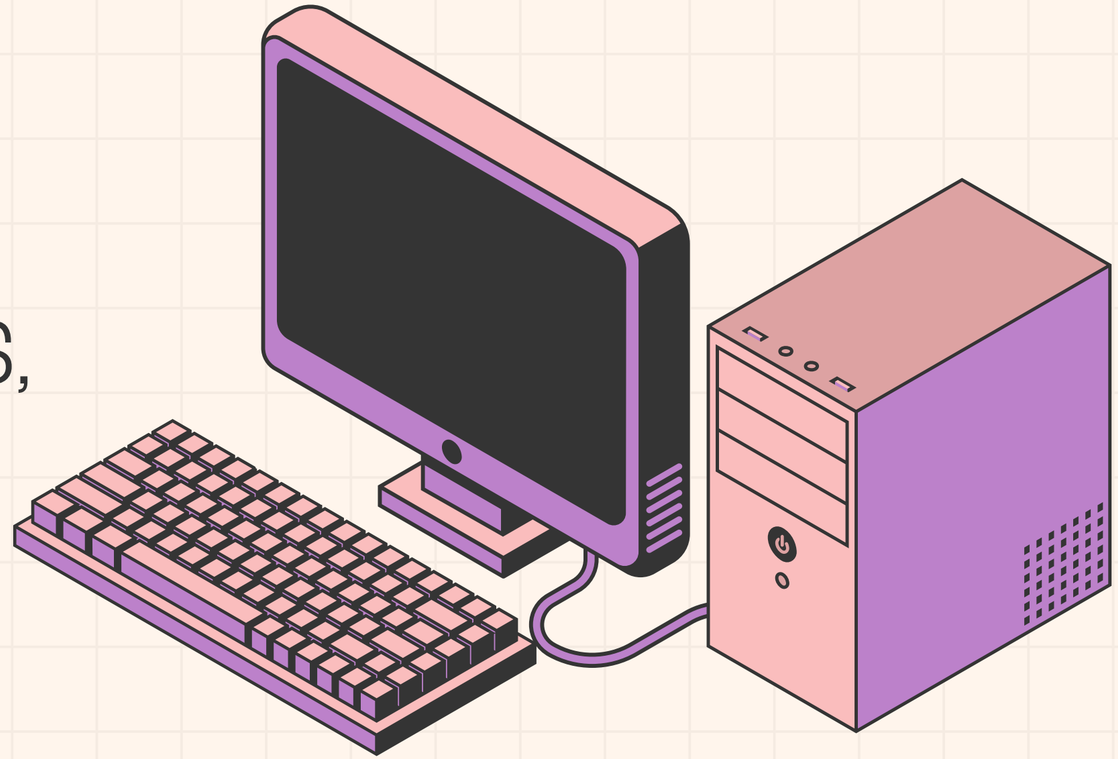
ACTIVIDAD 2

DEL CÓDIGO A LA PRODUCCIÓN: INFRAESTRUCTURA, CONTENEDORES, DESPLIEGUE Y OBSERVABILIDAD



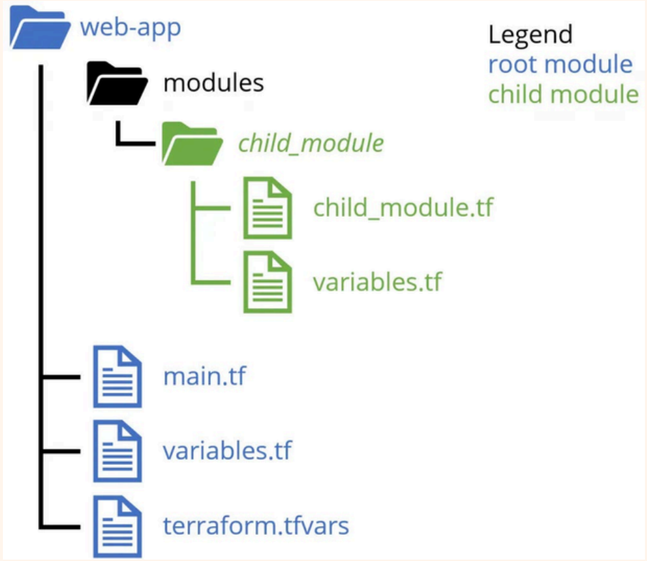
Integrantes:

- Chowdhury Gomez, Junal Johir
- La Torre Vasquez, Andres Sebastian
- Zapata Inga, Janio Adolfo

A. INFRAESTRUCTURA COMO CODIGO

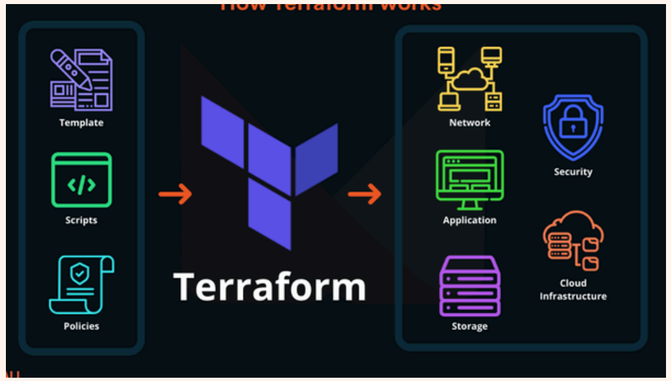
TAREA TEORICA:

- INVESTIGAR UNA HERRAMIENTA DE IAC (P. EJ. TERRAFORM) Y DESCRIBIR COMO ORGANIZA SUS MODULOS.
- PROPONER LA ESTRUCTURA DE ARCHIVOS Y DIRECTORIOS PARA UN PROYECTO HIPOTETICO QUE INCLUYA TRES MODULOS: NETWORK, DATABASE Y APPLICATION. JUSTIFICAR LA JERARQUIA ELEGIDA.



