PASIR

JADG

COMPARATIVA DE DESPLIEGUE Y AUTOESCALADO DE CONTENEDORES DOCKER

REALIZADO POR: JOSÉ ANTONIO DÍAZ GÓMEZ

CURSO: S21AR

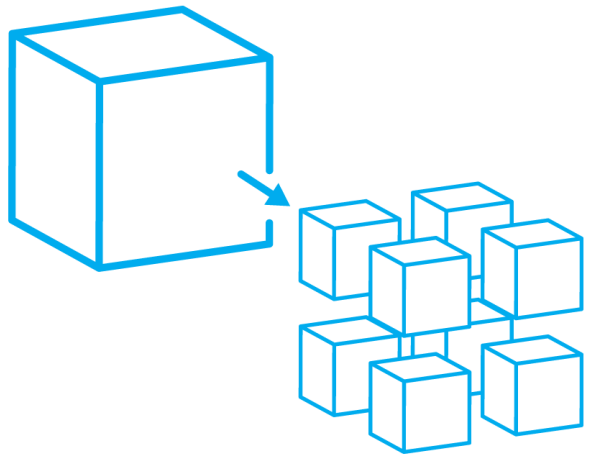
AÑO: 2020

JADG

Microservicios

Definición

- Son elementos **independientes** que funcionan en conjunto para llevar a cabo las mimas tareas.
- Es un elemento **fundamental de la optimización del desarrollo de aplicaciones** hacia un modelo nativo de la **nube**.
- **Libertad de desarrollar y desplegar servicios** de forma independiente.
- Fácil integración y despliegue.
- Fácil de entender y modificar.
- Fácil de escalar e integrar.

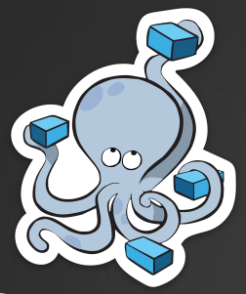


JADG

Ansible

¿Qué es?, Uso y utilidad

- Es un software para **automatizar el aprovisionamiento**.
- Permite gestionar los servidores y toda su configuración.
- Permite trabajar con proveedores en la nube como **AWS**, **Azure** o **Google Cloud Plataform**.
- Utiliza ficheros en formato YAML llamado **playbooks** en el que se describen todas las opciones a realizar en el nodo remoto.
- Gestiona los nodos **a través de SSH**.
- Sólo requiere tener instalado **Python** en los nodos remotos.



JADG

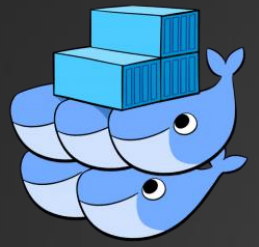
Docker-compose

¿Qué es?, Uso y utilidad

- Es una herramienta para **definir y ejecutar aplicaciones** Docker de contenedores múltiples.
- Utiliza un archivo **docker-compose.yml** para configurar servicios de la aplicación a desplegar.
- Para ejecutarlo, **nos encontramos en el directorio** donde tengamos todos nuestros ficheros.

```
root@swarm1:~# docker-compose up -d
```

```
version '3'
services:
   apache:
      container_name:webserver
      image: httpd:latest
      ports:
         - '80:80'
      volumes:
        - web:/var/www/html
```

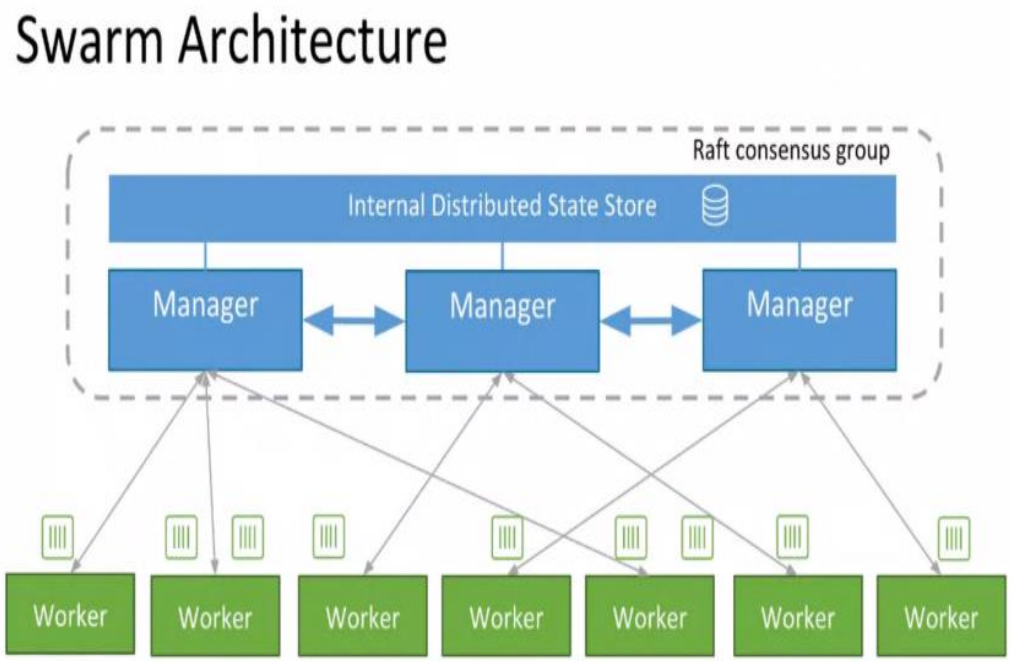


JADG

Docker Swarm

¿Qué es?

