Revolut HomeTask

In this document you will find the details of my proposed solution for the Revolut HomeTask.

You will find components description, AWS architecture diagrams, CI/CD workflow diagram and some notes about missing pieces and alternatives.

For full disclosure I will be adding notes regading my experience on each part of the proposed solution.

I am always open for discussion and happy to share different points of view. Also, if you find there is any important part missing, please let me know.

+34623104290

Jmrs@protonmail.com

The API

Following the requirements the API responds to:

- PUT `/hello/<username> { “dateOfBirth”: “YYYY-MM-DD” }`
- GET `/hello/<username>`

Additionally, it responds to:

- GET `/metrics`
  - Prometheus metrics
- GET `/docs`
  - Swagger style docs

The code

Experience notes

- This is my second time using FastAPI.
- I implemented an API long time ago and it was quite basic.
- I'll have some proefficient programmer to take a look before going into prod

Built using FastAPI to facilitate the implementation It introduces minimal fuss and comes with a quick start up and plenty of examples.

Unit tests are included, they use pytest and shows 100% coverage for api file and 98% overall.

Take a look at README.md

The GitHub CI/CD

Experience notes

- This is my first time using GitHub actions. I wanted to try it.
- Quite a lot more experience with Bitbucket pipelines.

The CI/CD included in the repository comprises:

- Code, docker and helm quality checks after git push

- Image delivery to GHCR
- Helm chart build
- Delivery of the Helm chart
  - Disabled due to privacy concerns.

Experience notes

- We don't use docker-compose or minikube at work
- Personal experience using them in my server

The local deployment

For local deployment you have two options:

- Docker-compose
  - Quick
- Minikube
  - A bit more laborious
  - You need read access to my Github Docker registry
  - You need to use local helm chart

Both are described in the README.md

Experience notes

- This is where I feel confortable right now

Cloud deployment

I know the task was:

> *3. Write configuration scripts for building and no-downtime production deployment of this application, keeping in mind aspects that an SRE would have to consider.*

I have not produced deployment scritps because I think bigger. I know, it may sound presumptuous, but, bare with me.

In this and folliwing slides I will detail the deployment and tools I consided basic to manage a cloud deployment

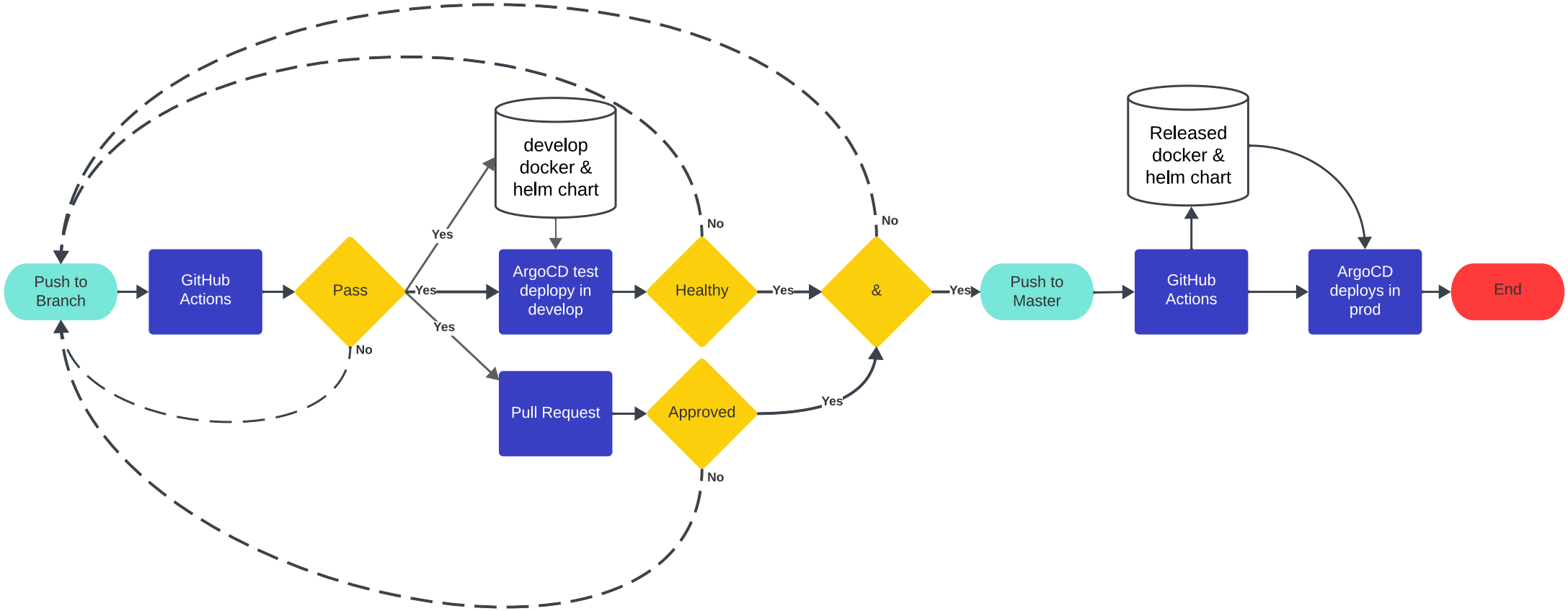
For no-downtime production deployment I would use several moving parts:

