🐾 PetClinic Cloud Infrastructure | Team Argentina, Córdoba (ARG-CBA) – Bootcamp DevOps 2

Desarrollado por:

- 👨‍💻 Angel Mario Perez – DevOps Engineer Github
- 👨‍💻 Kevin Leonel Yanes – DevOps Engineer Github

📐 Arquitectura del Sistema

La infraestructura de **PetClinic** está diseñada bajo un enfoque **cloud-native**, modular y escalable, utilizando los siguientes componentes clave:

- **Orquestación**: Clúster de Kubernetes sobre **Amazon EKS** para gestionar microservicios.
- **Base de datos**: **MySQL** desplegado en pods dedicados, aislados por nodo para mantener performance y seguridad.
- **CI/CD**: Pipelines automatizados con **GitHub Actions** que ejecutan pruebas unitarias (JUnit), análisis estático (SonarQube), construcción (Maven) y publicación de imágenes en **AWS ECR**.
- **GitOps**: Sincronización continua del estado deseado con **ArgoCD**, desplegando automáticamente hacia el clúster EKS.
- **Observabilidad**: Stack completo de monitoreo y trazabilidad distribuida con **Prometheus**, **Grafana**, **Loki**, **Tempo** y **OpenTelemetry Collector**.
- **Red y Seguridad**: Subredes públicas y privadas en múltiples zonas de disponibilidad (us-east-1), siguiendo buenas prácticas de resiliencia y aislamiento.

⚙️ *Todos los recursos son provisionados con **Terraform**, usando módulos reutilizables y configuraciones por entorno.*

🚀 Guía de Despliegue

Requisitos Previos

- Cuenta (gratuita) de AWS configurada vía CLI. (https://docs.aws.amazon.com/cli/v1/userguide/cli-chap-welcome.html)
- Terraform >= 1.3.x

  (https://developer.hashicorp.com/terraform/install)

- kubectl + Kustomize instalados (https://kubernetes.io/docs/tasks/tools/)

  Es importante destacar que pueden experimentar este proyecto completo usando:

- Kind + LocalStack (Opcional para entorno local https://github.com/localstack/localstack)
- Minikube (Opcional https://minikube.sigs.k8s.io/docs/start)

1. Infraestructura Cloud (Terraform)

(Para copiar y pegar)

Clonar el repositorio
https://github.com/mperezgithub/fc-devops-infraestructure-automation.git

$ cd terraform

$ terraform init (Inicializa el terraform)

$ terraform plan (Muestra el plan de ejecucion)

$ terraform apply (Aplica los cambios solicitando confirmacion)

$ terraform destroy (Borra la infraestructura completa sin dejar rastros de lo aprovisionado)

Esto crea:

- VPC

- Subredes en dos zonas de disponibilidad a y b
- EIP - Elastic IP
- Nat Gateway
- EKS Cluster
- Node Groups
- Load Balancer Controller
- Roles y políticas necesarias
- Almacenamiento (EBS)

2. Despliegue de Aplicación y Observabilidad (Kustomize)

$ kubectl apply -f kustomize/overlays/dev/namespace.yaml (crea namespaces)

$ kubectl apply -k k8s/kustomize/overlays/dev (levanta toda los servicios necesarios)

Este comando despliega:

- PetClinic App
- MySQL
- Stack de monitoreo (Grafana, Prometheus, Loki, etc.)
- ArgoCD
- Configuraciones específicas de entorno

Opcionales. Crear clúster local para pruebas (localstack)

$ kind create cluster --config kind/cluster.yaml

$ docker-compose up -d --build

⚙️ Procedimientos Operativos

Acceso al clúster

$ aws eks update-kubeconfig --name <nombre-cluster> --region us-east-1

$ kubectl config use-context <Cluster ARN>

Ejemplo:

$ aws eks update-kubeconfig --region us-east-1 --name cf-devops-cbateam-cluster

$ kubectl config use-context
arn:aws:eks:us-east-1:739275443318:cluster/cf-devops-cbateam-cluster