- Un clúster Swarm consiste en **Docker Engine implementado en múltiples nodos**.
- Es una herramienta que permite generar **servicios distribuidos entre diferentes nodos**, creando replicas de contenedores idénticos.
- Los nodos de trabajo reciben y ejecutan las tareas desde los nodos de administración.
- Los **servicios** pueden aumentar o disminuir en número.
- Los **nodos implicados** también pueden aumentarse, por lo que además de **alta disponibilidad** obtenemos **escalabilidad vertical.**

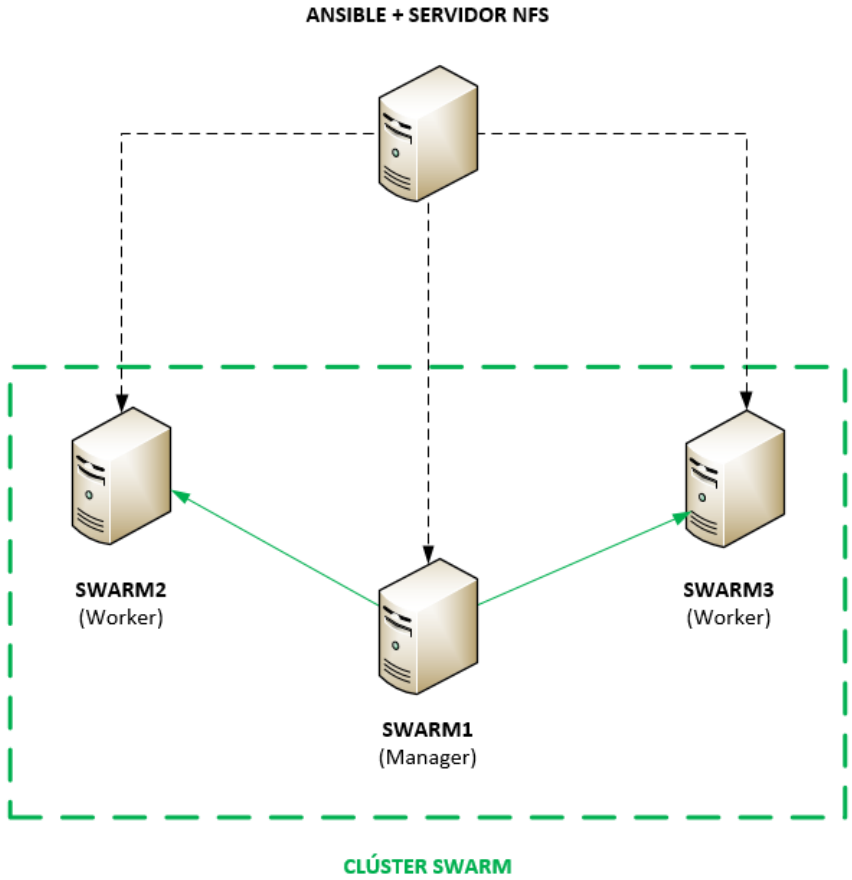


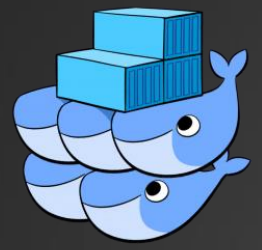
JADG

Docker Swarm

Servidor Web + Redis

- Para crear la alta disponibilidad con docker Swarm, utilizaremos:
  - Equipo externo al clúster con servidor NFS.
  - **Ansible:** instalamos cliente NFS en los nodos del clúster.
  - Dos imágenes ya creadas de redis y web.
  - Swarm1 se comporta como **Manager.**
  - Swarm2 y Swarm3 son los nodos **Workers.**