TERRAFORM ORGANIZA SU IAC MEDIANTE MODULOS , QUE SON UNIDADES REUTILIZABLES DE CONFIGURACION

- UN EJEMPLO DE MODULOS ES:
  - MODULO RAIZ: EL PUNTO DE ENTRADA PRINCIPAL DE TERRAFORM
  - MODULOS LOCALES: SE ENCUENTRAN DENTRO DEL MISMO REPOSITORIO DEL PROYECTO
  - MODULOS REMOTOS: GUARDADOS EN REPOSITORIOS COMO GITHUB, TERRAFORM REGISTRY O SISTEMAS DE ALMACENAMIENTO REMOTO



ESTRUCTURA DE ARCHIVOS Y DIRECTORIOS PARA UN PROYECTO

- UNA RED (NETWORK)) CON UNA VPC Y SUBREDES
- UNA BASE DE DATOS (DATABASE) EN POSTGRESQL EN LA NUBE POR EJEMPLO
- UNA APLICACION (APPLICATION) QUE SE EJECUTA EN INSTANCIAS DE COMPUTACION

```
C:\Users\Junal\Desktop\25-1\CC3S2 Desarrollo de Software\Actividades-CC3S2\Actividad-02\resources\terraform-project>tree /F /A
Listado de rutas de carpetas para el volumen OS
El número de serie del volumen es 4283-D7F0
C:.
|   backend.tf
|   main.tf
|   outputs.tf
|   provider.tf
|   README.md
|   terraform.tfvars
|   variables.tf
|
\---modules
    +---application
    |       main.tf
    |       outputs.tf
    |       README.md
    |       variables.tf
    |
    +---database
    |       main.tf
    |       outputs.tf
    |       README.md
    |       variables.tf
    |
    \---network
            main.tf
            outputs.tf
            README.md
            variables.tf
```

LOS \MODULES CONTIENE MODULOS (NETWORK, DATABASE, APPLICATION) LOS MODULOS CONTIENE:

- **MAIN.TF:** CONFIGURACION PRINCIPAL
- **VARIABLES.TF**: PARAMETROS CONFIGURABLES
- OUTPUTS.TF: VALORES EXPORTABLES PARA OTROS MODULOS
- **README.MD**: DOCUMENTACIÓN DEL MÓDULO

- el archivo `main.tf` en la raiz es el que define el uso de los modulos

```
module "network" {
    source = "./modules/network"
    vpc_name = "mi-vpc"
}

module "database" {
    source = "./modules/database"
    db_name = "mi-database"
}

module "application" {
    source = "./modules/application"
    app_name = "mi-app"
}
```

  - el archivo `provider.tf` define el provedor de nube, puede ser AWS, GCP, Azure
  - `terraform.tfvars` contiene variables pesrsonalizadas
  - `backend.tf` es opcional en el caso de que se utilice almacenamiento remoto para el estado

- LOS PRINCIPALES BENEFICIOS LA ESTRUCTURA:
- **MODULARIDAD**: YA QUE PERMITE REUTILIZAR COMPONENTES Y SEPARARLOS DE LA CONFIGURACION PRINCIPAL
- **ESCALABILIDAD**: ES FACIL AGREGAR MAS MODULOS SI EL PROYECTO CRECE
- **MANTENIBILIDAD**: SEPARAR LOGICA FACILITA LA DEPURACION Y COLABORACION

B. CONTENERIZACION Y DESPLIEGUE DE APLICACIONES MODERNAS

TAREA TEORICA:

DESCRIBIR UN FLUJO SIMPLE DE DESPLIEGUE DONDE UN DESARROLLADOR HACE UN CAMBIO EN EL CODIGO, SE CONSTRUYE UNA NUEVA IMAGEN DOCKER Y SE ACTUALIZA UN DEPLOYMENT DE KUBERNETES.

EXPLICAR LAS VENTAJAS DE USAR KUBERNETES PARA ESCALAR UNA APLICACION EN UN EVENTO DE ALTO TRAFICO.

