Release Management Report

Craig Dillon
Technological University Dublin
x00205790@mytudublin.ie
MSc. DevOps
Enterprise Architecture Deployment
CA2

Contents

Introduction

The purpose of this document is to provide a structured and detailed overview of the microservice deployment project. The following topics will be discussed;

- GitOps methodology
- Microservice overview
- Container orchestration platform
- Deployment strategy
- Infrastructure deployment strategy

1.1 GitOps methodology

This project employs a GitOps methodology which is a framework that builds on DevOps best practices to provide effective deployment strategies in a microservice environment. *The three main pillars of GitOps are:*

- *Git is the single source of truth*
- *Treat everything as code*
- *Operations are performed through Git workflows*

The GitOps Principles are defined as;

- *Declaritive - A system managed by GitOps must have its desired state expressed declaratively.*
- *Versioned and immutable - The desired state is stored in a way that enforces immutability and versioning retains a complete version history.*
- *Pulled automatically - Software agents automatically pull the desired state declarations from the source*

Vinto & Bueno (2022)

Overview of microservice application

The microservice application is a basic website to enter and query car details and is based on a collection of loosely coupled services, each performing its own specific business logic. This approach allows for greater flexibility, scalability and maintability compared to monolithic applications. The application consists of several microservices deployed on an Azure Kubernetes Service (AKS) cluster with Cilium operating as both the network layer and the service mesh. Each microservice is encapsulated within a Docker container allowing for independent configuration, deployment and scaling. The application is designed to allow a user to enter and query a database for car details with the fields, make, model, year and registration. The application is written in React and is split between frontend and backend containers which are then deployed to AKS. The mySQL database uses persistent storage within the cluster and the application also uses the Memcached service for session persistence.

2.1 Communication patterns between microservices

Communication between the frontend and backend is handled using a RESTful API between the services. From here the backend service then communicates with mySQL to perform CRUD (Create, Read, Update, Delete) functions on the database. For example, when a user creates a new entry on the frontend, this data is submitted to the backend via the RESTful API and the backend handles the entries in the relevant fields in the database. The service also relies on Memcached for session persistence and creating session cookies which provides a level of fault tolerance in the application should a pod fail or roll back to a previous version.

2.2 Data layer

Cilium is operating as both the network layer and service mesh. This brings advantages such as greater visibility, improved performance and removing the requirement of running a sidecar proxy for service mesh functions and instead uses a per node proxy. It deploys with a control plane which is managed by the Cilium agent and handles configuration updates and applies them to the orchestrators running on each node, *Nagendra & Hemavathy (2023)*. Cilium differs from many other options in that it utilises eBPF allowing it to run on at the kernel level of the hosts Kubernetes is deployed on. Isovalent, the developers of Cilium state that eBPF allows for real-time packet manipulation and processing by attaching functions to specific events like packet ingress and egress. As a result this can improve the efficiency and scalability by offloading tasks from the CPU, *.IO (2024)*.