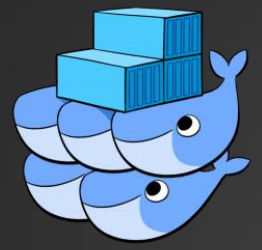
JADG

Docker Swarm

Explicación yml

- Las **imágenes** serán las que habremos creado en apartados anteriores.
- **Replicas:** nº de réplicas que tendrá el servicio definido.
- **Limits:** Limites de recursos, que en este caso es un 20% CPU y 256MB.
- **Placement:** se define el rol de nodo donde quieres correr ese servicio, en este caso se redis solo se creara en los nodos worker.
- **Restart policy:** reinicio en caso de fallar, con un retraso de 5 segundos y con 3 intentos.
- **Ports:** puertos a exponer en el servicio.
- **Volume:** se define un espacio para compartir archivos del contenedor al exterior o al contrario.
- **Network:** red que tendrán los servicios desplegados.

```
version: "3"

services:
  web-php:
    image: joseadiaz/web-php
    deploy:
        mode: replicated
        replicas: 2
        resources:
          limits:
            cpus: "0.2"
            memory: 256M
        restart_policy:
            condition: on-failure
            delay: 5s
            max_attempts: 3
    ports:
      - "8080:80"
    volumes:
      - /html:/var/www/html
    networks:
      - clusnet


  redis:
    image: joseadiaz/redis
    deploy:
        mode: replicated
        replicas: 2
        resources:
          limits:
            cpus: "0.2"
            memory: 256M
        restart_policy:
            condition: on-failure
            delay: 5s
            max_attempts: 3
        placement:
            constraints: [node.role == worker]
    ports:
     - "6379:6379"
    volumes:
      - /data:/data
    networks:
      - clusnet

networks:
   clusnet:
```



Docker Swarm

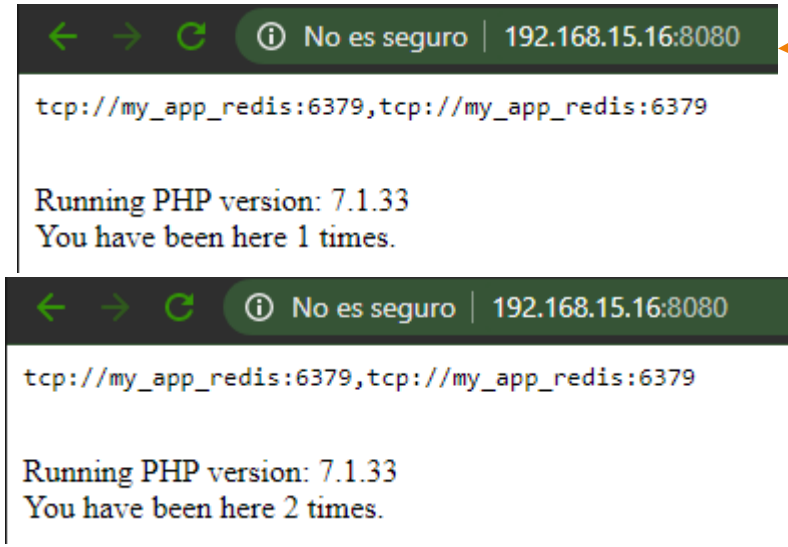
Despliegue

JADG

- Se desplegará el stack.

```
root@swarm1:~/docker-php-redis# docker stack deploy -c docker-compose.yml my_app
Creating network my_app_clusnet
Creating service my_app_web-php
Creating service my_app_redis
```

- Se comprobará con un php que redis y el servidor web están funcionado.



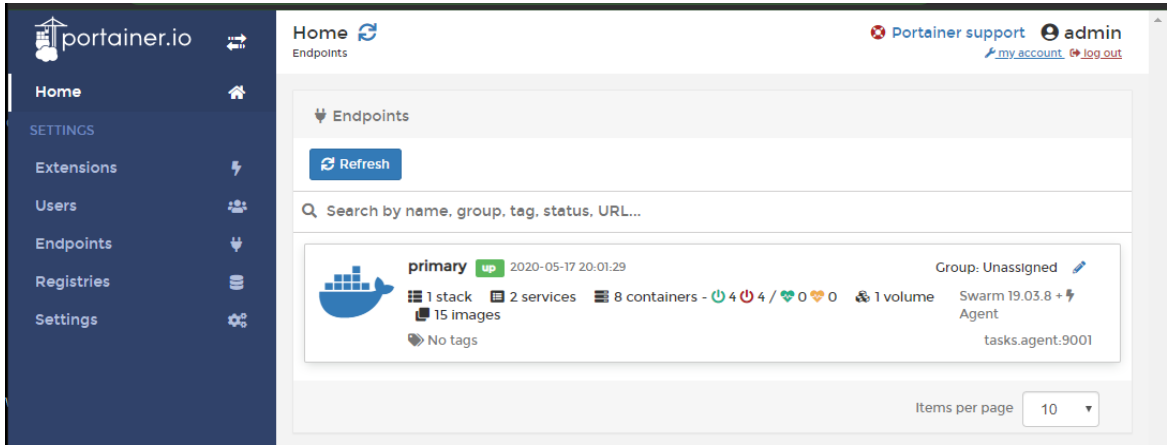

```php
<?php

ini_set('session.save_handler', 'redis');
ini_set('session.save_path', 'tcp://my_app_redis:6379,tcp://my_app_redis:6379');

session_name('FOOBAR');
session_start();
echo nl2br('<pre>' . session_save_path() . '</pre>' . PHP_EOL);

echo nl2br('Running PHP version: ' . phpversion() . PHP_EOL);

if (!array_key_exists('visit', $_SESSION)) {
    $_SESSION['visit'] = 0;
}
$_SESSION['visit']++;

echo nl2br('You have been here ' . $_SESSION['visit'] . ' times.');
```