1. UN DESSARROLLADOR HACE UN CAMBIO EN EL CODIGO DEL PROYECTO
   - AGREGA CAMBIOS AL STAGE

     GIT ADD .
   - CONFIRMA CAMBIOS

     GIT COMMIT -M "ACTUALIZACION DEL CODIGO"
   - ENVIA CAMBIOS

     GIT PUSH ORIGIN MAIN

2. SE CONSTRUYE UNA IMAGEN NUEVA DE DOCKER
   - CONTRUYE IMAGEN

     DOCKER BUILD -T MI-APP:V2 .
   - SE VERIFICA LA IMAGEN

     DOCKER IMAGES

3. SUBE LA IMAGEN A DOCKERHUB

   DOCKER PUSH MI-REPO/MI-APLICACION:V2

4. ACTUALIZACION DEPLOYMENT DE KUBERNETES

SE MODIFICA EL ***DEPLOYMENT.YAML*** CON LA NUEVA ACTUALIZACION DE VERSION DE LA IMAGEN

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: mi-aplicacion
spec:
    replicas: 3
    selector:
        matchLabels:
        app: mi-app
template:
    metadata:
        labels:
            app: mi-app
    spec:
        containers:
            - name: mi-contenedor
              image: mi-repo/mi-aplicacion:v2  # <--aqui se actualiz la version
              ports:
              - containerPort: 8080
```

- APLICAMOS EL CAMBIO EN KUBERENTES
  KUBECTL APPLY -F DEPLOYMENT.YAML
- COMPROBAR EL DESPLIEGUE
  KUBECTL GET PODS
  KUBECTL DESCRIBE DEPLOYMENT MI-APLICACION

VENTAJAS DE USAR KUBERNETES PARA ESCALAR UNA APLICACION EN UN EVENTO DE ALTO TRAFICO

- **AUTOESCALADO HORIZONTAL:** KUBERNETES PUEDE AUMENTAR O DISMINUIR LA CANTIDAD DE PODS SEGNN LA CARGA
- **BALANCEO DE CARGA:** KUBERNETES USA UN SERVICE PARA DISTRIBUIR EL TRAFICO ENTRE LOS PODS DISPONIBLES Y EVITA QUE UN SOLO SERVIDOR COLAPSE
- **TOLERANCIA A FALLOS:** SI UN POD FALLA, KUBERNETES LO REINICIA AUTOMATICAMENTE Y GARANTIZA ALTA DISPONIBILIDAD
- **EFICIENCIA EN COSTOS:** KUBERNETES PUEDE REDUCIR LA CANTIDAD DE PODS CUANDO EL TRAFICO BAJA, OPTIMIZANDO RECURSOS Y COSTOS

C. OBSERVABILIDAD Y TROUBLESHOOTING

TAREA TEORICA:

- INVESTIGAR Y DESCRIBIR COMO PROMETHEUS Y GRAFANA SE INTEGRAN CON KUBERNETES PARA MONITOREAR LOS CONTENEDORES Y EL CLUSTER.
- PROPONER UN SET DE METRICAS Y ALERTAS MINIMAS PARA UNA APLICACION WEB (POR EJEMPLO, LATENCIA DE PETICIONES, USO DE CPU/MEMORIA, TASA DE ERRORES).

INTEGRACION DE PROMETHEUS Y GRAFANA CON KUBERNETES

PROMETHEUS Y GRAFANA SE USAN PARA MONITOREAR CONTENEDORES Y CLUSTERS DE KUBERNETES, OBTENIENDO METRICAS EN TIEMPO REAL SOBRE EL USO DE RECURSOS, DISPONIBILIDAD Y ERRORES

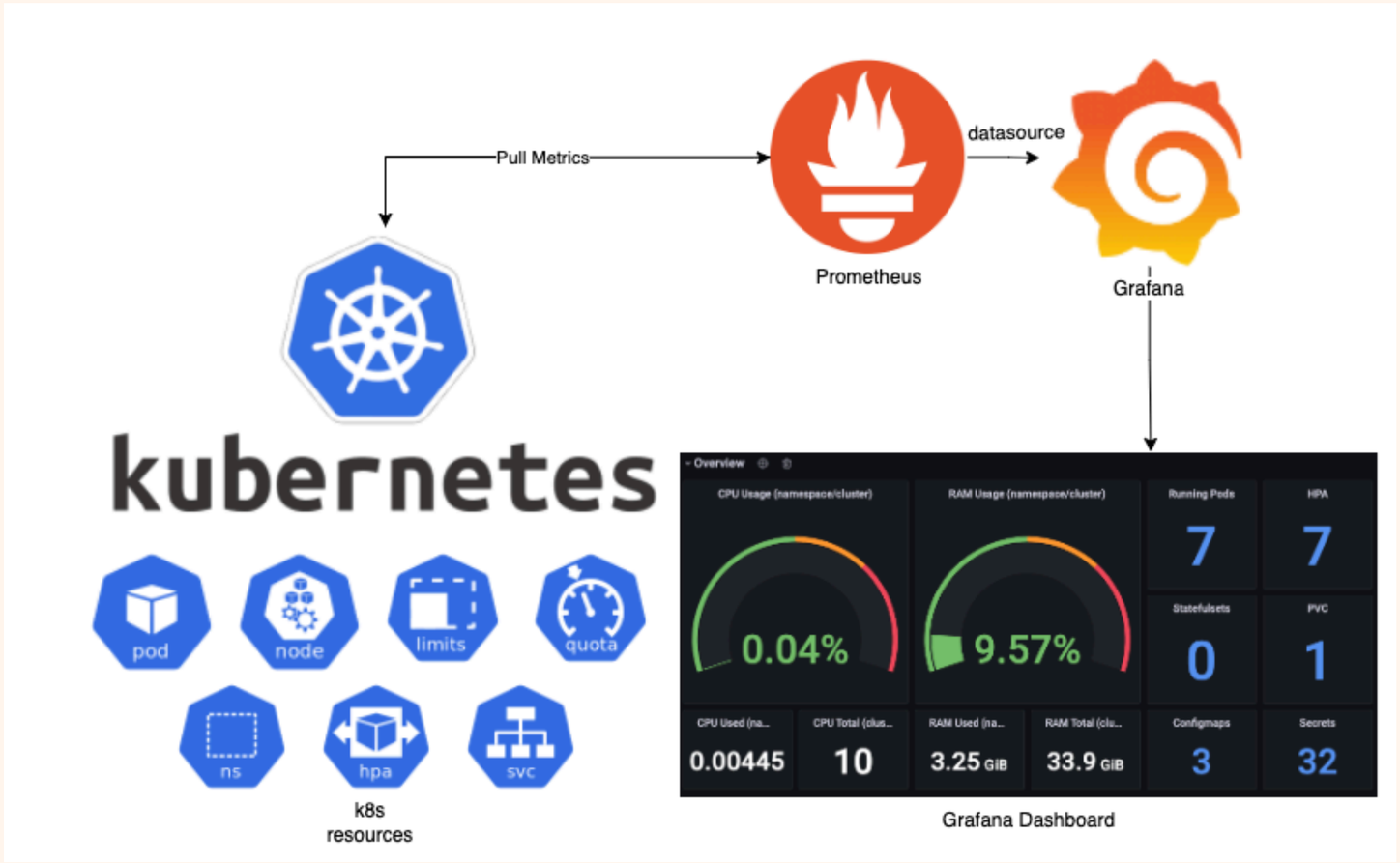
INSTALACION DE PROMETHEUS Y GRAFANA EN KUBERNETES

LA FORMA MAS COMUN ES USANDO HELM EL CUAL ES UN GESTOR PAQUETES PARA KUBERNETES.

HELM REPO ADD PROMETHEUS-COMMUNITY HTTPS://PROMETHEUS-COMMUNITY.GITHUB.IO/HELM-CHARTS

ACTUALIZA REPOSITORIO
HELM REPO UPDATE

INSTALA PROMETHEUS Y GRAFANA
HELM INSTALL MONITORING PROMETHEUS-COMMUNITY/KUBE-PROMETHEUS-STACK --NAMESPACE MONITORING --CREATE-NAMESPACE



FUNCIONAMIENTO

- **PROMETHEUS** SE COMUNICA CON KUBERNETES A TRAVES DE SERVICE MONITORS QUE RECOLECTAN METRICAS DESDE NODOS, PODS Y APLICACIONES
- **GRAFANA** USA PROMETHEUS COMO FUENTE DE DATOS PARA VISUALIZAR METRICAS EN PANELES PERSONALIZADOS
- **ALERTMANAGER (DE PROMETHEUS)** ENVIA NOTIFICACIONES CUANDO SE DETECTAN PROBLEMAS COMO POR EJEMPLO UN ALTO USO DE CPU

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
    name: mi-app-monitor
    labels:
        release: prometheus
spec:
    selector:
        matchLabels:
            app: mi-app
    endpoints:
        - port: http
            path: /metrics
            interval: 10s
```

