

Containers

quarta-feira, 8 de agosto de 2018

15:05

A ideia de **container** transforma como serviços são entregados. Seu uso traz benefícios como redução de custos de infraestrutura e de mão de obra necessária para mantê-la.

Para entender melhor o que é um container é preciso saber o que é uma **imagem docker**. Uma imagem docker é um pacote executável que inclui tudo o que você precisa para rodar uma aplicação: código, bibliotecas, variáveis de ambiente, etc.

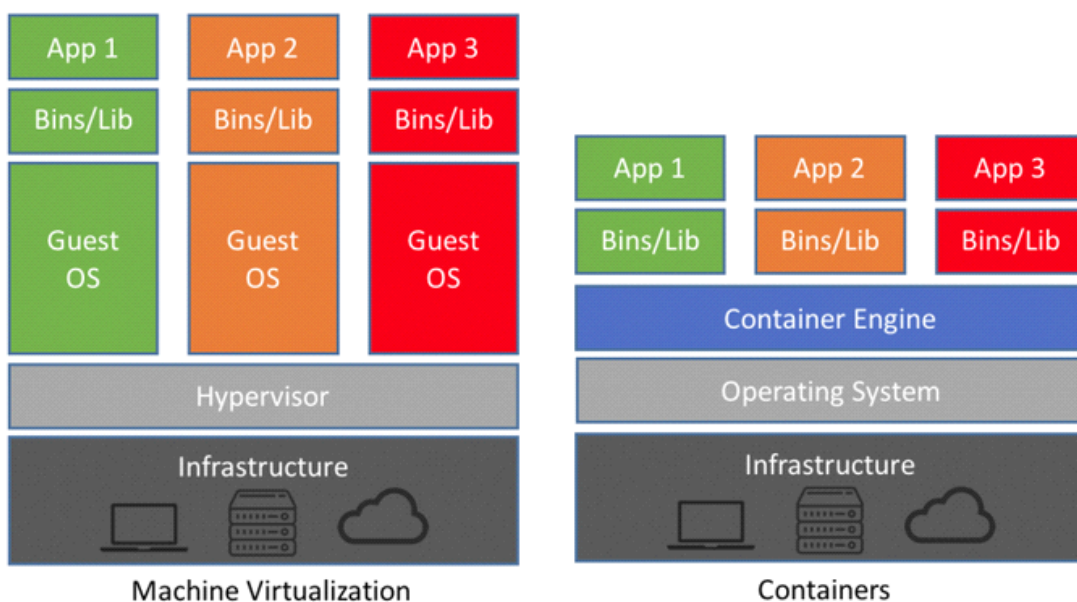
Um container é uma execução, ou uma **instância**, de uma imagem. Assim, pode-se executar ao mesmo tempo vários containers baseados em uma mesma imagem.

Uma grande vantagem no uso de containers está na **portabilidade**. É possível executar um container localmente, dentro de uma máquina virtual como VirtualBox e VMWare, e nas principais plataformas em nuvem que oferecem suporte como AWS, Azure e Google Cloud.

Além disso é muito fácil criar uma snapshot de um imagem docker, compartilhá-la e fazer seu deploy em diferentes ambientes.

Imagens docker usam um sistema de camadas. Se você pegar uma docker que tenha uma instalação linux, instalar bibliotecas, e realizar um "commit" da imagem, o commit conterá apenas as mudanças realizadas. Além disso as imagens são versionadas. Caso algum problema ocorra com uma nova versão é possível realizar um rollback para uma versão anterior. É fácil perceber as semelhanças com o GIT e portanto os ganhos de produtividade que isso traz.

Docker Vs. Máquina Virtual



Vantagens do Docker em relação a Máquina Virtual

- Leveza: Containers compartilham recursos com o sistema operacional base (kernel) e entre si (file system). Por exemplo: Considerando que um container tenha uma imagem base de 1GB, contendo Ubuntu e bibliotecas usadas pelo serviço. Desejamos rodar 10 instâncias desse serviço. Caso usássemos VMs, precisaríamos de 10 GB. Já com docker, apenas um pouco mais que 1 GB.
- Eficiência: Recursos de hardware como CPU e memória são alocados dinamicamente entre os containers ao contrário de VMs, em que são fixos

- Rapidez: Containers demoram segundos para inicializar, ao contrário de minutos no caso de VMs.

Vantagens de Máquina Virtual

- VMs fornecem isolamento maior da aplicação com o sistema operacional Host.
- Por usar virtualização de hardware é possível garantir melhor os recursos alocados a uma VM.
- Dependendo do caso é mais fácil e rápido usar VM, principalmente para testes. É relativamente fácil criar uma máquina virtual e instalar tudo o que você precisa para executar um ou mais serviços. Já em containers geralmente buscamos uma imagem base que tenha um SO "adaptado" para container, e então usamos scripts para automatizar o download/instalação/execução de um único serviço. Apesar de ser possível rodar múltiplos serviços em um container (ver [supervisord](#)), o docker foi feito para executar um único processo.