JADG

Portainer

¿Qué es?, Uso y utilidad

- Portainer es una **herramienta web open-source** la cual se ejecuta ella misma como un contenedor, por lo tanto deberemos tener instalado Docker.
- Esta aplicación nos va a permitir gestionar de forma **muy fácil e intuitiva** nuestros contenedores.
- Con esta herramienta podemos administrar **las pilas de Docker, contenedores, imágenes, volúmenes y redes**.

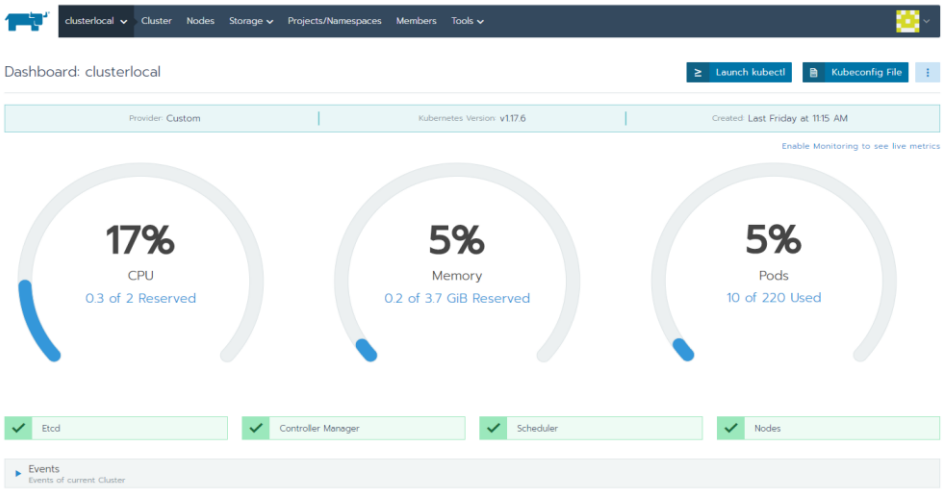


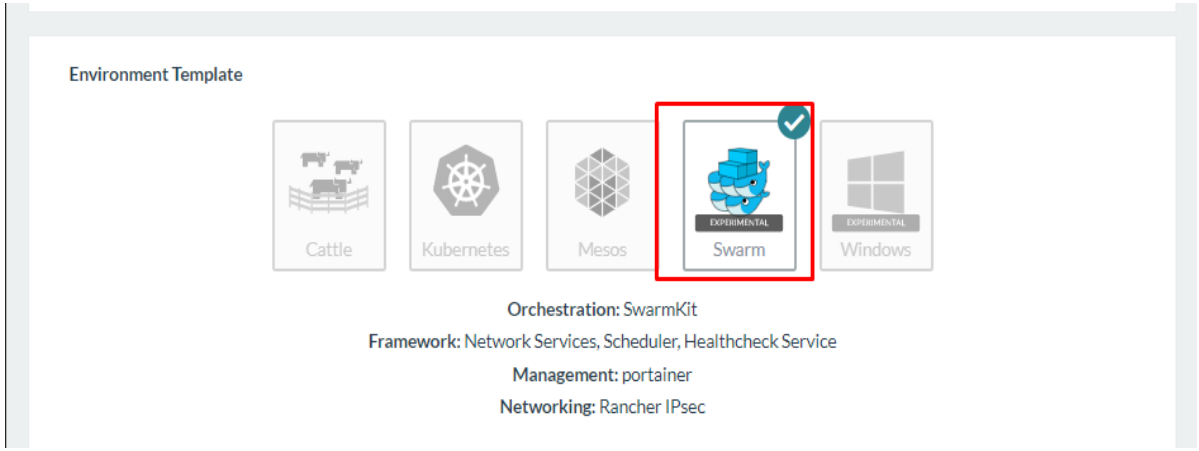
JADG

Rancher

¿Qué es?, Uso y utilidad

- Rancher es una pila completa de software para equipos que adoptan contenedores.
- Puede **administrar múltiples clústeres en cualquier infraestructura**, al mismo tiempo proporciona **herramientas integradas para ejecutar cargas de trabajo** en contenedores.