- Kubernetes pod disruption budget
  - With this set for a deployment you are guaranteed that K8S updates pods as you wish, keeping a minimum for the service to be alive.

- ArgoCD (I have included a sample application in the code)
  - Overseen all deployments among all clusters is ArgoCD (It also lives in a K8S cluster)
  - ArgoCD has the concept of App Health which is intertwine with K8S resources state and may be fine tuned even further
  - ArgoCD perform staged deployments among clusters, so, for instance, it can deploy to canary clusters and if it goes well continue to production.
  - ArgoCD does GitOps easy, a push to master may trigger an automatic deployment in develop clusters and a simple click in the Web UI will deploy to other clusters, like CI or production.

- Other tools for monitoring, alerting, notification, backups…

Deployment pipeline



Infrastructure deployment

It is imperative to deploy IaC so,

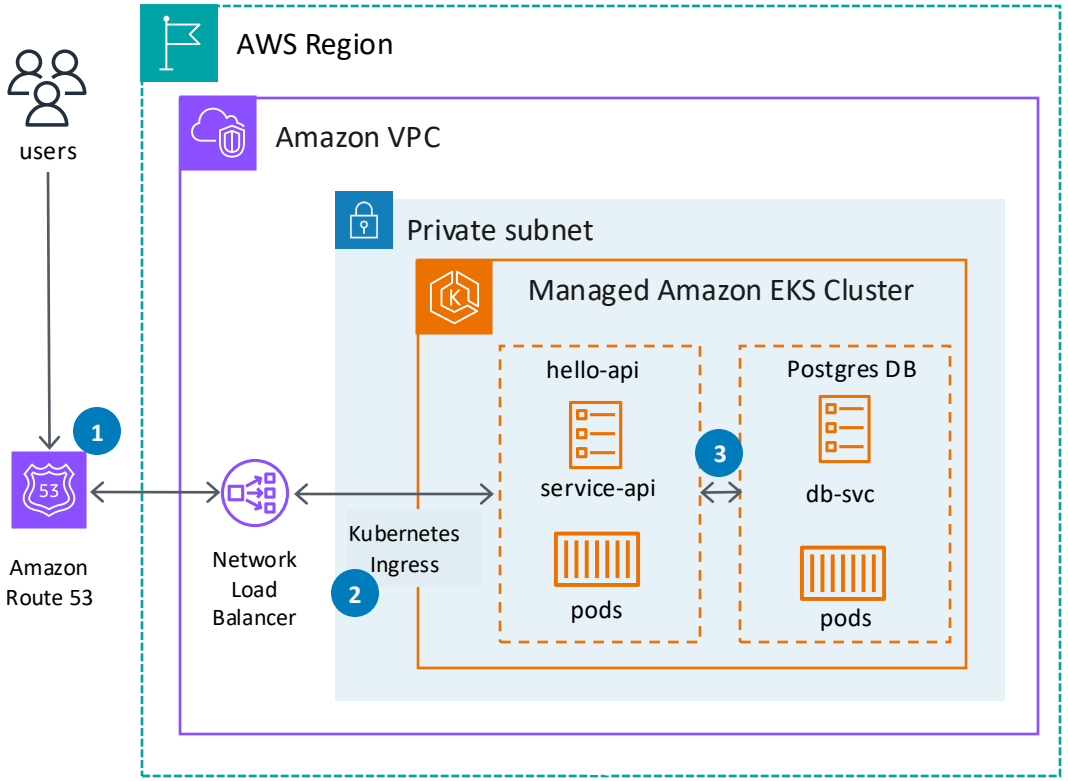
- To deploy the AWS resources except AWS EKS I would use Terraform with Terragrant.
  - Automate IaC with GitOps using Atlantis

- To deploy EKS clusters I would use Cluster API with Cluster API AWS Provider.
  - CAPI/CAPA allow a very powerfull integration with Argo taking IaC to the next level with GitOps