Cheatsheet

quarta-feira, 8 de agosto de 2018 15:04

Fonte:

https://www.docker.com/sites/default/files/Docker_CheatSheet_08.09.2016_0.pdf

BUILD

Build an image from the Dockerfile in the current directory and tag the image

```
docker build -t myapp:1.0 .
```

List all images that are locally stored with the Docker engine

```
docker images
```

Delete an image from the local image store

```
docker rmi alpine:3.4
```

SHIP

Pull an image from a registry

```
docker pull alpine:3.4
```

Retag a local image with a new image name and tag

```
docker tag alpine:3.4 myrepo/myalpine:3.4
```

Log in to a registry (the Docker Hub by default)

```
docker login my.registry.com:8000
```

Push an image to a registry

```
docker push myrepo/myalpine:3.4
```

RUN

```
docker run
```

```
--rm remove container automatically after it exits
-it connect the container to terminal
--name web name the container
-p 5000:80 expose port 5000 externally and map to port 80
-v ~/dev:/code create a host mapped volume inside the container
alpine:3.4 the image from which the container is instantiated
/bin/sh the command to run inside the container
```

Stop a running container through SIGTERM

```
docker stop web
```

Stop a running container through SIGKILL

```
docker kill web
```

Create an overlay network and specify a subnet

```
docker network create --subnet 10.1.0.0/24
--gateway 10.1.0.1 -d overlay mynet
```

List the networks

```
docker network ls
```

List the running containers

```
docker ps
```

Delete all running and stopped containers

```
docker rm -f $(docker ps -aq)
```

Create a new bash process inside the container and connect it to the terminal

```
docker exec -it web bash
```

Print the last 100 lines of a container's logs

```
docker logs --tail 100 web
```

Docker

sexta-feira, 17 de agosto de 2018 11:49

Instalação do Docker no Windows

Observação: Existem serviços web que fornecem um ambiente Docker para aprendizado, de forma gratuita:

<https://labs.play-with-docker.com/>

<https://www.katacoda.com/courses/docker/playground>

Baixar Docker do Windows em:

Windows 10:

<https://download.docker.com/win/stable/Docker%20for%20Windows%20Installer.exe>

Versões anteriores do Windows:

<https://download.docker.com/win/stable/DockerToolbox.exe>

Ou acessar esta página para maiores detalhes:

<https://store.docker.com/editions/community/docker-ce-desktop-windows>

Ambas versões (normal e legacy) vem com uma interface gráfica que auxilia no gerenciamento de containers Docker. É recomendado, no entanto, o uso de linha de comando.

Quando pessoas falam em "Docker" eles geralmente se referem à Docker Engine, que é uma aplicação que contém o daemon do Docker e uma CLI para se comunicar com o daemon.

As versões mais recentes de Windows e Mac suportam o Docker Engine nativamente. Já para sistemas operacionais mais antigos é necessário usar a ferramenta **docker-machine**, que provisiona máquinas virtuais com a Docker Engine.

docker-machine (apenas legacy)

Listar máquinas:

`docker-machine ls`

Criar uma máquina com nome "default":

`docker-machine create --driver virtualbox default`

Configurar o comando "docker" para que use a máquina "default":

`docker-machine env default`

Parar máquina "default":

`docker-machine stop default`

Iniciar máquina "default":

`docker-machine start default`

Mais detalhes sobre o docker-machine:

<https://docs.docker.com/machine/get-started/#create-a-machine>

Testando instalação

Caso esteja usando docker-machine, visualizar IP da VM criada pelo docker-machine:

`docker-machine ip`

Caso esteja usando Linux ou Windows 10, como o Docker Engine roda nativamente na máquina, o seu IP é "localhost"

O comando a seguir cria um container a partir da imagem "nginx" e faz um mapeamento da porta 8000 da máquina host do docker à porta 80 do container:

`docker run -d -p 8000:80 nginx`

É possível verificar se o container está rodando por meio do comando "**docker ps**":

```
PS C:\Users\soval> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS               NAMES
d3a2f354a6cf   nginx     "nginx -g 'daemon of"   About a minute ago   Up 58 seconds   0.0.0.0:8000->80/tcp   amazing_brahmagupta
```

Testando agora no navegador:



Usando o Docker

Neste tutorial usaremos uma aplicação java para exemplificar os principais comandos do Docker.

O programa usa a framework Spring Boot, o gerenciador de dependências Gradle e é escrito em kotlin. O utilitário a seguir gera uma estrutura de projeto com as pastas, arquivos e dependências necessárias para começar o desenvolvimento: <https://start.spring.io/>

Inicialmente, as únicas modificações feitas no projeto base é a adição de um Controlador REST que recebe uma string via chamada GET e retorna-la ao cliente, e o parâmetro server.port no arquivo src/main/resources/application.properties, que está definindo a porta da aplicação a partir da variável de ambiente HELLO_PORT (a porta é 8080 caso a variável não exista).

HelloScopusController.kt:

```
1 package br.scopus.exemplo.helloscopus
2
3 import ...
4
5 @Controller
6 class HelloScopusController{
7
8     @GetMapping(path= arrayOf("/{nome}","/"))
9     fun hello(@PathVariable(value="nome",required=false) nome : String?) : ResponseEntity<String>{
10         return ResponseEntity( body: "Olá ${if(nome.isNullOrBlank()) "anonimo" else nome} !!",HttpStatus.OK)
11     }
12 }
```

application.properties:

```
1 server.port=${HELLO_PORT:8080}
```

Hierarquia de uma aplicação Docker:

1. Container: É uma instância de uma imagem docker
2. Serviço: Define como containers de uma determinada imagem se comportam em produção.
3. Stack: Grupo de diferentes serviços que compartilham dependências

Container

O container é definido por um arquivo chamado **Dockerfile**. Normalmente em um Dockerfile nós definimos a imagem base do container, comandos de instalação de inicialização do aplicativo, variáveis de ambiente e porta em que o container irá se expor para o mundo externo.