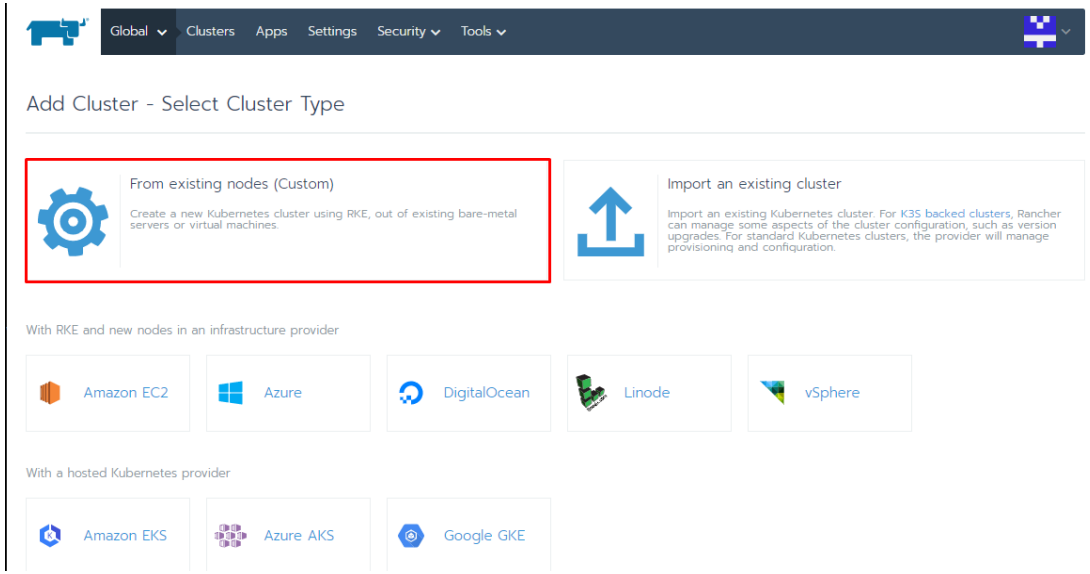


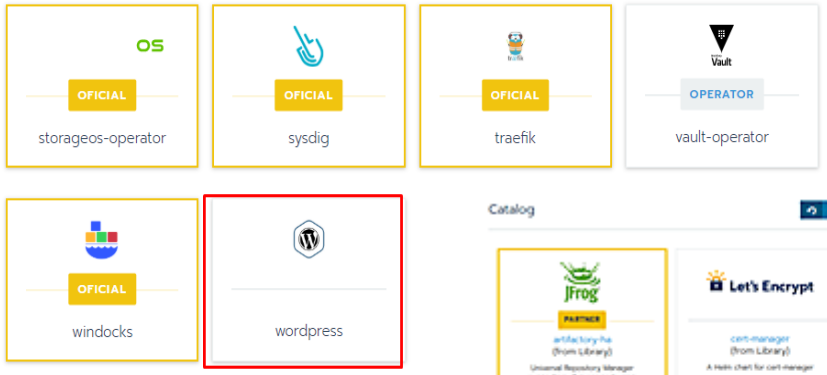
JADG

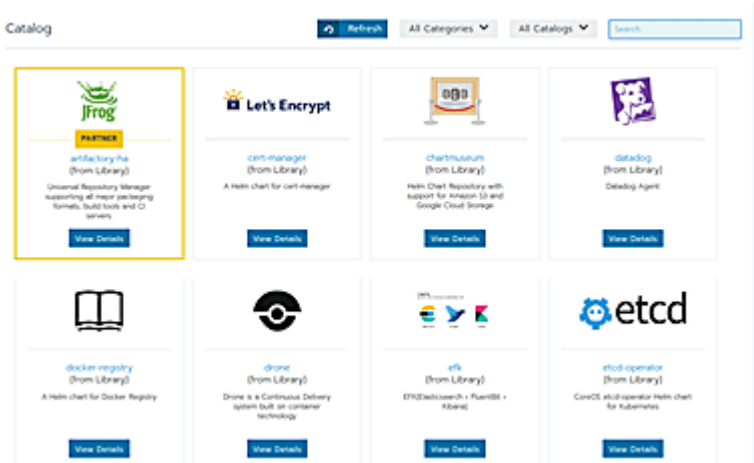
Rancher

¿Qué es?, Uso y utilidad

- Tiene un **catalogo amplio de aplicaciones** centralizado.
- Podremos **administrar clústeres locales y aquellos alojados en servicios en la nube**.