EN EL CODIGO EL SERVICEMONITOR ORDENA A PROMETHEUS QUE RECOLECTE LAS METRICAS DESDE EL SERVICE MY-APPP, Y SE CONSULTA LA RUTA METRICS/ CADA 10 SEG

Metricas	Descripcion
latencia HTTP	Tiempo de respuesta de la API (p99, p95, promedio)
Tasa de errores HTTP	Porcentaje de respuestas 5xx y 4xx
uso de CPU y memoria	Consumos por pod y por nodo
Numero de peticiones	Total de requests por segundo
numero de pods activos	Cantidad de replicas funcionando

ALERTA POR ALTA LATENCIA EN LA API (> 2S EN P99)

```yaml
groups:
- name: api-latency
  rules:
  - alert: HighAPILatency
    expr: histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m])) > 2
    for: 2m
    labels:
      severity: critical
    annotations:
      summary: "alta latencia en la API (p99 > 2s)"
```

ALERTA POR NUMERO DE ERRORES HTTP 5XX ALTO (> 5% DE LAS PETICIONES)

```yaml
groups:
- name: http-errors
  rules:
  - alert: HighHttp5xxErrors
    expr: (sum(rate(http_requests_total{status=~"5.."}[5m])) / sum(rate(http_requests_total[5m]))) > 0.05
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "mas del 5% de las solicitudes estan fallando con errores 5xx"
```

kubernetes Grafana Prometheus

D. CI/CD (INTEGRACION CONTINUA / DESPLIEGUE CONTINUO)

TAREA TEORICA:

- EXPLICAR LA DIFERENCIA ENTRE ENTREGA CONTINUA (CONTINUOUS DELIVERY) Y DESPLIEGUE CONTINUO (CONTINUOUS DEPLOYMENT).

ENTREGA CONTINUA (CD - CONTINUOUS DELIVERY) ES LA PRÁCTICA DE ASEGURARSE DE QUE EL CÓDIGO ESTÉ SIEMPRE EN UN ESTADO LISTO PARA PRODUCCIÓN. CADA CAMBIO EN EL CÓDIGO PASA POR UNA SERIE DE PRUEBAS Y REVISIONES, Y UNA VEZ APROBADO, SE DEJA PREPARADO PARA SER DESPLEGADO MANUALMENTE CUANDO EL EQUIPO LO DECIDA.

DESPLIEGUE CONTINUO (CD - CONTINUOUS DEPLOYMENT) ES LA EVOLUCIÓN DEL PROCESO DE ENTREGA CONTINUA, DONDE CADA CAMBIO APROBADO SE DESPLIEGA AUTOMÁTICAMENTE EN PRODUCCIÓN SIN INTERVENCIÓN MANUAL. SOLO SE DETENDRÍA EN CASO DE QUE ALGUNA PRUEBA FALLE.

Concepto	Entrega Continua (Continuous Delivery)	Despliegue Continuo (Continuous Deployment)
¿Se automatiza el proceso hasta producción?	No, requiere aprobación manual.	Sí, todo el proceso es automático.
¿Cuándo se libera el software?	Cuando el equipo lo decide.	Inmediatamente después de pasar las pruebas.
Ejemplo de uso	Aplicaciones críticas (banca, salud).	Aplicaciones web con despliegue frecuente.