Foi criado o seguinte Dockerfile na pasta root do projeto:

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
COPY build/libs/helloscopus-*.jar helloscopus.jar
ENV HELLO_PORT 8090
ENTRYPOINT exec java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /helloscopus.jar
```

O comando FROM indica a imagem base. No caso, a imagem é composta por uma distro linux "alpine" com o OpenJDK8 instalado.

De forma alternativa seria possível usar apenas uma imagem linux como base e adicionar ao Dockerfile comandos para instalar o JDK, como "RUN apt-get install openjdk". É comum usar o comando RUN para instalar dependências.

O comando VOLUME monta um diretório no container com o path indicado. Para maiores detalhes: <https://docs.docker.com/storage/volumes>

COPY é usado para adicionar arquivos do computador para a imagem. No exemplo é usado para mover o executável java para a pasta raiz do container.

ENV define variáveis de ambiente no container.

ENTRYPOINT é um comando que sempre será executado quando o container iniciar. No exemplo, ele executa o .jar da aplicação.

Para criar a imagem:

```
docker build -t helloscopus .
```

Verificar se a imagem está disponível no computador:

```
docker images
```

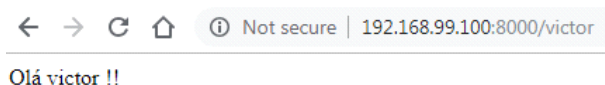
Rodar imagem na porta 8000 do computador:

```
docker run -d -p 8000:8090 helloscopus
```

Verificar se o container foi instanciado:

```
docker ps
```

Se o container foi criado de forma bem-sucedida, a aplicação já está disponível para ser usada:



The screenshot shows a web browser address bar with the URL '192.168.99.100:8000/victor'. The page content displays 'Olá victor !!'.

Serviço

Um serviço está atrelado à uma imagem e é responsável por configurar como um container se comporta em produção. Existem vários parâmetros de configuração, como o número de replicas e a porta que será usada. O **docker-compose.yml** é um arquivo que define um ou mais serviços.

docker-compose.yml usado:

```
version: "3"
```

```

services:
  web:
    image: vval/helloscopus:0.4
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: "0.1"
          memory: 100M
      restart_policy:
        condition: on-failure
    ports:
      - "8001:8090"

```

O arquivo define um serviço com nome "web", composto por 3 containers gerados pela imagem "vval/helloscopus:0.4". O serviço limita a esses containers 10% da CPU e 100MB de memória da máquina host.

Além do docker-compose.yml o aplicativo foi alterado para exibir um número aleatório gerado em sua inicialização. Isso é útil para determinar qual réplica recebeu a requisição. Também foi adicionado um contador de requisições.

```

13  @Controller
14  class HelloScopusController{
15
16      val hostId = Random().nextInt( bound: 1000000)
17      var numAcessos = 0
18
19      @GetMapping(path= arrayOf("/{nome}","/"), produces = arrayOf(MediaType.TEXT_PLAIN_VALUE))
20      fun hello(@PathVariable(value="nome",required=false) nome : String?) : ResponseEntity<String>{
21          numAcessos++
22          val resp = "Olá ${(if(nome.isNullOrBlank()) "anonimo" else nome)} !! \nHost: ${hostId} \nAcessos: ${numAcessos}"
23          return ResponseEntity(resp,HttpStatus.OK)
24      }
25  }

```

Para iniciar o serviço:

```

docker swarm init
docker stack deploy -c docker-compose.yml helloscopus_service

```

Verificando que o serviço foi iniciado:

```

docker service ls

```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
h7k061ma6rkf	helloscopus_web	replicated	3/3	vval/helloscopus:0.4	*:8001->8090/tcp

A coluna REPLICAS mostra o número de containers disponíveis.

Para testar usei o comando curl:

```

Acessos: 6victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 282187
Acessos: 38victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 419713
Acessos: 38victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 954957
Acessos: 7victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 282187
Acessos: 39victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 419713
Acessos: 39victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 954957
Acessos: 8victor@victor-VirtualBox:~/helloscopus$ curl http://localhost:8001/
Olá anonimo !!
Host: 282187

```

Como pode ser visto, as requisições podem ser respondidas por qualquer um dos 3 containers do serviço.

Para deletar o serviço:

```

docker service rm helloscopus_web

```

Stack

Stack é um grupo que serviços relacionados que compartilham dependências. Um único stack é capaz de definir e coordenar a funcionalidade de toda a aplicação. No entanto, também é possível uma aplicação utilizar mais de um stack.

Na realidade já realizamos o deploy de um stack na seção anterior. Porém o stack anterior tinha apenas 1 serviço. O **docker-compose.yml** a seguir define um stack com 2 serviços, sendo um deles um app de monitoramento dos containers na rede.

```

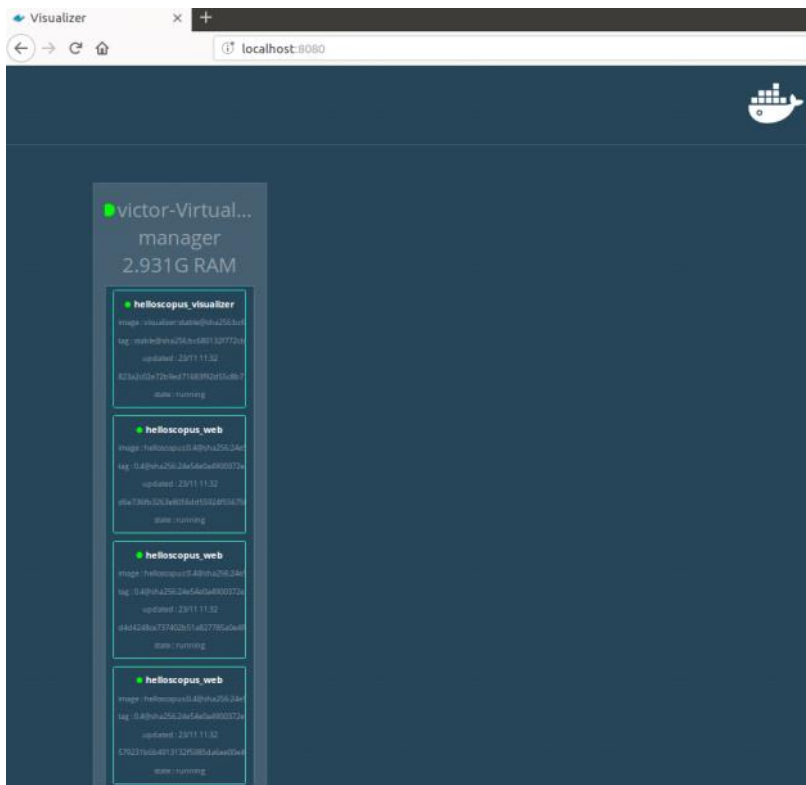
version: "3"
services:
  web:
    image: vval/helloscopus:0.4

```