Experience notes

- Daily experience with all these tools deployment and configuration

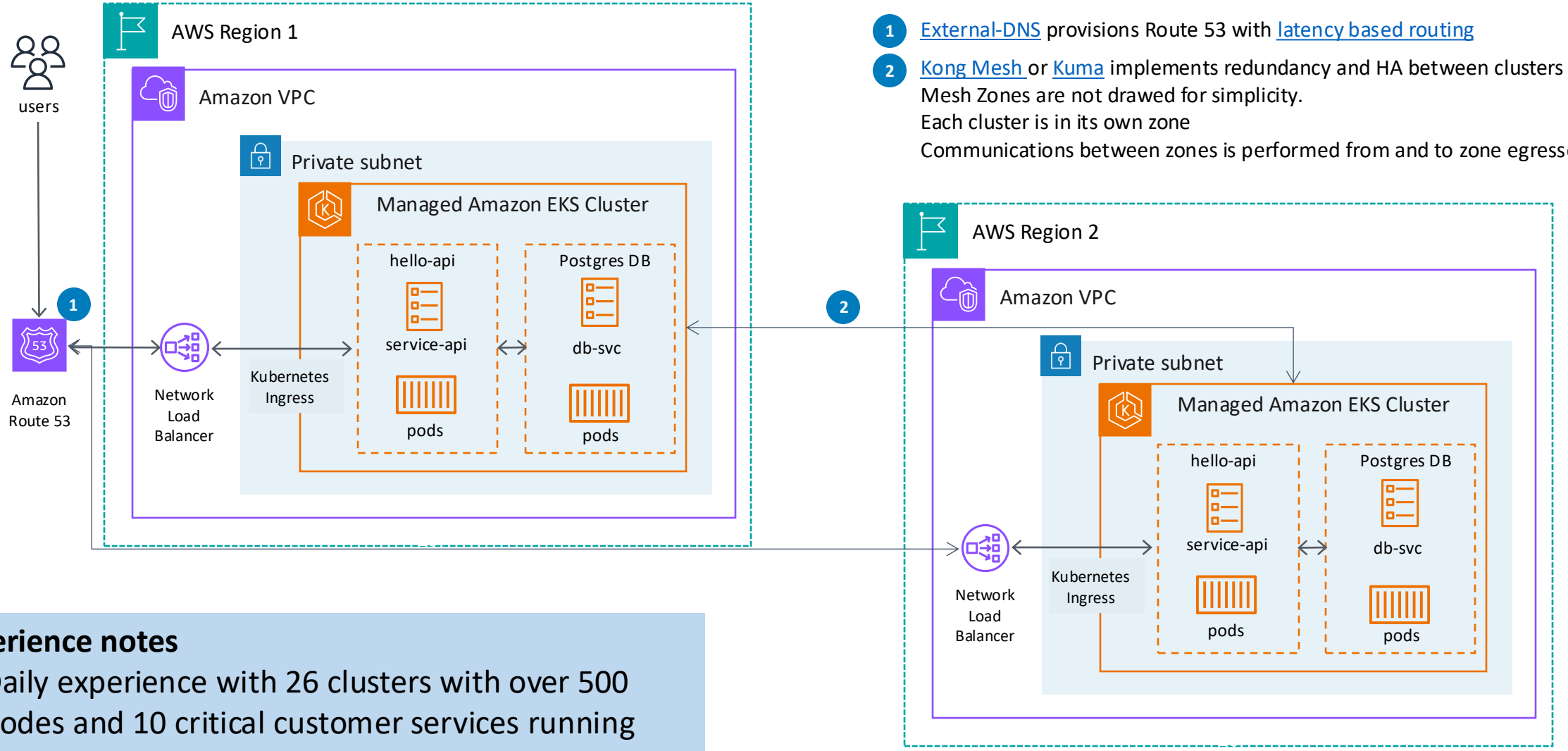
AWS EKS deployment



1. External-DNS takes care of Route53 domain provisioning
2. Kong Ingress Controller takes care of LoadBalancer provisioning and configuration
3. Kong Mesh or Kuma implements mTLS, auth, traffic monitoring… between pods