Verificación del estado del clúster

$ kubectl get nodes

$ kubectl get pods -A

Ingreso a la aplicacion principal

- Revisar los ingress DNS asignados por AWS
  kubectl get ingress -A

- http://DNSdePetclinic

Observabilidad

- Revisar los ingress DNS asignados por AWS

  kubectl get ingress -A

  Ingreso a Grafana

- http://DNSdegrafana

- Usuario/Contraseña por defecto: admin/admin (se puede cambiar en el deployment inicial)

GitOPs

Ingreso a ArgoCD

- Revisar los ingress DNS asignados por AWS
  kubectl get ingress -A

- http://DNSdeargocd

Extraer admin password del Argocd

Para ingresar a la consola del argocd, el usuario es admin y la password se obtiene en dos pasos:

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

2-Luego usar un decode en base64 https://www.base64decode.org/ para decodear la clave que devuelve el paso previo y que esta encodeada en base64.

🧯 Runbooks de Incidentes

🐾 Incidente: La aplicación PetClinic no responde

1. **Verificar estado de los pods:**

   $ kubectl get pods -n petclinic

2. **Revisar logs:**

   $ kubectl logs <nombre-pod> -n petclinic

3. **Validar sincronización en ArgoCD:**

- Acceder a la UI de ArgoCD.
- Verificar si la app está en estado “Synced” y “Healthy”.
- Aplicar sync manual si es necesario.

📉 Incidente: No hay métricas en Grafana

1. **Verificar pods de monitoreo:**

   $ kubectl get pods -n monitoring

2. **Validar Prometheus:**

- Confirmar targets disponibles (/targets endpoint o UI).
- Revisar prometheus.yaml en el overlay kustomize/monitoring.

3. **Reiniciar pods afectados:**

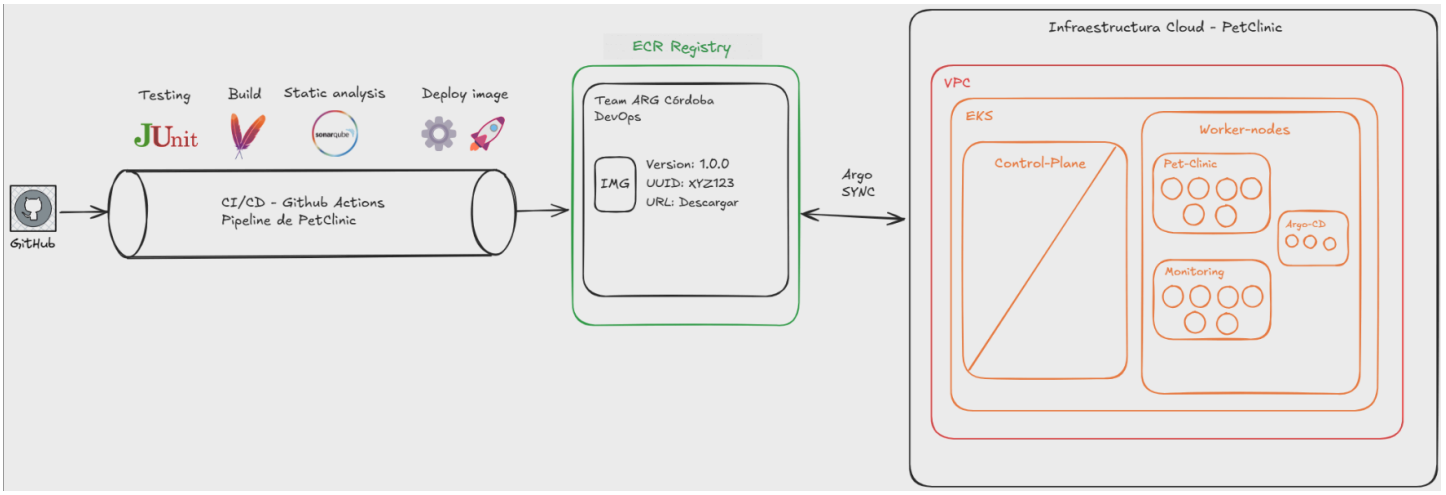
   $ kubectl delete pod <nombre-pod> -n monitoring

