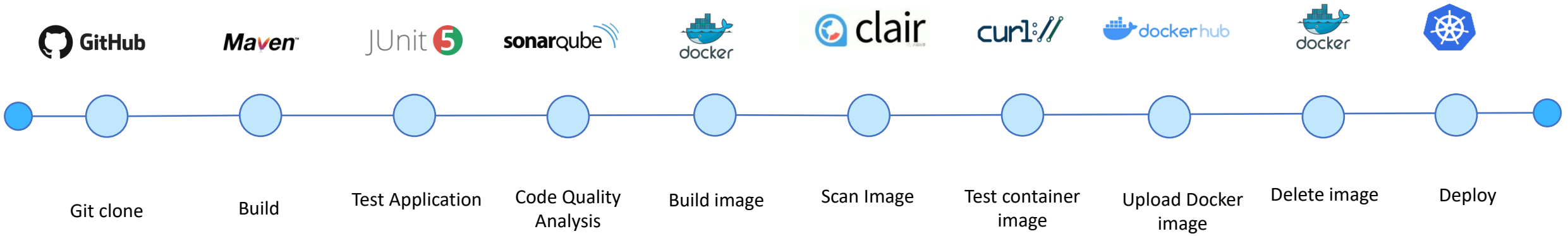
This slide describes the example CI / CD pipeline *Jenkinsfile* included in the repository:



1. Git clone: the repository is downloaded from the pipeline with the latest changes. The pipeline is multibranch, so depending on which branch the changes have been made, the pipeline will be conditioned to make certain steps depending on the branch in which the changes have been made.
2. This is followed by the compilation of the Maven project.
3. Once the project is compiled, the tests will be carried out: unit test and test with JUnit, once the tests are done, the results will be published in an .xml file represented by Jenkins in charts.

07. Example pipeline description
CI/CD flow

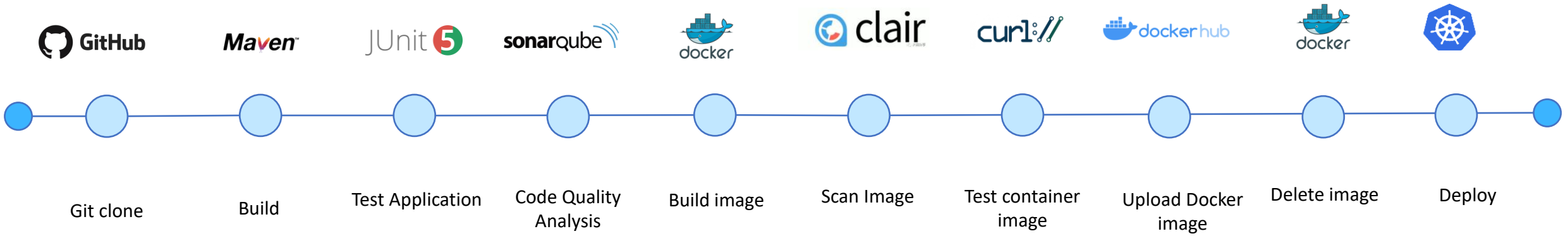
This slide describes the example CI / CD pipeline *Jenkinsfile* included in the repository:



4. When the unit tests are done, the pipeline has the Analysis code stage, SonarQube will be used which will analyze the possible vulnerabilities and improvements in the code, depending on the defined standards, it will approve or cancel the pipeline execution.
5. Once the code has been verified, the image will be compiled using Docker or Kanico (no root required).
6. Using tools such as clair or similar, the vulnerabilities of the container will be checked, detecting possible security vulnerabilities in the base images or in the installed tools.

07. Example pipeline description
CI/CD flow

This slide describes the example CI / CD pipeline *Jenkinsfile* included in the repository:



7. Test contaner image using tools like curl and load testing probes. If the tests fail, the pipeline will be canceled as the code is not robust enough to be deployed.
8. When the image was tested and validate, uplodad the image to docker hub.
9. Delete unused images and intermediate images.
10. Finally, the pipeline deploy application using Helm chart include in the repo.

08. Deployment strategy

Istio for canary deployment

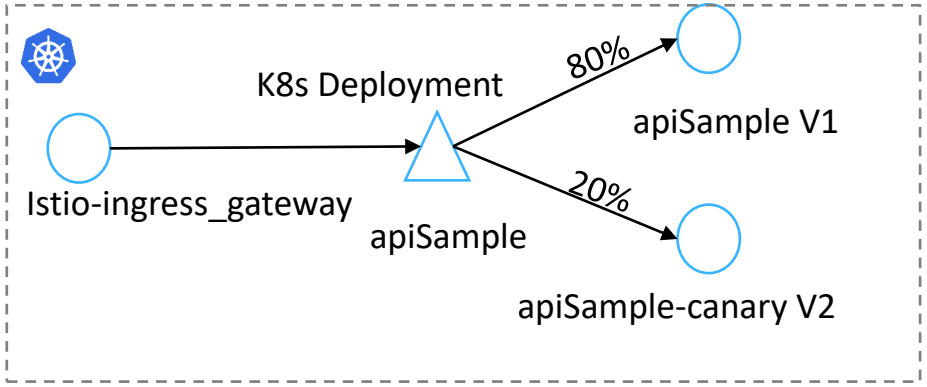
For deploy and add observability of the application, I propose the use of the Istio service mesh. This application allows to transparently add capabilities like observability, traffic management, and security, without adding them in code.