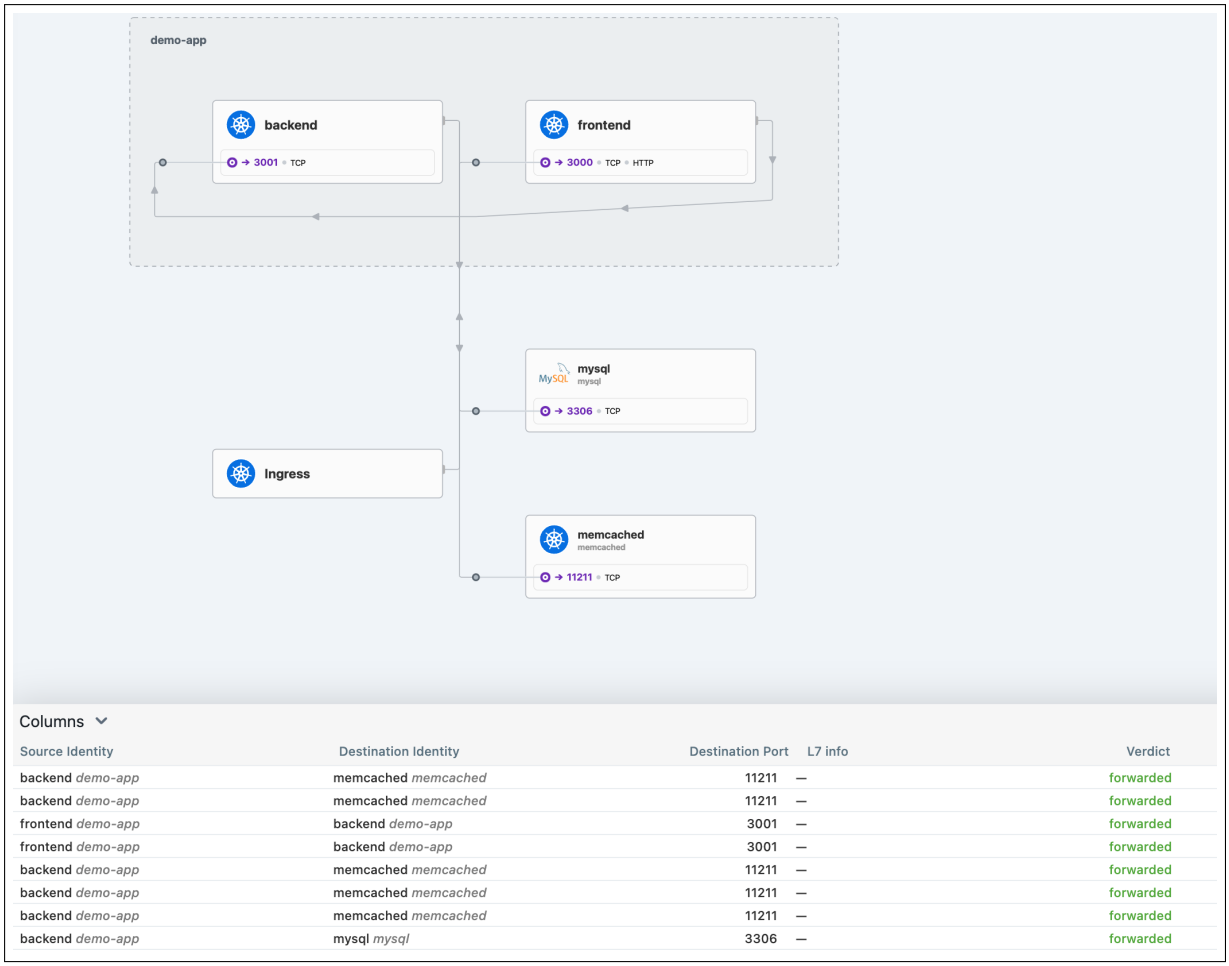

Figure 1: Hubble UI showing real-time logs and communication between services (Source: Author, 2024)

Container orchestration API

AKS (Azure Kuberenetes Service) is the orchestration platform hosting the application and auxiliary services. AKS was chosen due to familiarity and customisation options. The initial deployment used the Istio-based service mesh add-on for AKS. While this streamlined the deployment of the cluster and service mesh there were limitations using this implementation such as lack of Gateway API support and documentation was at times lacking in certain areas meaning configuration beyond the initial setup became challenging, *shashankbarsin (2024)*. As a result, other solutions were explored and a deployment using Cilium as the network layer and service mesh proved to be an easy-to-use, full-featured service.

3.1 Features of AKS

AKS is a mature and well established orchestration platform. As a result, there are a number of methods that can be employed to automate deployment. This project uses Terraform with the Azure provider to deploy the AKS cluster and other cloud based components. The deployment is handled in a Gitlab CI/CD pipeline which verifies the HCL configuration and is applied manually when the operator is ready to do so. AKS

provides several cost-effective, easy to deploy features such as;

- Scalability - AKS allows for both vertical and horizontal scaling by either adding more virtual machines or nodes as needed.
- Security - As a hosted service AKS can leverage a variety of access controls including Azure Active Directory integration, role based access controls, secrets management and network policies.
- High availability - AKS provides built-in high availability features such as automatic node repairs, node pools and cluster autoscaling.
- Managed service - Azure manages the backend infrastructure of the cluster, removing overhead on teams to manage the logistics of provisioning, scaling and upgrading Kubernetes clusters.

Azure AKS is a cost-efficient platform as it removes the engineering time requirements for provisioning and managing a Kubernetes cluster. AKS also operates on a pay per usage model where customers are only charged for the resources used. There are no upfront costs and no long term commitments. Overall, AKS is a fully featured, mature and easy to manage Kubernetes platform, *Microsoft (n.d.)*.