🧭 Diagramas de Arquitectura

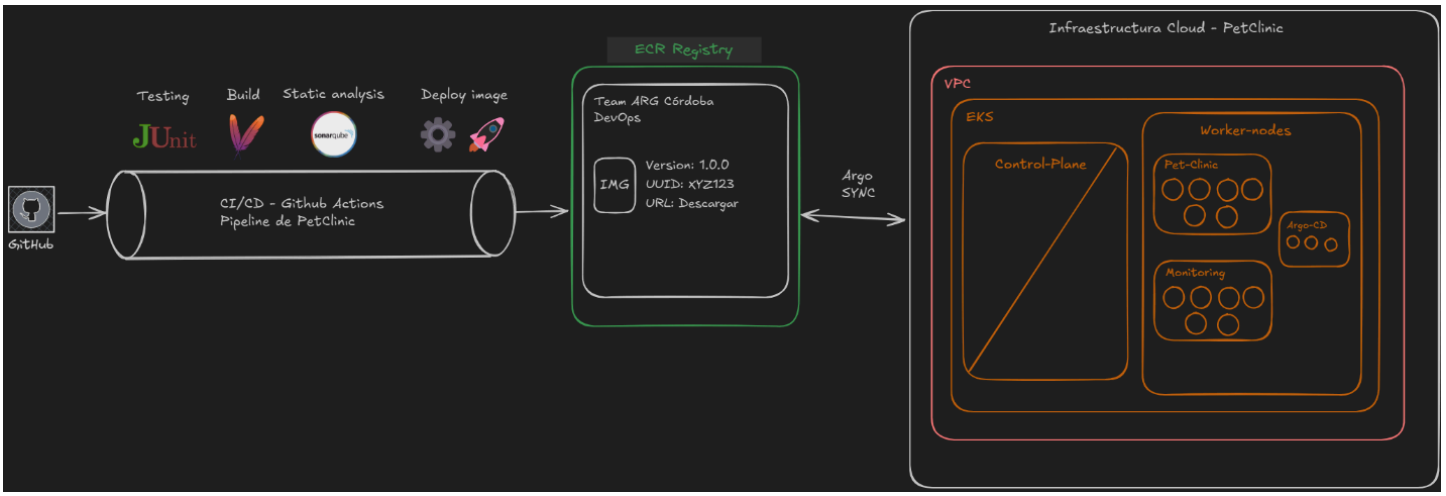
Los diagramas visuales que acompañan este proyecto incluyen:

Diagrama CI/CD

📄 Diagrama CI/CD (Fondo Blanco)



📄 Diagrama CI/CD (Fondo Oscuro)



Visualiza el pipeline completo desde GitHub Actions, publicación en ECR y sincronización con ArgoCD para despliegue en EKS.

1. GitHub + GitHub Actions Pipeline

📌 ¿Qué representa?

Un flujo de CI/CD automatizado que comienza desde GitHub.

🔍 Detalles:

- **Repositorio GitHub**: Punto central de colaboración y versionado de código. Aquí reside el código fuente de la app PetClinic y las configuraciones.
- **CI/CD con GitHub Actions**: Pipeline automatizado que ejecuta una serie de etapas:

Etapa	Herramienta	Función
🧪 Testing	JUnit	Ejecuta pruebas unitarias para validar el comportamiento del código.
⚙️ Build	Maven	Compila y empaqueta la aplicación en formato .jar o .war.
🔍 Static Analysis	SonarQube	Evalúa la calidad del código y detecta posibles bugs o vulnerabilidades.
🚀 Deploy Image	Docker + AWS CLI	Construye la imagen Docker y la publica en el ECR (Elastic Container Registry).

Este pipeline garantiza que cada cambio en el repositorio es **validado, probado y empaquetado** automáticamente.