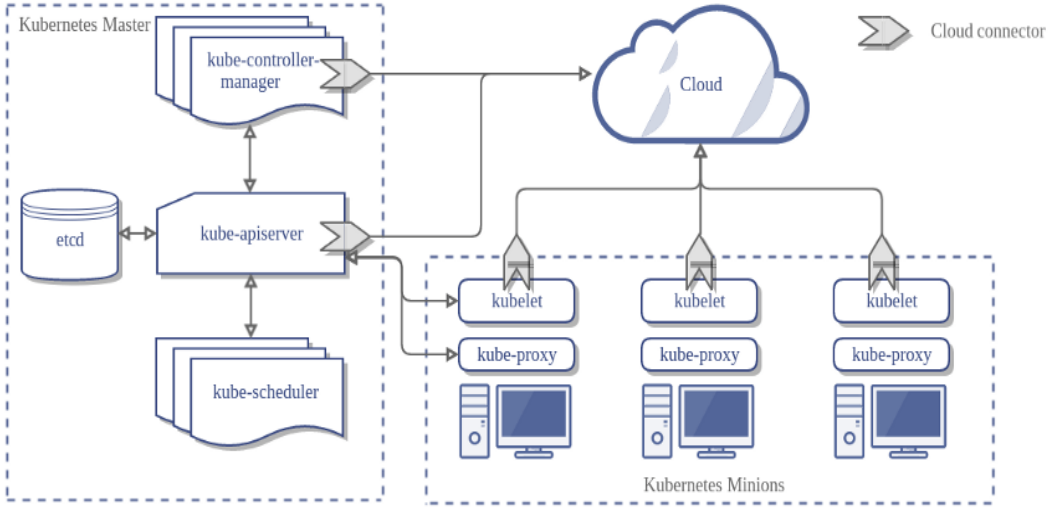


JADG

Kubernetes

¿Qué es?

- Kubernetes es una herramienta extensible y de código abierto para gestionar cargas de trabajo y servicios en contenedores.
- Tiene un ecosistema grande y de rápido crecimiento.
- Existen una serie de componentes asociados al clúster de kubernetes.
  - **Etcd:** Almacena los datos de configuración.
  - **Kube-Apiserver:** es el centro de gestión para el nodo Master, facilita la comunicación entre los diversos omponentes.
  - **Kube-Scheduler:** coloca la carga de trabajo en el nodo que corresponde
  - **Kubelet:** recibe las especificaciones del pod del servidor API y administra los pods que se ejecutan en el host.



JADG

Kubernetes

Objetos

- **Pods**: es la unidad fundamental de despliegue en Kubernetes. Un pod sería el equivalente a la **mínima unidad funcional de la aplicación**.
- **Replicaset:** asegura que se esté ejecutando un **número específico de réplicas de pod** en un momento dado.
- **Deployments:** estos bloques de construcción se pueden usar para crear y administrar un grupo de pods.
- **Service:** Definiremos un service para poder **exponer el pod dentro y/o fuera de nuestro clúster**. Puede ser de tres tipos:
  - **ClusterIP:** expone el servicio en una IP interna de clúster.
  - **NodePort:** Expone el servicio en la IP de cada Nodo en un puerto estático.
  - **LoadBalancer:** expone el servicio externamente utilizando el equilibrador de carga de un proveedor en la nube.

Kubernetes

Ejemplos yaml

JADG

- Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dotnet
spec:
  containers:
  - name: dotnet
    image: ualmtorres/dotnet2019web:v1
```

- Estos yaml se desplegaran con **kubectl.**