Continuous software delivery

This project follows a GitOps process where the application container is built using a Gitlab CI/CD pipeline. Using versioning tags for the containers, ArgoCD then detects changes in the source repository requesting the newer version of the container image which then triggers a canary deployment by Argo Rollouts. This is a continuous delivery process that is completely automated. The backend service and frontend service have their own directories in the Gitlab repository and pipelines are controlled in a DAG (Directed Acyclic Graph) manner which means that changes to each sub directory will trigger a child pipeline within the modified directories only.

Deployment strategy

The deployment strategy uses Argo Rollouts to perform a canary deployment. This allows for granular control of the deployment by migrating the updated service into production in steps that can either be triggered manually or controlled using a range of parameters such as wait times or relying on feedback from a monitoring too, Prometheus for example. This strategy, when implemented correctly can reduce the impact of bugs and other issues to the entire user base and allows for rollbacks to the previous working deployment at any stage in the rollout.

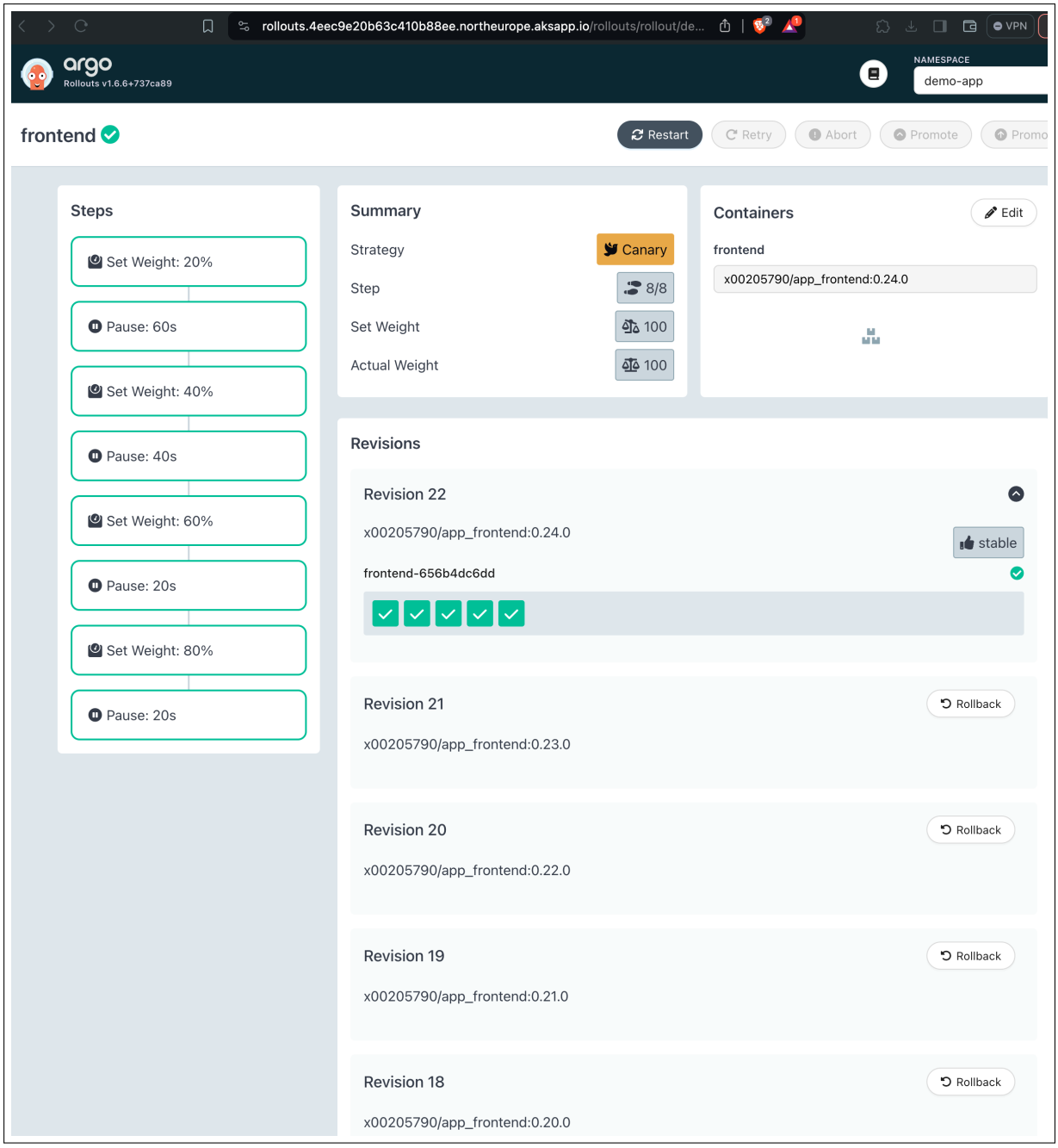

Figure 2: Argo Rollouts dashboard showing a successful canary deployment and option to rollback to any previous deployment (Source: Author, 2024)