---

2. ECR Registry (AWS Elastic Container Registry)

📌 ¿Qué representa?

Un repositorio privado donde se almacenan las imágenes Docker ya construidas.

🔍 Detalles:

- **Imagen Versionada**: Se muestra una imagen IMG etiquetada con:
    - **Version**: 1.0.0
    - **UUID**: XYZ123 (identificador único de imagen)

    - **URL**: Enlace para descargar o usar la imagen

- **Uso del Registro**:
    - La imagen Docker generada es almacenada aquí.
    - ArgoCD tomará esta imagen desde el ECR para desplegarla automáticamente en el clúster EKS.

Este paso garantiza **trazabilidad, versionado y disponibilidad segura** de las imágenes.

---

3. Sincronización ArgoCD (GitOps)

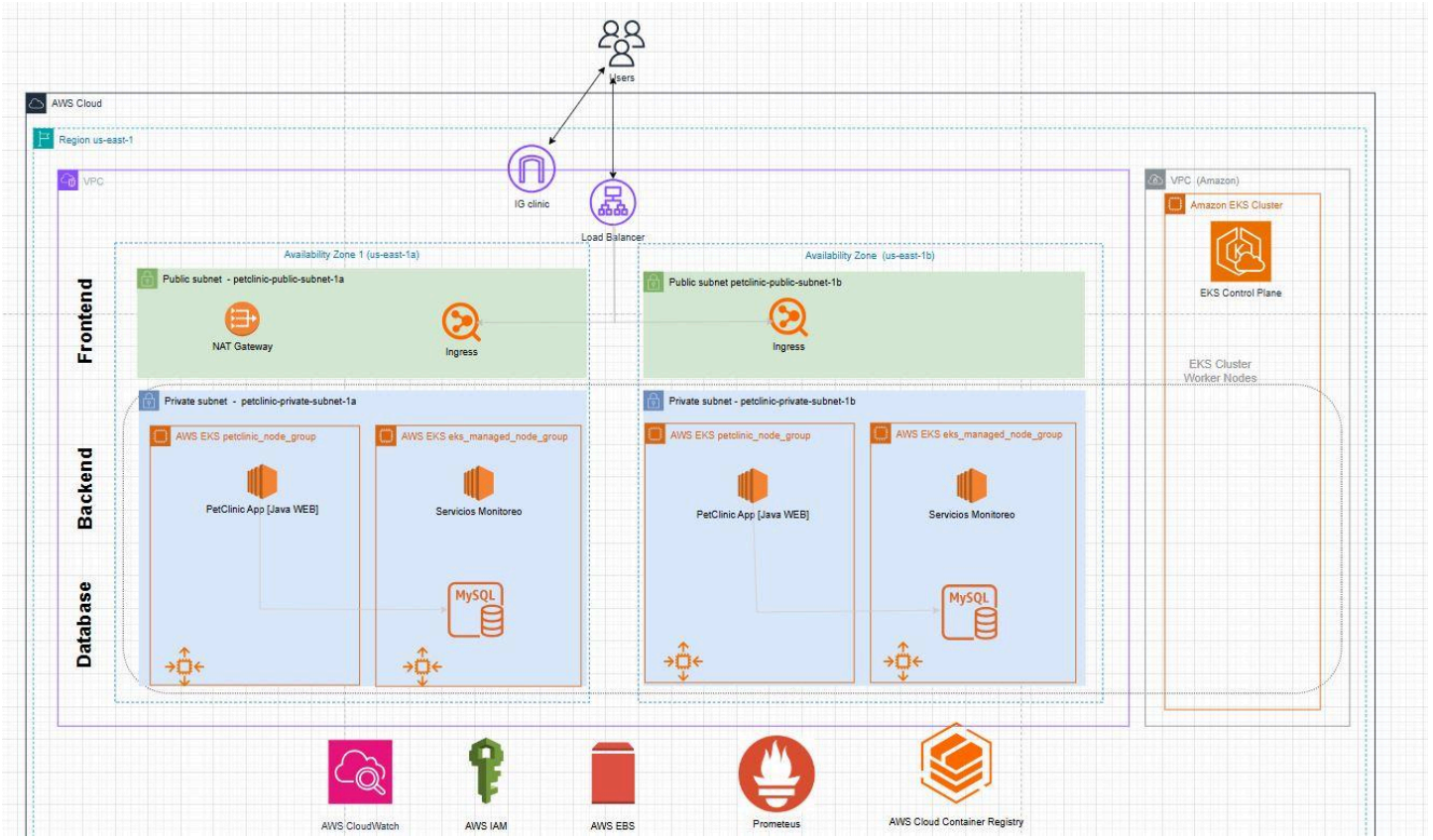
📌 ¿Qué representa?