Istio generates detailed telemetry for all communications and Routing traffic. Istio's traffic routing rules let you easily control the flow of traffic and API calls between services.

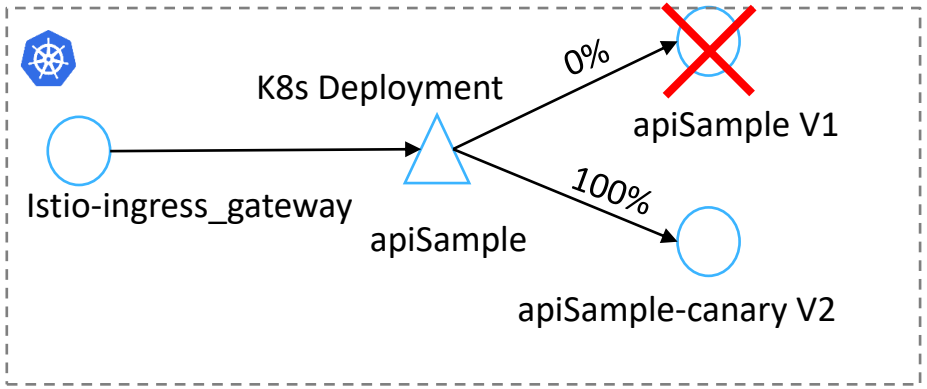
I recommend to use canary deployment as an early warning indicator with less impact on downtime and allows for flexibility for businesses to experiment with new features: if the canary deployment fails, the rest of the services aren't impacted. Its very easy implement this deployment with Istio.

- First, deploy with helm the new version of the application and route only 20% of the traffic with istio weight routing. Test and validate the V2 application.
- Then, increment in different stages the weigh of the traffic route to 100% and the app V1 to 0% of the traffic.
- Finally, undeploy V1 application.