```

deploy:
  replicas: 3
  restart_policy:
    condition: on-failure
resources:
  limits:
    cpus: "0.1"
    memory: 100M
ports:
  - "8001:8090"
networks:
  - webnet
visualizer:
  image: dockersamples/visualizer:stable
  ports:
    - "8080:8080"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
deploy:
  placement:
    constraints: [node.role == manager]
networks:
  - webnet
networks:
  webnet:

```



Observação: No tutorial foi usado o comando **docker swarm** para iniciar serviços. O swarm é um orquestrador de containers, ou seja, provê funcionalidades como deploy de containers, auto-scaling, load-balancing, monitoramento, etc. No tutorial o swarm criado possui apenas 1 nó, mas é possível ter múltiplos nós (computadores físicos ou virtuais) em um cluster swarm.

O Swarm não foi visto com maiores detalhes porque seu uso no mercado está diminuindo. Atualmente o orquestrador mais utilizado é o Kubernetes e suas distribuições comerciais (como o Openshift da Redhat).

CMD vs ENTRYPOINT

<https://stackoverflow.com/questions/21553353/what-is-the-difference-between-cmd-and-entrypoint-in-a-dockerfile>

The ENTRYPOINT specifies a command that will always be executed when the container starts. The CMD specifies arguments that will be fed to the ENTRYPOINT

ADD vs COPY

<https://stackoverflow.com/questions/24958140/what-is-the-difference-between-the-copy-and-add-commands-in-a-dockerfile?noredirect=1&lq=1>

Container Registry

quinta-feira, 24 de janeiro de 2019 14:02

Docker Hub

Docker Trusted Registry - Enterprise-grade

Docker Distribution

Harbor

Nexus

Introdução ao Kubernetes

quinta-feira, 12 de julho de 2018 16:55

O Kubernetes é uma plataforma open-source para gerenciar aplicações containerizadas e serviços em um cluster.

O kubernetes elimina necessidade de instalar aplicações manualmente em máquinas físicas. Além disso ele possui mecanismos que garantem alta disponibilidade para as aplicações do cluster e balanceamento da carga de trabalho entre as máquinas disponíveis.

Algumas das vantagens do kubernetes são:

- **Agendamento automático de containers** em função de limitações computacionais, sem sacrificar a disponibilidade.
- **Self-healing**: faz uso de health checks para verificar se um container está disponível. Automaticamente elimina e reagenda containers.
- **Horizontal scaling** (escalonamento horizontal): aumenta ou diminui o número de containers de uma aplicação por meio de comandos do administrador ou de forma dinâmica, com base em recursos computacionais ou até mesmo em métricas de negócios.
- **Load Balancing** (balanceamento de carga): serviços agrupam os containers de uma aplicação em 1 endereço IP e se encarregam de distribuir o tráfego aos containers.
- **Service Discovery** (descoberta de serviço): o kubernetes possui um registro DNS, de forma que a comunicação dentro do cluster pode ser realizada usando os nomes dos recursos, sem ter que saber o IP deles.
- **Rollout e Rollback**: é possível atualizar ou reverter a versão de uma aplicação sem causar tempo de indisponibilidade.
- **Gestão de configuração e segredos**: expor arquivos de configuração e senhas em repositórios é uma má prática. O kubernetes guarda essas informações no cluster e expõe-nas aos containers em forma de volume montado ou variável de ambiente. Essas informações podem ser atualizadas sem ter que reiniciar o container ou a aplicação contida nele.
- **Orquestração de armazenamento**: o kubernetes pode montar no container volumes locais ou provenientes de serviços em nuvem como AWS EBS, Azure Disk e GCE Persistent Disk.
- **Execução de jobs e batch**: um job cria um ou mais containers, executa comandos e elimina-os após completarem a tarefa designada.

Glossário

quinta-feira, 12 de julho de 2018 15:17

Cheatsheet (comandos principais) :

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

Deployment: construção alto-nível que define uma aplicação

Pods: são instâncias de um container em deployment.

<https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

A Pod is a group of one or more application containers (such as Docker or rkt) and includes shared storage (volumes), IP address and information about how to run them.

Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.

Ps.: Containers só devem ser lançados num único Pod se eles são altamente acoplados e precisam compartilhar recursos, como um disco.

Services: são endpoints que exportam portas para o mundo externo

A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.

Nodes:

A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster. The Master's automatic scheduling takes into account the available resources on each Node.

Every Kubernetes Node runs at least:

- Kubelet, a process responsible for communication between the Kubernetes Master and the Node; it manages the Pods and the containers running on a machine.
- A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container, and running the application.

Containers should only be scheduled together in a single Pod if they are tightly coupled and need to share resources such as disk.

Labels:

Kubectl: ferramenta para controlar clusters kubernetes

Kubectl expose deployment hello-minikube --type=NodePort

Expõe o deployment hello-minikube com um serviço do tipo "NodePort", que permite ter um endereço ip externo privado conectado na porta do aplicativo

Tipos de serviço: NodePort, ClusterIp, LoadBalancer

kubectl run: The run command creates a new deployment

kubectl get - list resources

kubectl describe - show detailed information about a resource

kubectl logs - print the logs from a container in a pod

kubectl exec - execute a command on a container in a pod

Exportar nome de um Pod para variavel de ambiente

```
export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}
{{.metadata.name}}{{"\n"}}{{end}}') echo Name of the Pod: $POD_NAME
```

Logs de um Container: (observação: soh precisa do POD_NAME caso só tenha um container no Pod)