Un flujo de GitOps mediante **ArgoCD**, que observa continuamente el repositorio o la imagen para sincronizar el estado deseado en el clúster.

🔍 Detalles:

- Cuando se detecta una nueva imagen en el ECR (o un cambio en los manifiestos K8s), **ArgoCD ejecuta la sincronización**.
- ArgoCD compara el estado real del clúster con el estado definido en Git, y **aplica los cambios necesarios automáticamente**.
- Esto asegura que el entorno Kubernetes **refleje siempre el estado deseado** del repositorio y CI/CD.

Diagrama de Infraestructura Cloud



Incluye la disposición de subredes, AZs, nodos EKS, flujo de datos, acceso a servicios, y componentes del stack de monitoreo.

🌐 1. Entorno General y Alcance

Este diagrama representa la arquitectura desplegada en la **nube de AWS**, en la región **us-east-1**, utilizando un enfoque de **alta disponibilidad (HA)** basado en **dos zonas de disponibilidad (AZs)**: us-east-1a y us-east-1b.

Toda la infraestructura está contenida dentro de una **VPC personalizada** con subredes públicas y privadas, y gestionada por un clúster de **Amazon EKS** (Elastic Kubernetes Service).

📊 2. División de Capas Funcionales

El diagrama está dividido verticalmente en tres capas o dominios funcionales:

Capa	Función
Frontend	Acceso desde el exterior (usuarios, balanceador, ingress).
Backend	Aplicación PetClinic y servicios de monitoreo.
Database	Persistencia de datos (MySQL desplegado en pods).

🏞️ 3. Subredes y Zonas de Disponibilidad

- 🔹 **Zona de Disponibilidad 1 (us-east-1a)**
  - **Subred Pública:**
    - petclinic-public-subnet-1a
    - Contiene:
      - 🔁 **NAT Gateway**: Permite a las instancias en subredes privadas salir a internet.
      - 🌐 **Ingress Controller**: Controla y enruta tráfico externo a servicios internos.
  - **Subred Privada:**
    - petclinic-private-subnet-1a
    - Contiene:
      - 🐾 **PetClinic App (Java WEB)** en un grupo de nodos EKS.
      - 📊 **Servicios de Monitoreo** (Grafana, Prometheus, Loki, etc.).
      - 🗄️ **Base de Datos MySQL** como pod de Kubernetes.

🔹 **Zona de Disponibilidad 2 (us-east-1b)**

- **Subred Pública:**
  - petclinic-public-subnet-1b
  - Contiene:
    - 🌐 **Ingress Controller** redundante para HA.
- **Subred Privada:**
  - petclinic-private-subnet-1b
  - Contiene la misma estructura que la zona 1:
    - 🐾 PetClinic App
    - 📊 Servicios de Monitoreo
    - 🗄️ MySQL

✅ Este diseño multi-AZ garantiza tolerancia a fallos, escalabilidad y balanceo de carga en caso de incidentes.