- DESCRIBIR LA RELEVANCIA DE IMPLEMENTAR PRUEBAS AUTOMÁTICAS (UNITARIAS, DE INTEGRACIÓN, DE SEGURIDAD) DENTRO DEL PIPELINE.

IMPLEMENTAR PRUEBAS AUTOMÁTICAS EN EL PIPELINE CI/CD ES FUNDAMENTAL PARA GARANTIZAR QUE CADA CAMBIO EN EL CÓDIGO SEA ESTABLE, SEGURO Y FUNCIONAL ANTES DE LLEGAR A PRODUCCIÓN.

- DETECCIÓN TEMPRANA DE ERRORES → EVITA QUE FALLOS LLEGUEN A PRODUCCIÓN.
- MAYOR VELOCIDAD DE DESARROLLO → SE PRUEBAN CAMBIOS RÁPIDAMENTE Y SIN INTERVENCIÓN MANUAL.
- REDUCCIÓN DE COSTOS → CORREGIR ERRORES EN DESARROLLO ES MÁS BARATO QUE EN PRODUCCIÓN.
- MEJOR CALIDAD DEL SOFTWARE → GARANTIZA QUE LAS FUNCIONALIDADES SIGAN FUNCIONANDO TRAS CAMBIOS.
- SEGURIDAD DESDE EL INICIO → DETECTA VULNERABILIDADES ANTES DEL DESPLIEGUE.

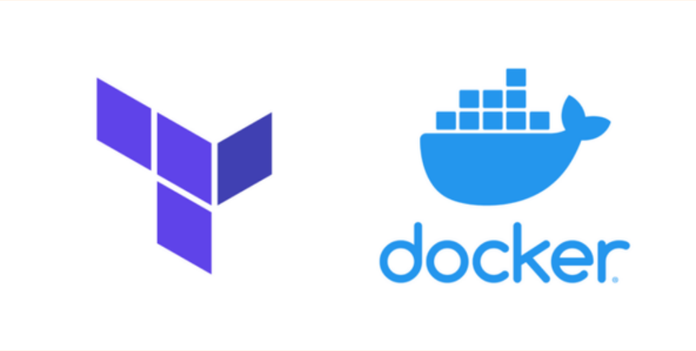
4. EVALUACION Y DISCUSION FINAL

IMPORTANCIA DE IAC, CONTENEDORES, KUBERNETES, OBSERVABILIDAD Y CI/CD PARA LA ENTREGA AGIL Y CONFIABLE DE SOFTWARE

AUTOMATIZACION DE LA INFRAESTRUCTURA Y EL DESPLIEGUE

UNO DE LOS PRINCIPALES BENEFICIOS DE LA COMBINACION DE IAC , CONTENEDORES , KUBERNETES, OBSERVABILIDAD Y CI/CD ES LA AUTOMATIZACION COMPLETA DEL CICLO DE VIDA DEL SOFTWARE

- IAC PERMITE DEFINIR LA INFRAESTRUCTURA COMO CODIGO , ASEGURANDO QUE LOS ENTORNOS SEAN REPRODUCIBLES Y ESCALABLES SIN INTERVENCION MANUAL
- CONTENEDORES FACILITAN LA PORTABBILIDAD DE LAS APLICACIONE , ASEGURANDO QUE SE EJECUTEN DE LA MISMA MANERA EN CUALQUIER ENTORNO
- KUBERNETES SE ENCARGA DE GESTIONAR Y ESCALAR LOS CONTENEDORES AUTOMATICAMENTE
- CI/CD AUTOMATIZA LAS PRUEBAS Y DESPLIEGUES , REDUCIENDO ERRORES HUMANOS Y ACELERANDO EL TIEMPO DE ENTREGA