```
kubectl logs $POD_NAME
```

Controlando um Container:

```
kubectl exec -ti $POD_NAME bash
```

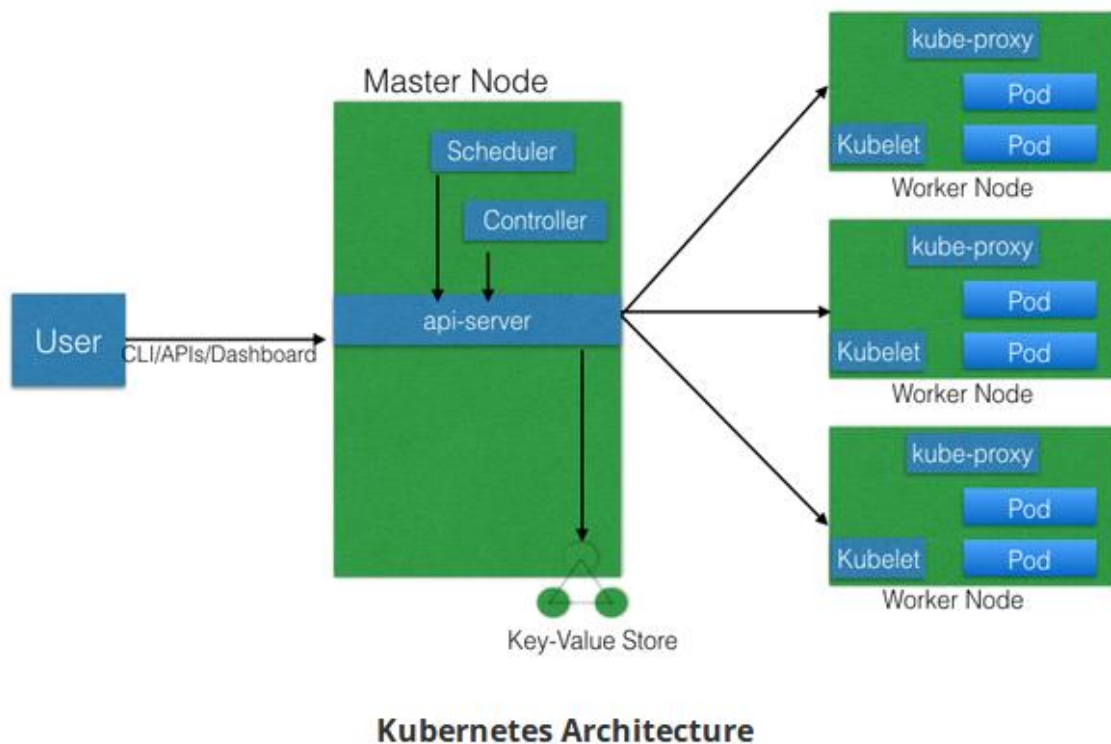
```
kubectl exec $POD_NAME env (lista variaveis de ambiente do container)
```