```
kubectl apply -f Servicie.yaml
```

- Deployment.yaml

Deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- Service.yaml

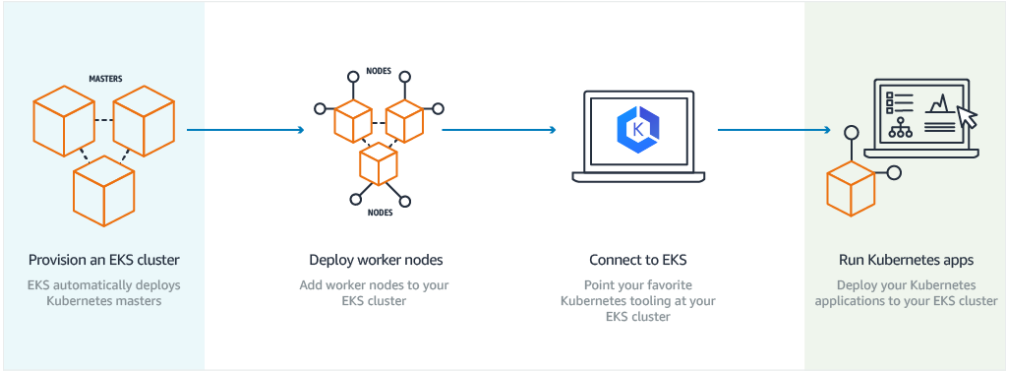
```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      nodePort: 31111
      targetPort: 80
```

JADG

Amazon EKS

¿Qué es?

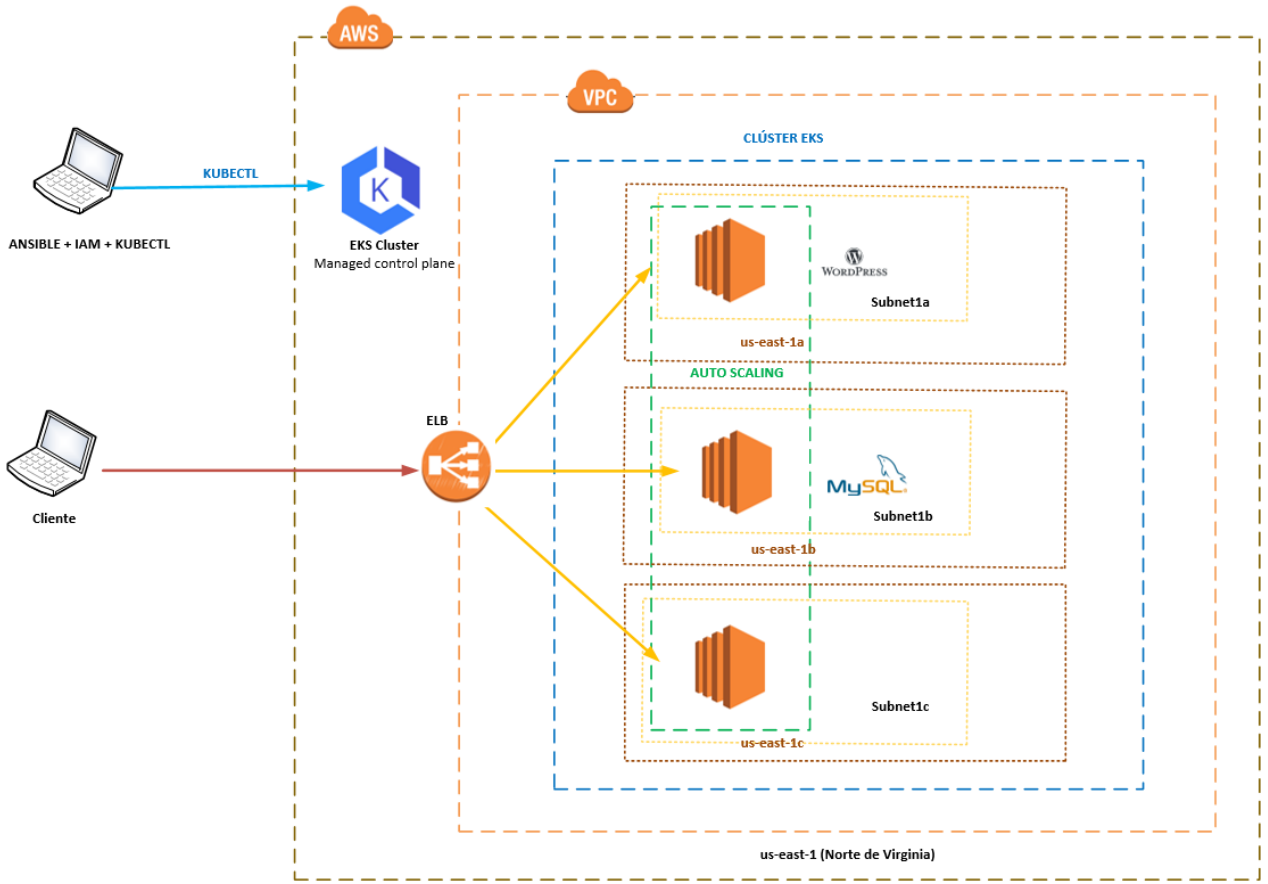
- **Amazon Elastic Kubernetes Service** (Amazon EKS) es un servicio administrado que le permite ejecutar fácilmente Kubernetes en AWS sin necesidad de crear ni mantener su propio plano de control de Kubernetes.
- Amazon EKS se integra también con numerosos servicios de AWS para ofrecer escalabilidad y seguridad a las aplicaciones, como los siguientes:
  - **Amazon ECR** para imágenes de contenedor.
  - **Elastic Load Balancing** para la distribución de carga.
  - **IAM** para la autenticación.
  - **Amazon VPC** para el aislamiento.



JADG

Ansible + EKS

Esquema de red

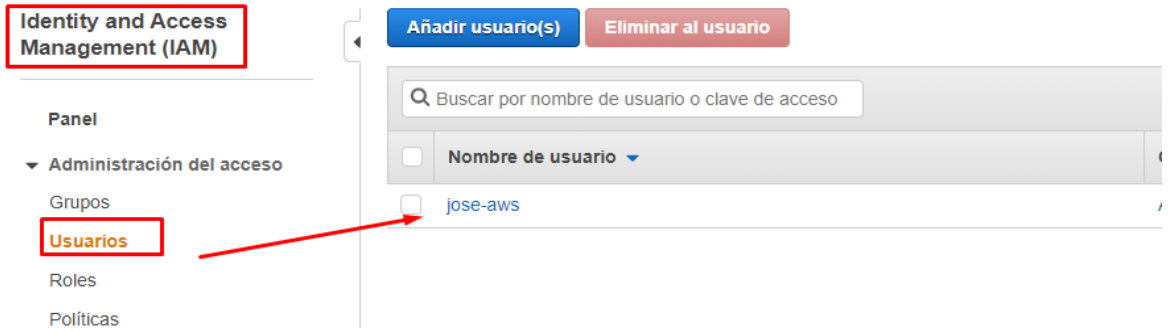


JADG

Ansible + EKS

Prerrequisitos

- Ansible instalado.
- Usuario IAM con privilegios para crear la infraestructura.



- Credenciales. Formato csv.