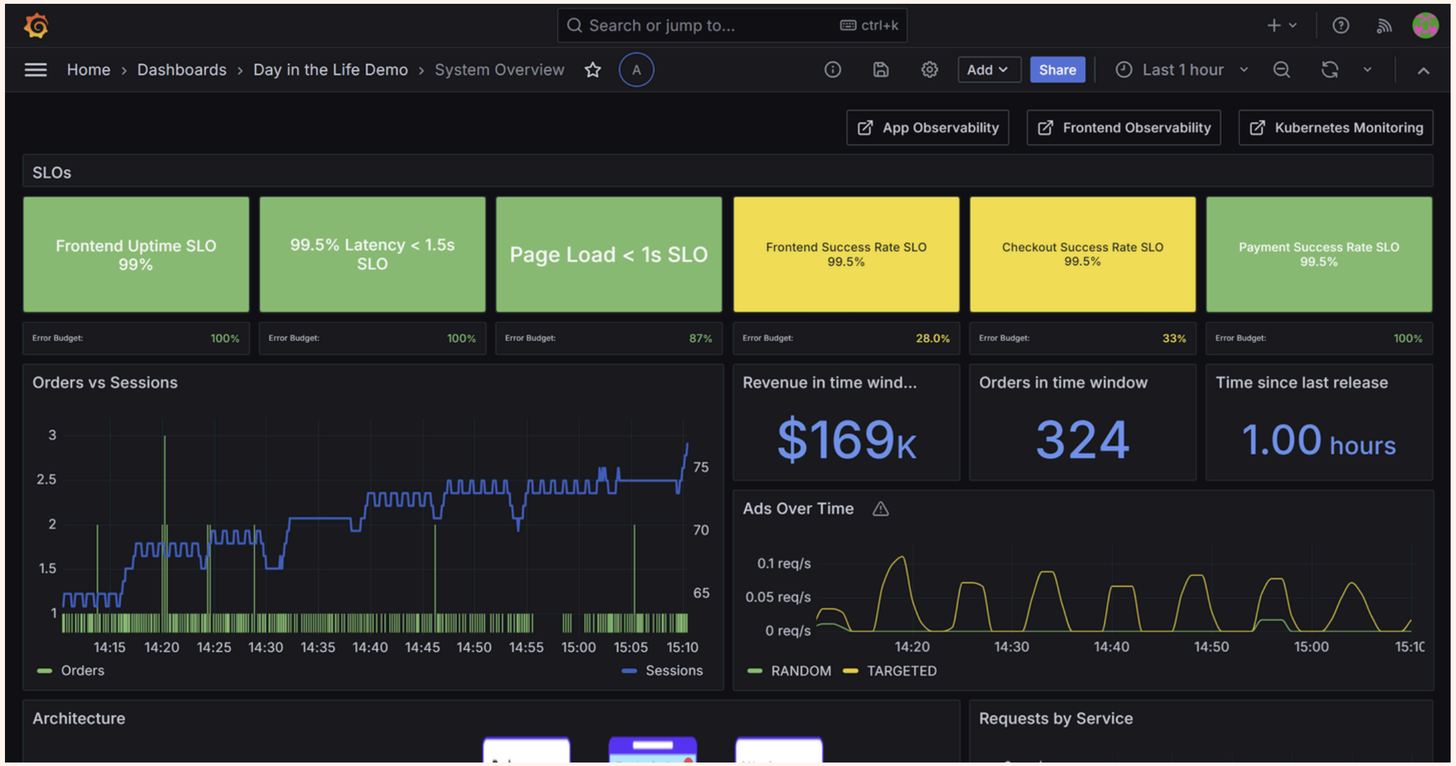


EJEMPLO:
EN EL CURSO UTILIZAREMOS TERRAFORM PARA DEFINIR SU INFRAESTRUCTURA DE MANERA LOCAL , DOCKER PARA EMPAQUETAR SUS APLICACIONES, KUBERNETES PARA DESPLEGAR Y ESCALAR AUTOMATICAMENTE Y GITHUB ACTIONS PARA EJECUTAR PRUEBAS Y DESPLIEGUES

ESCALABILIDAD Y DISPONIBILIDAD

LAS EMPRESAS NECESITAN SOLUCIONES QUE SE ADAPTEN A LA DEMANDA SIN INTERVENCION MANUAL

- KUBERNETES PERMITE ESCALAR APLICACIONES AUTOMATICAMENTE SEGUN EL TRAFICO Y LA CARGA DEL SISTEMA
- IAC FACILITA LA CREACION Y ELIMINACION DE RECURSOS EN LA NUBE DE MANERA PROGRAMATICA
- OBSERVABILIDAD PROPORCIONA METRICAS CLAVE PARA AJUSTAR LA INFRAESTRUCTURA EN TIEMPO REAL

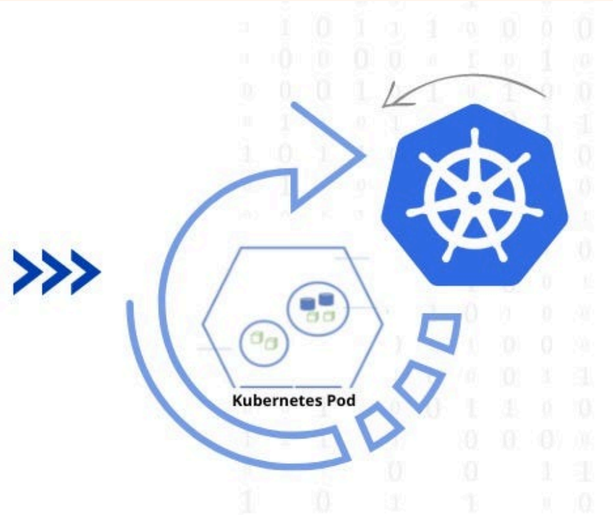


EJEMPLO:
UN SITIO DE ECOMMERCE QUE USA KUBERNETES PUEDE ESCALAR AUTOMATICAMENTE DURANTE EVENTOS COMO EL BLACK FRIDAY , ASEGURANDO DISPONIBILIDAD SIN DESPERDICIAR RECURSOS EN MOMENTOS DE BAJA DEMANDA

CONFIABILIDAD Y RCUPERACION ANTE FALLOS

PARA LOGRAR ALTA DISPONIBILIDAD, ES NECESARIO GARANTIZAR QUE LOS SISTEMAS PUEDAN RECUPERARSE RAPIDAMENTE ANTE FALLOS

- OBSERVABILIDAD AYUDA A DETECTAR PROBLEMAS ANTES DE QUE AFECTEN A LOS USUARIOS
- CI/CD REDUCE EL RIESGO DE FALLOS EN PRODUCCION CON PRUEBAS AUTOMATIZADAS Y DESPLIEGUES CONTROLADOS
- KUBERNETES IMPLEMENTA MECANISMOS DE AUTO-REPARACION , REINICIANDO CONTENEDORES FALLIDOS Y REDISTRIBUYENDO LA CARGA DE MANERA AUTOMATICA