🧠 4. Amazon EKS: Control Plane y Worker Nodes

- **EKS Control Plane**: Gestionado por AWS (no visible directamente), orquesta todo el clúster, ejecutando el API server, scheduler y etcd.
- **Worker Nodes (Node Groups)**:
  - Grupos de nodos administrados (managed node groups) divididos por funciones:
    - **App Nodes**: Ejecutan los contenedores de PetClinic.
    - **Monitoring Nodes**: Ejecutan los contenedores de Grafana, Prometheus, Tempo, Loki, etc.
    - **DB Pods**: Ejecutan MySQL dentro del clúster (no servicios externos).

🌉 5. Balanceador e Ingress

- **Load Balancer** (en la capa frontal):
    - Reparte el tráfico HTTP/HTTPS entrante entre los ingress controllers.
    - Controlado por Kubernetes mediante Ingress objects y ALB Controller.
- **Ingress Controllers**:
    - Deploys en las subredes públicas.
    - Enrutamiento del tráfico a los servicios expuestos en los namespaces correspondientes.

👥 6. Interacción con Usuarios

- Los usuarios acceden a través del DNS o IP pública del balanceador.
- El tráfico es recibido por el **ingress**, que lo redirige a los servicios de frontend (UI de PetClinic).

🧰 7. Servicios de Soporte (parte inferior del diagrama)

Servicio	Función
AWS CloudWatch	Observabilidad y métricas de infraestructura (nivel AWS).
AWS IAM	Gestión de permisos para recursos y servicios.
AWS EBS	Volúmenes de almacenamiento persistente para los pods de MySQL.
Prometheus	Recolección de métricas a nivel aplicación y clúster.
AWS Container	Almacenamiento de imágenes Docker para

Registry (ECR)	despliegue.

🐾 Conclusión 🐾

El proyecto **PetClinic Cloud Infrastructure**, desarrollado por el equipo **Team Córdoba (ARG)**, representa una implementación moderna y completa del enfoque **DevOps**, integrando herramientas de infraestructura como código, despliegue automatizado, monitoreo y GitOps en un entorno real en la nube.

A través de este trabajo logramos:

- **Automatizar desde el código hasta la producción** utilizando **GitHub Actions** como motor CI/CD para testing, análisis, construcción y despliegue de imágenes Docker hacia **Amazon ECR**.
- Desplegar la aplicación PetClinic en un entorno **Kubernetes gestionado por EKS**, utilizando **Terraform** para asegurar reproducibilidad y consistencia.
- Aplicar un enfoque **GitOps con ArgoCD**, logrando que los cambios en el repositorio se reflejen automáticamente en un entorno pre-productivo (también con enfoque a un entorno productivo) mediante sincronización declarativa.
- Establecer una **infraestructura distribuida, segura y de alta disponibilidad**, aprovechando múltiples zonas de disponibilidad y una arquitectura en subredes públicas y privadas.
- Incorporar un completo stack de **observabilidad**, compuesto por **Prometheus**, **Grafana**, **Loki**, **Tempo** y **OpenTelemetry**, permitiendo visibilidad total del comportamiento de la aplicación y del entorno.
- Validar procesos de despliegue local con **Kind y LocalStack**, favoreciendo entornos de desarrollo consistentes y pruebas previas sin consumir recursos en la nube.

Además, se documentaron de manera estructurada:

- La arquitectura técnica de la solución, con diagramas claros.
- La guía paso a paso de despliegue y operación.
- Runbooks para la resolución de incidentes comunes.

Este proyecto no solo consolidó conocimientos técnicos avanzados en herramientas clave del ecosistema DevOps, sino que también reforzó habilidades colaborativas, metodologías de trabajo ágil, y buenas prácticas de documentación profesional.

La infraestructura y el pipeline resultantes están preparados para escalar, adaptarse a cambios y facilitar el mantenimiento a largo plazo, posicionando esta solución como un ejemplo sólido y replicable para escenarios reales en la industria.

🔙 Gracias a CódigoFacilito por el espacio de aprendizaje y al equipo de mentores por su acompañamiento técnico durante todo el proceso.

Autores:

- 👨‍💻 *Angel Mario Perez – DevOps Engineer* Github
- 👨‍💻 *Kevin Leonel Yanes – DevOps Engineer* Github