```
root@debiangraf:~/cluster-aws-eks# export AWS_ACCESS_KEY_ID='[redacted]'
root@debiangraf:~/cluster-aws-eks# export AWS_SECRET_ACCESS_KEY='[redacted]'
```

- Librería de Python **boto.**

```
root@debiangraf:~/cluster-aws-eks# pip install boto
```

JADG

Ansible + EKS

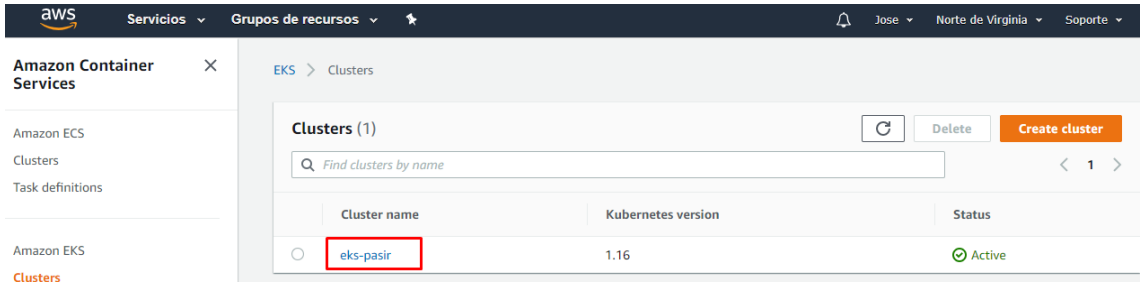
Despliegue de la infraestructura

- Ejecución de yaml para desplegar la infraestructura.

```
root@debiangraf:~/cluster-aws-eks# ansible-playbook -i inventory main.yml -vvvv
Using /etc/ansible/ansible.cfg as config file
Loading callback plugin default of type stdout, v2.0 from /usr/lib/python2.7/dist-packages/ansible/plugins/callback/__init__.pyc

PLAYBOOK: main.yml ************************************************************
1 plays in main.yml

PLAY [localhost] **************************************************************

TASK [Ensure VPC exists via CloudFormation.] **********************************
task path: /root/cluster-aws-eks/main.yml:9
Using module file /usr/lib/python2.7/dist-packages/ansible/modules/core/cloud/amazon/cloudformation.py
<127.0.0.1> ESTABLISH LOCAL CONNECTION FOR USER: root
<127.0.0.1> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo ~/.ansible/tmp/ansible-tmp-1592311392.01-45070944
```

```
            "last_updated_time": null,
            "logical_resource_id": "NodeInstanceRole",
            "physical_resource_id": "eks-pasir-nodegroup-NodeInstanceRole-1R09P3JHRMGDW",
            "resource_type": "AWS::IAM::Role",
            "status": "CREATE_COMPLETE",
            "status_reason": null
        }
    ]
}

PLAY RECAP ********************************************************************
127.0.0.1                  : ok=5    changed=3    unreachable=0    failed=0
```

- Infraestructura creada.



JADG

Ansible + EKS

Despliegue de la aplicación

- Instalación de AWS CLI para creación de **kubeconfig**.
- Creación de kubeconfig.

```
root@debiangraf:~# aws eks --region us-east-1 update-kubeconfig --name eks-pasir --kubeconfig ~/.kube/eks-pas
ir
Added new context arn:aws:eks:us-east-1:915508415579:cluster/eks-pasir to /root/.kube/eks-pasir
```

- Instalación de **Kubectl** para interacción con el clúster.

```
root@debiangraf:~# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   5h30m
```

- Ejecución de yaml para desplegar aplicación.

```
root@debiangraf:~/cluster-aws-eks# ansible-playbook -i inventory deploy.yml -vvvv
ansible-playbook 2.9.9
  config file = /etc/ansible/ansible.cfg
```

```
}
META: ran handlers
META: ran handlers

PLAY RECAP *********************************************************************************************************************************************************
127.0.0.1                  : ok=7    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Ansible + EKS

JADG

DEMOSTRACIÓN

JADG

FIN

¿Preguntas?

Los script del proyecto han sido subido a mi repositorio: https://github.com/JoseDiazGomez/PASIR-2020