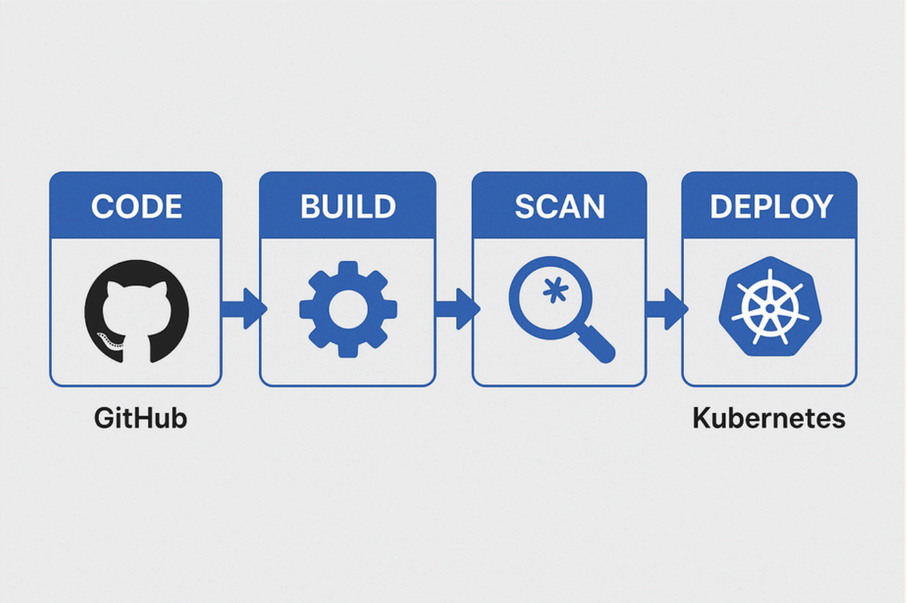


EJEMPLO:
SI UN MICROSERVICIO FALLA , KUBERNETES LO REINICIA AUTOMATICAMENTE Y LOS LOGS DE OBSERVABILIDAD PERMITEN IDENTIFICAR LA CAUSA RAIZ DEL PROBLEMA

SEEGURIDAD Y GESTION DE CONFIGURACION

LAS MALAS CONFIGURACIONES PUEDEN SER EXPLOTADAS POR ATACANTES O PROVOCAR ERRORRES CRITICOS

- IAC AYUDA A DEFINIR REGLAS DE SEGURIDAD CONSISTENTES PARA TODA LA INFRAESTRUCTURA
- CI/CD PUEDE INCORPORAR ANALISIS DE SEGURIDAD AUTOMATIZADOS PARA PREVENIR VULNERABILIDADES ANTES DEL DESPLIEGUE
- KUBERNETES GESTIONA PERMISOS Y SECRETOS DE MANERA CENTRALIZADA, EVITANDO FILTRACIONES DE CREDENCIALES

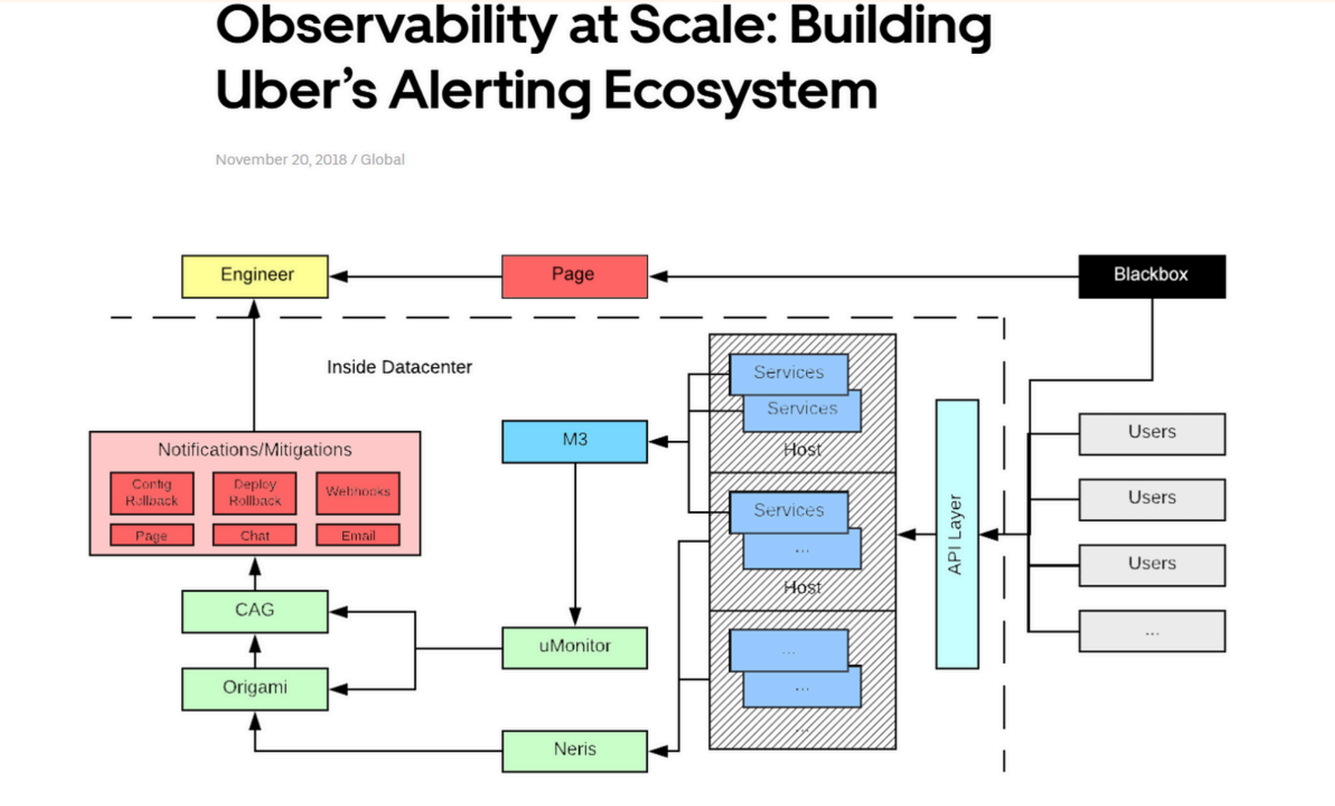


ejemplo:
Un pipeline de CI/CD en Github incluye una etapa de escaneo de vulnerabilidades antes de desplegar una aplicacion en Kubernetes