```
Exit (sai do container)
```

Architecture

quarta-feira, 18 de julho de 2018 13:54

- One or more **master nodes**
- One or more **worker nodes**
- Distributed key-value store, like **etcd**.



Master Node

A master node has the following components:

- API server
- Scheduler
- Controller manager
- etcd.

Sceduler

the **scheduler** schedules the work to different worker nodes, in terms of Pods and Services.

The scheduler has the resource usage information for each worker node. It also knows about the constraints that users/operators may have set, such as scheduling work on a node that has the label **disk==ssd** set

Control Manager

The **controller manager** manages different non-terminating control loops, which regulate the state of the Kubernetes cluster.

Each one of these control loops knows about the desired state of the objects it manages, and watches their current state through the API server. In a control loop, if the current state of the objects it manages does not meet the desired state, then the control loop takes corrective steps to make sure that the current state is the same as the desired state.

Etc

Etc is a distributed key value store that provides a reliable way to store data across a cluster of machines.

In Kubernetes, besides storing the cluster state, etc is also used to store configuration details such as subnets, ConfigMaps, Secrets, etc.

De <<https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+1T2018/courseware/474c6145f5e0436992a2d9ca212cd283/f0d11db04aff45479f54a3075d2286c1/?child=first>>

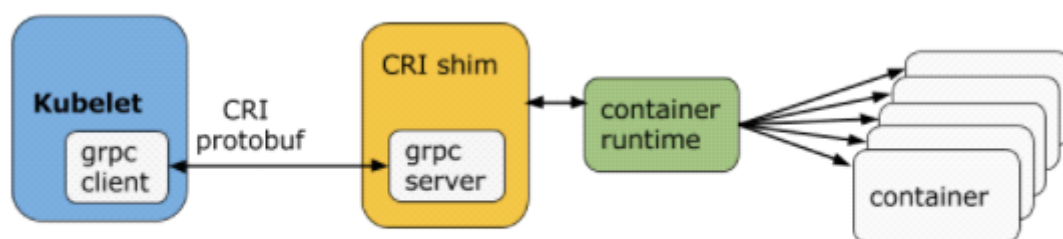
Kubelet

The kubelet is the primary “node agent” that runs on each node. The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn’t manage containers which were not created by Kubernetes.

Worker Node Components

A worker node has the following components:

- Container runtime (docker)
- kubelet
- kube-proxy.



Container Runtime Interface

Container Runtime Interface (CRI) integra o kubelet com o container runtime (docker)

Grpc:

https://en.wikipedia.org/wiki/Remote_procedure_call
https://en.wikipedia.org/wiki/Protocol_Buffers

kube-proxy is the network proxy which runs on each worker node and listens to the API server for each Service endpoint creation/deletion

De <<https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+1T2018/courseware/474c6145f5e0436992a2d9ca212cd283/f0d11db04aff45479f54a3075d2286c1/>>

Arquitetura

quarta-feira, 28 de novembro de 2018 15:05

Um cluster kubernetes consiste de dois tipos de máquinas:

- **Master:** gerencia o cluster. Possui responsabilidades como agendar, manter o estado desejado, escalar e realizar update de aplicações.
- **Node:** hospeda as aplicações. Os nós são controlados pelos masters.

Objetos do Kubernetes

O Kubernetes usa containers para implantar aplicações, mas há ainda camadas adicionais de abstração. Os administradores do cluster definem e interagem com instâncias baseadas no modelo de objetos do Kubernetes.

É importante explicar a funcionalidade dos seguintes objetos:

Pod

É a menor unidade dentro do cluster, residindo dentro de um Node. Um Pod geralmente representa um aplicação, possui um ou mais containers fortemente acoplados que compartilham armazenamentos (volumes) e endereço IP.

Não é recomendado o gerenciamento direto dos Pods. Para isso pode-se usar um objeto de alto nível que oferece recursos mais sofisticados de ciclo de vida e escalonamento, como o **deployment**.

Deployment

É uma construção alto-nível que define uma aplicação e facilita o gerenciamento do ciclo de vida de pods. Ele define um modelo de pod por meio de parâmetros como imagens dos containers, volumes, variáveis de ambiente. Define também parâmetros de controle como número de réplicas do Pod, comandos que executam durante a inicialização do pod, comandos que verificam se o Pod está "vivo" (liveness probe)

Service

Expõe os deployments no cluster por meio de endereços IPs. Isso elimina a necessidade de conhecer os IPs de cada Pod dentro do cluster.

Os clientes acessam os IPs desses services e o kubernetes é encarregado de distribuir o tráfego entre os pods desses services. Geralmente associa-se um service a um deployment.

Ingress e Ingress Controller

O Ingress atua como uma coleção de regras de roteamento em HTTP para que usuários externos possam acessar os serviços presentes no cluster. O Ingress Controller é uma aplicação responsável por executar essas regras. O Controller pode não ser criado junto com o cluster. Nesse caso é possível usar o [ingress-nginx](https://kubernetes.io/docs/concepts/services-networking/ingress-nginx/), suportado oficialmente pelo Projeto Kubernetes, ou outra solução.