Initial stage

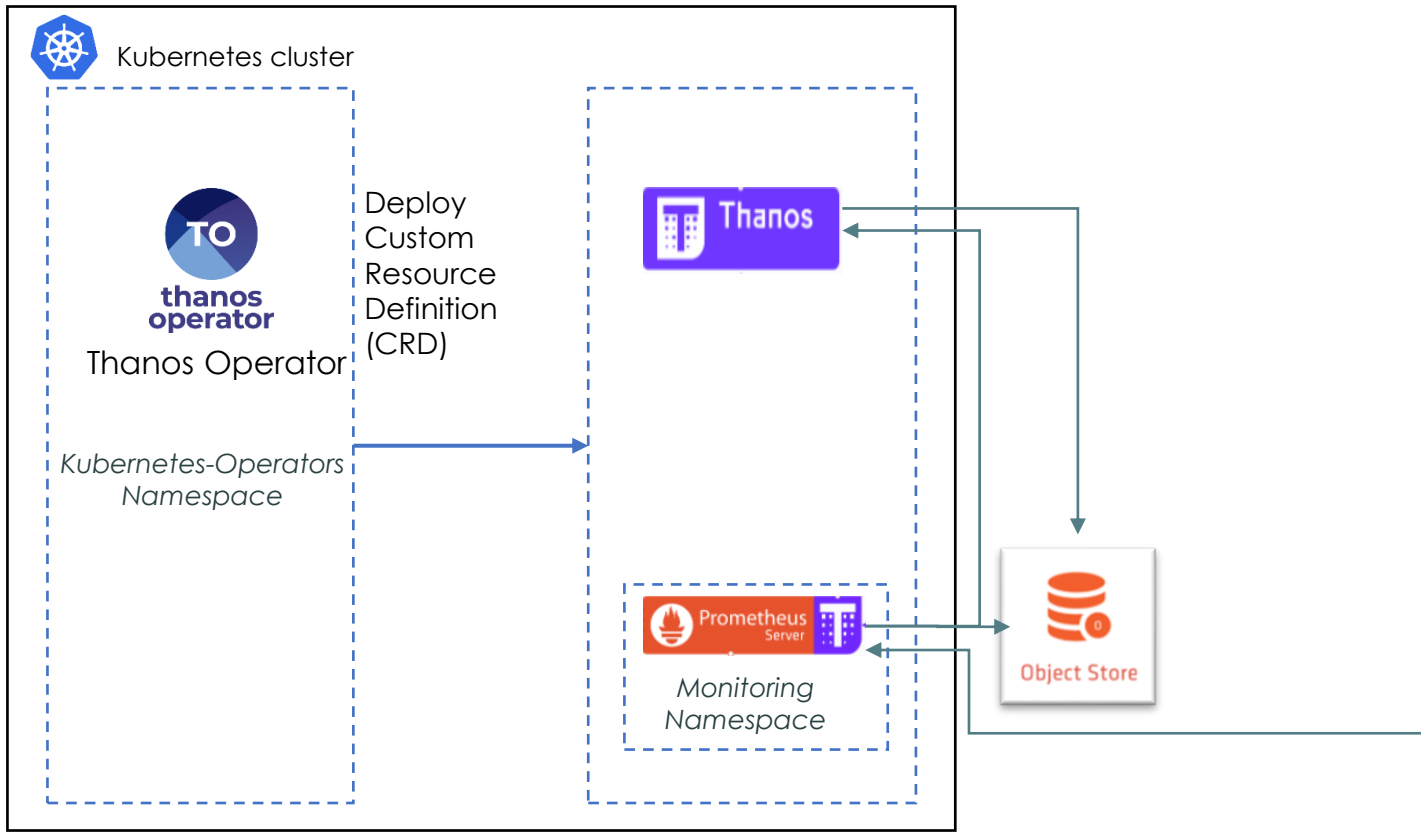


Final stage



09. Monitoring
Overview

To monitoring and instrumentation the application, I purpose Thanos operator. Thanos provides a global query view, high availability, data backup with historical, data access as its core features in a single binary. Thanos install an HA Prometheus installation.
Prometheus is a pull-based system. It sends an HTTP request, a so-called scrape, based on the configuration defined in a file.



Thanos operator

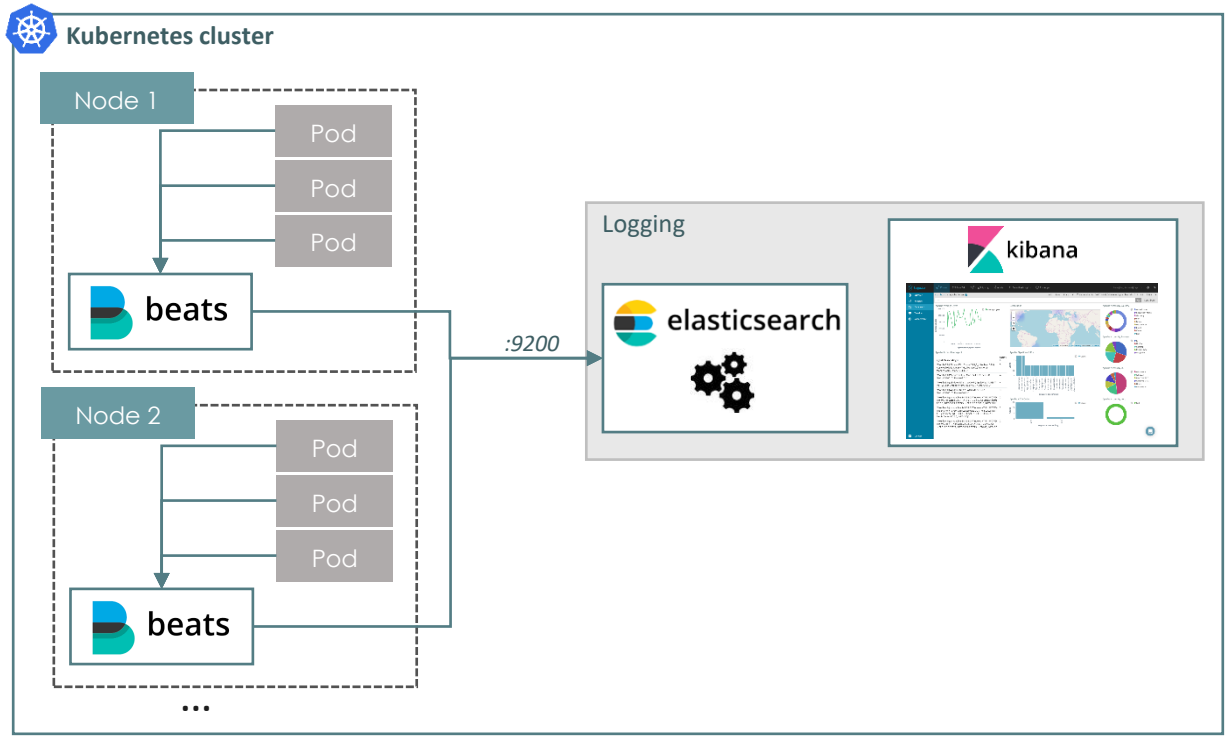
- IaC configuration
- Achieve HA using Thanos querier
- Query historical data with Thanos store
- Reduce data size with Thanos compactor

Grafana

10. Logging
Overview

When running multiple services and applications on one or several Kubernetes clusters, it's required to manage a centralized, cluster-level work stack, which will assist and analyze the serious volume of logs and traces made by the Pods.

For this challenge, I purpose use EFK stack. It is an Open Source logs centralized solution, which is composed by Elasticsearch, Beats and Kibana



beats

Component which gathers logs from pods, applies some format rules and forwards the logs to centralized repository

elasticsearch

Indexed and centralized repository where the logs registries are stored.

kibana

User friendly console to visualice the log information