4.1 Rollback plan

This deployment strategy allows for either manual or automatic rollback at any step in the canary deployment. For example, using a Prometheus Analysis Template to detect pod issues can trigger an automatic rollback to the previous known working version. Alternatively, an engineer may detect an issue and choose to rollback manually via the Argo Rollouts UI, this option is also available as a CLI (command line interface) tool.

4.2 Backup and restore strategy

Infrastructure deployments are maintained in a repository as IaC (Infrastructure as code) meaning a redeployment is readily available. The application manifests are also stored in a repository and as part of the infrastructure deployment Cilium is installed as the network layer and ArgoCD is installed and configured to automatically being deploying from the repository containing the manifests. The database uses a persistent volume on the cluster, meaning in the event of a full AKS deployment this data would be lost. Relying on a dedicated Azure Database service or backups to an external service would mitigate this issue.

4.3 Scaling strategies

The AKS cluster is deployed using terraform as has a node count set to minimum 1 and maximum 3. Depending on requirements these figures could be set higher or lower and AKS will handle scaling of Kubernetes nodes as required. A number of VM sizes for the nodes can be chosen at deployment allowing for flexibility in terms of how much vertical and horizontal scaling is desired.

Monitoring

This project uses Promstack, a community developed package that includes Prometheus, Grafana and Alertmanager with pre-configured query sets to collect, visualise and provide a platform for creating alerts. Cilium also has an observability platform called Hubble which provides realtime monitoring of the network infrastructure along with visibility of communication and behaviour of services on the cluster. Cilium allows for the creation of network policies to control the flow of data between services and Hubble UI provides a frontend to track this.