ConfigMap e Secret

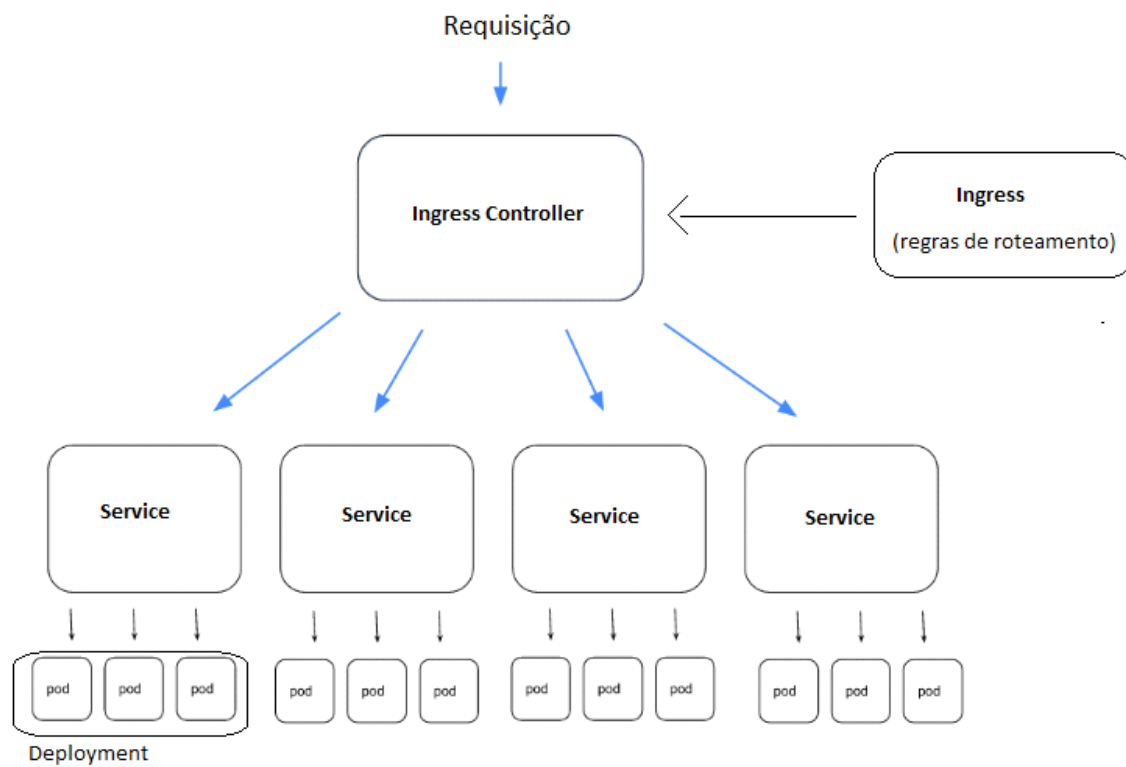
O Kubernetes permite desacoplar a configuração das aplicações containerizadas, aumentando a portabilidade e segurança. ConfigMaps e Secrets são arquivos de configuração que ficam guardados em um local central do cluster e podem ser acessados pelos Pods. Esses arquivos podem ser alterados a qualquer momento. A diferença entre esses dois tipos de configuração é que ConfigMaps são usados para dados não confidenciais e são armazenados em puro texto, enquanto Secrets são usados para dados como chaves e credenciais.

Secrets e ConfigMaps podem ser expostos aos Pods por meio de variáveis de ambiente ou volumes.

Namespace

Segmentação do cluster, permitindo a existência de deploys, services e outros resources com o

mesmo nome em diferentes namespaces. Além disso é possível aplicar configurações específicas a namespaces, como por exemplo controle de acesso e limites de recursos computacionais.



Communication

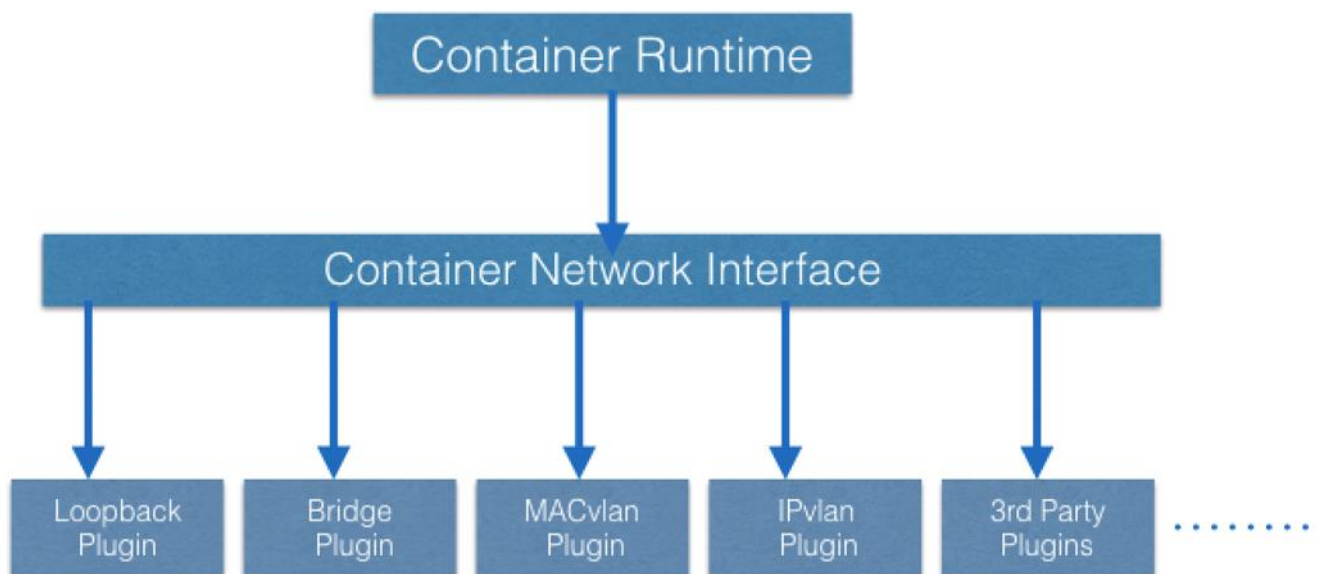
quarta-feira, 18 de julho de 2018 17:43

To have a fully functional Kubernetes cluster, we need to make sure of the following:

- A unique IP is assigned to each Pod
- Containers in a Pod can communicate to each other
- The Pod is able to communicate with other Pods in the cluster
- If configured, the application deployed inside a Pod is accessible from the external world.

De <<https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+1T2018/courseware/474c6145f5e0436992a2d9ca212cd283/f0d11db04aff45479f54a3075d2286c1/?child=first>>

Kubernetes uses CNI to assign the IP address to each Pod.



Container Network Interface (CNI)

The container runtime offloads the IP assignment to CNI, which connects to the underlying configured plugin, like Bridge or MACvlan, to get the IP address. Once the IP address is given by the respective plugin, CNI forwards it back to the requested container runtime.

De <<https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+1T2018/courseware/474c6145f5e0436992a2d9ca212cd283/f0d11db04aff45479f54a3075d2286c1/?child=first>>

Container-to-Container Communication Inside a Pod

With the help of the underlying host operating system, all of the container runtimes generally create an isolated network entity for each container that it starts. On Linux, that entity is referred to as a **network namespace**. These network namespaces can be shared across containers, or with the host operating system.

Inside a Pod, containers share the network namespaces, so that they can reach to each other via localhost.

Pod-to-Pod Communication Across Nodes

- Routable Pods and nodes, using the underlying physical infrastructure, like Google Kubernetes Engine
- Using Software Defined Networking, like [Flannel](#), [Weave](#), [Calico](#), etc.

Communication Between the External World and Pods

By exposing our services to the external world with **kube-proxy**, we can access our applications from outside the cluster

Instalação

quarta-feira, 18 de julho de 2018 19:09

[INSTALAR KUBERNETES FROM SCRATCH](#)

Kubernetes Installation Tools/Resources

- **kubeadm**

[kubeadm](#) is a first-class citizen on the Kubernetes ecosystem. It is a secure and recommended way to bootstrap the Kubernetes cluster. It has a set of building blocks to setup the cluster, but it is easily extendable to add more functionality. Please note that kubeadm does not support the provisioning of machines.

- **KubeSpray**

With [KubeSpray](#) (formerly known as Kargo), we can install Highly Available Kubernetes clusters on AWS, GCE, Azure, OpenStack, or bare metal. KubeSpray is based on Ansible, and is available on most Linux distributions. It is a [Kubernetes Incubator](#) project.

- **Kops**

With [Kops](#), we can create, destroy, upgrade, and maintain production-grade, highly-available Kubernetes clusters from the command line. It can provision the machines as well. Currently, AWS is officially supported. Support for GCE and VMware vSphere are in alpha stage, and other platforms are planned for the future.

De <<https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+1T2018/courseware/6958a7f158624bb5a62a89eb70f6f873/1711d30f316541a7bb18b2a548872ccd/?child=first>>

minikube linux

segunda-feira, 26 de novembro de 2018 12:20