Based on Modernize Applications with Microservices Using Amazon EKS

AWS EKS deployment: with HA and redundancy



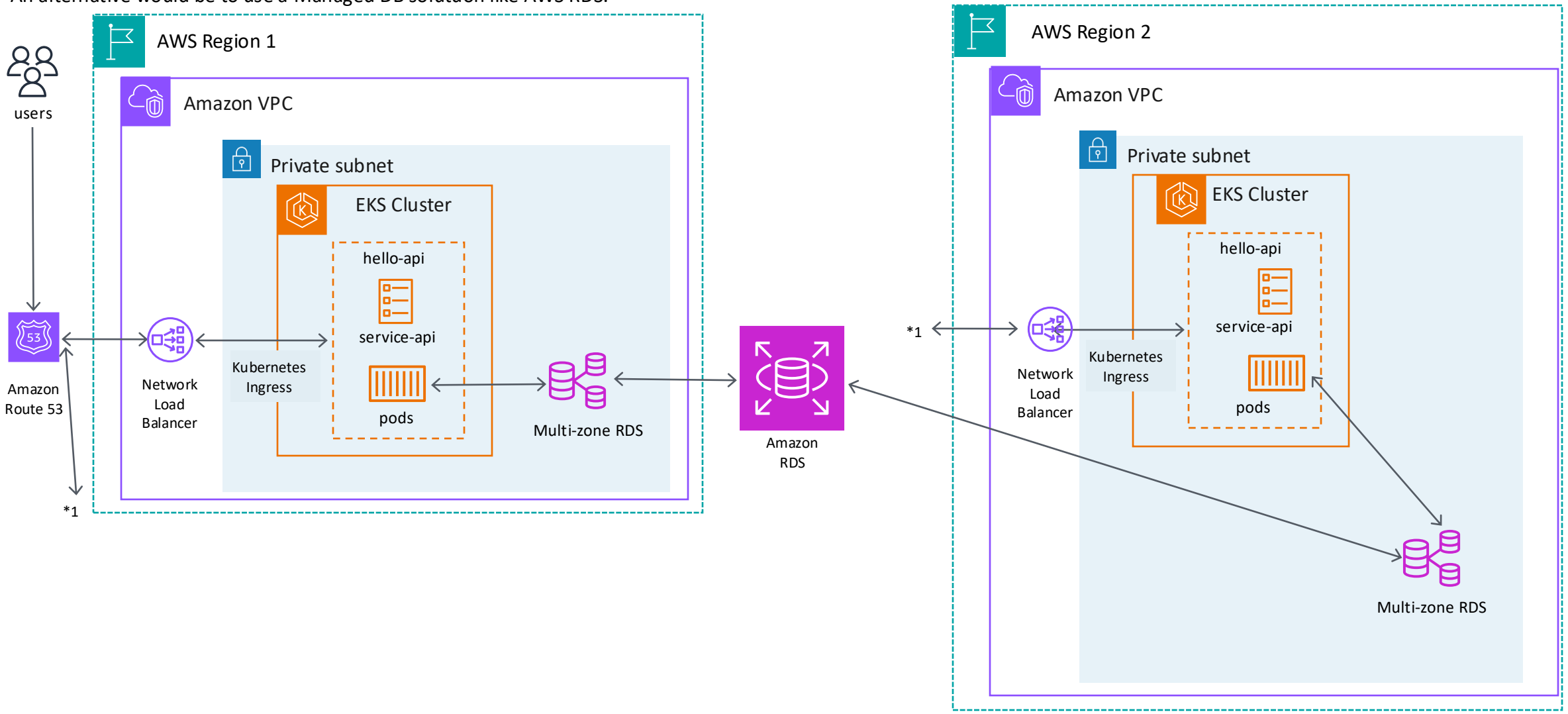
1. External-DNS provisions Route 53 with latency based routing
2. Kong Mesh or Kuma implements redundancy and HA between clusters
   Mesh Zones are not drawed for simplicity.
   Each cluster is in its own zone
   Communications between zones is performed from and to zone egresses and ingresses

Experience notes

- Daily experience with 26 clusters with over 500 nodes and 10 critical customer services running

AWS EKS deployment: alternative with RDS

I have assumed we want to use our own managed DB and choose PostgreSQL for local deployment and examples.
An alternative would be to use a Managed DB solutuon like AWS RDS.



Other tools

Monitoring:

- Prometheus for metrics collection (the code is already producind and exposing them)
- Local grafana deployment with Mimir or InfluxDB for long term storage
- GrafanaCloud

OR

- Datadog

Logs:

- Local Elastic Search with filebeat and logstash for logs storage

OR

- Grafana Loki

Backups:

- K8S backups Velero
- DB backups: I would use a cronjob to perform a pg_dump of the DB and a S3 Bucket for long term storage

OR

- Use AWS RDS and automate snapshots

Notifications:

- AlertManager and Slack/Email/Pagerduty

Experience notes

- Daily experience with all these tools, deployment and configuration