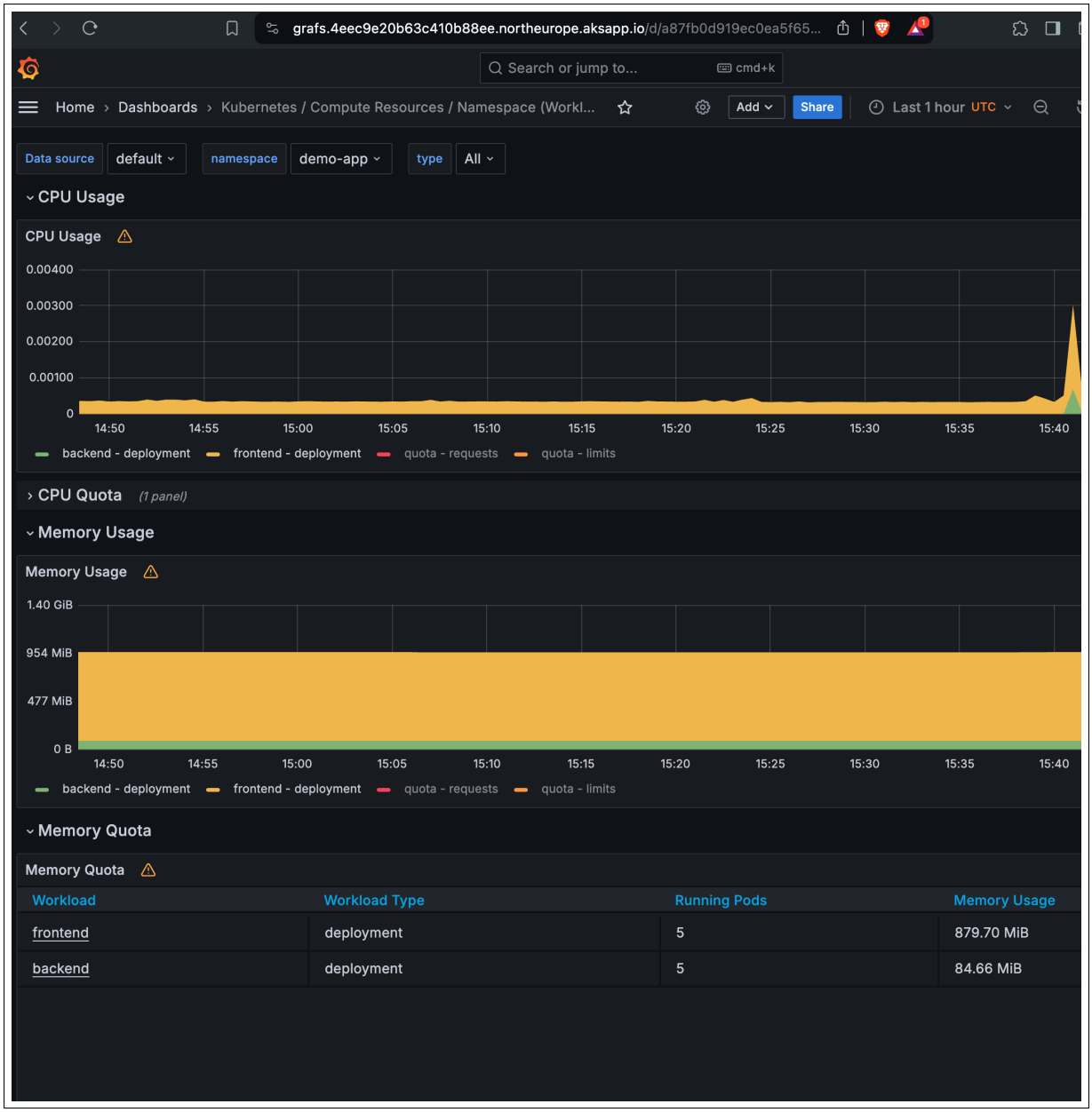

Figure 3: Grafana dashboard showing statistics on the application namespace (Source: Author, 2024)

Conclusion

This project highlights the potential of Kubernetes by leveraging CI/CD pipelines and GitOps methodologies utilising services such as; ArgoCD, Argo Rollouts, Prometheus and Cert-Manager to automatically deploy microservices to a cluster in a manner that is ready for production and without human intervention.

6.1 Improvements

As this project was for demonstration purposes there are a number of improvements that would be essential to implement before this could be placed into production.

- Use of a persistent secrets engine - instead of relying on the in built Kubernetes secrets engine which allows users of a namespace access, consider an alternative service such as Hashicorp Vault or Azure Key Vault.
- Deploy a dedicated database server - Cloud providers have a range of database services that provide resilience and geo-redudancy. This project uses a persistent storage volume that will be erased in the event of a cluster redeployment.
- Analysis templates for canary deployment - As discussed earlier, this project uses a basic canary deployment. Leveraging metrics provided by Prometheus to create customised parameters that define if the rollout can continue would be essential. This can be further improved upon by implementing performance testing with an application like k6s to ensure the new deployment can meet predefined performance thresholds.
- Centralised logging - a centralised logging service would be essential for troubleshooting any issues that may occur on the cluster, Loki is one such service.
- Dependency management - implementing dependency management software on the repository, RenovateBot for example, ensures that dependencies and vulnerabilities are managed in an automated and timely manner.
- Demo app - The demo app created is inconsistent with POSTing and GETing from the backend. Error logging states this is something to do with CORS (Cross-Origin Resource Sharing) handling on the backend. Testing by using curl to POST or GET to the backend from any of the frontend containers works, suggesting there is an issue with the frontend application rather than any networking or cluster misconfiguration.

References

.IO, C. (2024), 'How software networking is taking over the world'.
URL: *https://cilium.io/blog/2024/02/12/how-software-networking-is-taking-over-the-world/*

Microsoft (n.d.), 'Managed kubernetes service (aks) | microsoft azure'.
URL: *https://azure.microsoft.com/en-us/products/kubernetes-service*

Nagendra, T. & Hemavathy, R. (2023), Unlocking kubernetes networking efficiency: Exploring data processing units for offloading and enhancing container network interfaces, *in* '2023 4th IEEE Global Conference for Advancement in Technology (GCAT)', IEEE, pp. 1–7.

shashankbarsin (2024), 'Istio-based service mesh add-on for azure kubernetes service - azure kubernetes service'.
URL: *https://learn.microsoft.com/en-us/azure/aks/istio-about*

Vinto, N. & Bueno, A. (2022), *GitOps Cookbook*, O'Reilly Media.

List of Figures

List of Tables

Appendix A: Figures

Appendix B: Other items

Promstack repository `https://github.com/jbkc85/promstack`
Goldpinger repository `https://github.com/bloomberg/goldpinger`
Argo Rollouts `https://argoproj.github.io/rollouts/`