UBER - MONITOREO Y OBSERVABILIDAD

- **PROBLEMA**: UBER PROCESA MILLONES DE SOLICITUDES DE VIAJES EN TIEMPO REAL Y NECESITA DETECTAR PROBLEMAS DE LATENCIA RÁPIDAMENTE
- **SOLUCIÓN**: USA OBSERVABILIDAD CON PROMETHEUS Y GRAFANA PARA MONITOREAR MÉTRICAS EN TIEMPO REAL Y DETECTAR CUELLOS DE BOTELLA
- **IMPACTO**: MEJORA LA EXPERIENCIA DEL USUARIO REDUCIENDO TIEMPOS DE RESPUESTA Y DETECTANDO FALLOS ANTES DE QUE IMPACTEN EL SERVICIO

https://www.uber.com/en-PE/blog/observability-at-scale/



2.DEBATIR CÓMO LA ADOPCIÓN DE ESTAS PRÁCTICAS PUEDE ACELERAR EL “TIME TO MARKET” DE UN PRODUCTO.

LA IMPLEMENTACIÓN DE ESTAS TECNOLOGÍAS PERMITE REDUCIR EL TIEMPO NECESARIO PARA LANZAR UN PRODUCTO O ACTUALIZACIÓN AL MERCADO, ASEGURANDO CALIDAD, ESTABILIDAD Y ESCALABILIDAD DESDE EL INICIO.

INFRAESTRUCTURA COMO CÓDIGO (IAC): AUTOMATIZA LA CONFIGURACIÓN DE SERVIDORES Y ENTORNOS, REDUCIENDO TIEMPOS Y ERRORES MANUALES.
CONTENEDORES: FACILITAN LA PORTABILIDAD DEL SOFTWARE, EVITANDO PROBLEMAS DE COMPATIBILIDAD Y ACELERANDO DESPLIEGUES.
KUBERNETES: GESTIONA Y ESCALA APLICACIONES AUTOMÁTICAMENTE, ELIMINANDO CUELLOS DE BOTELLA OPERATIVOS.
OBSERVABILIDAD: PERMITE MONITOREAR EL SISTEMA EN TIEMPO REAL, DETECTAR FALLOS ANTES DE QUE AFECTEN A LOS USUARIOS Y OPTIMIZAR EL RENDIMIENTO.
CI/CD: AUTOMATIZA PRUEBAS Y DESPLIEGUES, PERMITIENDO ENTREGAS RÁPIDAS Y FRECUENTES SIN INTERVENCIÓN MANUAL.

AUTOMATIZACION

+

ESCALABILIDAD

+

OBSERVABILIDAD

=

REDUCCION DE TIEMPO

3. TRABAJO COLABORATIVO

FLUJO DEL PROCESO

1. COMMIT DEL CÓDIGO
   - UN DESARROLLADOR HACE UN PUSH DE CÓDIGO A UN REPOSITORIO EN GITHUB
2. EJECUCIÓN DEL PIPELINE CI/CD
   - UN SISTEMA COMO GITHUB ACTIONS DETECTA EL CAMBIO Y DISPARA EL PIPELINE.
   - SE EJECUTAN PRUEBAS UNITARIAS Y DE INTEGRACIÓN.
3. INFRAESTRUCTURA COMO CÓDIGO (IAC) - TERRAFORM
   - SI ES NECESARIO, SE PROVISIONAN RECURSOS EN LA NUBE (AWS, GCP, AZURE) USANDO TERRAFORM.
   - SE CREAN VPCS, CLÚSTERES DE KUBERNETES (EKS/AKS/GKE) Y ALMACENAMIENTO.
4. CONSTRUCCIÓN Y ALMACENAMIENTO DE LA IMAGEN DE CONTENEDOR
   - SE GENERA UNA IMAGEN DOCKER CON LA APLICACIÓN Y SE ENVÍA A UN REGISTRO DE CONTENEDORES (DOCKER HUB).
5. DESPLIEGUE EN KUBERNETES
   - KUBERNETES TOMA LA IMAGEN Y LA EJECUTA EN UN CLÚSTER MEDIANTE UN DEPLOYMENT.
   - SE USA UN SERVICE PARA EXPONER LA APLICACIÓN Y UN INGRESS PARA MANEJAR EL TRÁFICO HTTP.
6. MONITOREO Y OBSERVABILIDAD CON PROMETHEUS/GRAFANA
   - PROMETHEUS RECOLECTA MÉTRICAS DE LOS PODS Y DEL SISTEMA.
   - GRAFANA VISUALIZA ESTAS MÉTRICAS EN PANELES EN TIEMPO REAL.
   - SE CONFIGURAN ALERTAS PARA DETECTAR FALLOS AUTOMÁTICAMENTE.