```
sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
sudo touch /etc/apt/sources.list.d/kubernetes.list
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl
```

```
sudo apt-get update
sudo apt-get install virtualbox
```

```
sudo apt-get install curl
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.28.2/minikube-linux-amd64
&& chmod +x minikube && sudo mv minikube /usr/local/bin/
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add - && \
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list && \
sudo apt-get update -q && \
sudo apt-get install -qy kubelet=1.9.8> kubectl=<version> kubeadm=1.9.8
```

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.9.8/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

Cluster Kubernetes local com Minikube

segunda-feira, 3 de setembro de 2018 11:07

Minikube é uma ferramenta que facilita executar o Kubernetes localmente. Minikube cria um cluster Kubernetes com um único nó dentro de uma VM do computador.

Instalar Minikube

Requisitos

- 1) Virtualbox: <https://www.virtualbox.org/wiki/Downloads>
 - 2) kubectl:
 - a. Baixar <https://storage.googleapis.com/kubernetes-release/release/v1.12.0/bin/windows/amd64/kubectl.exe>
 - b. adicionar pasta à variável PATH do Windows
- Instruções completas: <https://kubernetes.io/docs/user-guide/kubectl/>
- 3) minikube
 - a. Baixar e usar o instalador "minikube-installer.exe" dessa página: <https://github.com/kubernetes/minikube/releases>
 - b. Para verificar instalação, abrir o cmd e digitar "minikube version"

```
PS C:\Users\oval> minikube version
minikube version: v0.20.0
```

Iniciar minikube:

```
minikube config set vm-driver virtualbox
minikube start
```

Para verificar se o minikube iniciou:

```
minikube status
```

```
PS C:\Users\oval> minikube status
minikube: Running
```

Agora verificamos se o minikube pôde configurar a ferramenta **kubectl**:

```
kubectl cluster-info
```

```
PS C:\Users\oval> kubectl cluster-info
Kubernetes master is running at https://192.168.99.100:8443
```

Kubectl é uma ferramenta de linha de comando que possibilita controlar os recursos e componentes de um cluster kubernetes

```
PS C:\Users\oval> kubectl get nodes
NAME      STATUS    AGE       VERSION
minikube  Ready     1d        v1.10.0
```

Criando um Deployment

O comando a seguir cria um Deployment com o nome "helloscopus", imagem docker "oval/helloscopus:0.4", abre a porta 8090 para o container enviar e receber tráfego, e com número

de Pods desejado igual a 3:

```
kubectl run helloscopus --image= vval/helloscopus:0.4 --port=8090 --replicas=3
```

O comando deverá retornar *"deployment "helloscopus" created"*

Para visualizar o Deployment criado:

```
kubectl get deploy
```

```
PS C:\Users\vval> kubectl get deploy
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
helloscopus    3         3         3             3           3h
```

Pode-se verificar que os 3 Pods foram criados e estão disponíveis (coluna "AVAILABLE")

É possível também ver cada Pod individualmente:

```
kubectl get pods
```

```
PS C:\Users\vval> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
helloscopus-1199117723-5h75t       1/1     Running   0           3h
helloscopus-1199117723-b1s1f       1/1     Running   0           3h
helloscopus-1199117723-d1f80       1/1     Running   0           3h
```

Testando a aplicação

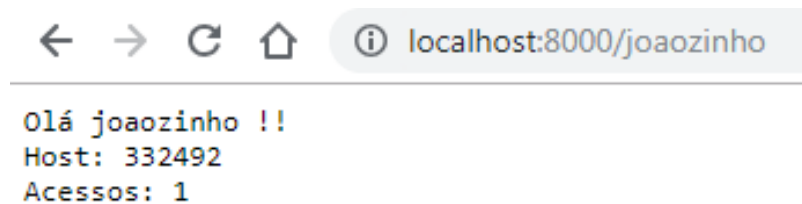
O comando port-forward redireciona as conexões de uma porta do computador local para a porta de um Pod.

```
kubectl port-forward NomeDoPod PortaLocal:PortaPod
```

exemplo:

```
kubectl port-forward helloscopus-1199117723-5h75t 8000:8090
```

Após abrir a conexão com um dos Pods, podemos testar a porta local com uma chamada HTTP para <http://localhost:8000>



The screenshot shows a web browser address bar with the URL `localhost:8000/joaozinho`. Below the address bar, the browser displays the response: `Olá joaozinho !!`, `Host: 332492`, and `Acessos: 1`.

Um comando muito útil e simples para debugging é:

```
kubectl logs NomeDoPod
```

Esse comando exibe os logs gerados pelo(s) container(s) do Pod.

```

=====|=====|=====|
:: Spring Boot ::      <v2.1.0.RELEASE>

2018-12-05 13:28:46.257 INFO 1 --- [main] b.s.e.h.HelloScopusApplicationKt : Starting Hello
2018-12-05 13:28:46.268 INFO 1 --- [main] b.s.e.h.HelloScopusApplicationKt : No active pro
2018-12-05 13:28:48.831 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initia
2018-12-05 13:28:48.919 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting serv
2018-12-05 13:28:48.921 INFO 1 --- [main] org.apache.catalina.core.StandardEngine : Starting Serv
2018-12-05 13:28:48.983 INFO 1 --- [main] o.a.catalina.core.AppLifecycleListener : The APR based
[usr/lib/jvm/java-1.8-openjdk/jre/lib/amd64/server:/usr/lib/jvm/java-1.8-openjdk/jre/lib/amd64:/usr/lib/jvm/j
2018-12-05 13:28:49.267 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing
2018-12-05 13:28:49.271 INFO 1 --- [main] o.s.web.context.ContextLoader : Root WebAppli
2018-12-05 13:28:49.426 INFO 1 --- [main] o.s.b.w.servlet.ServletRegistrationBean : Servlet displa
2018-12-05 13:28:49.435 INFO 1 --- [main] o.s.b.w.servlet.FilterRegistrationBean : Mapping filte
2018-12-05 13:28:49.436 INFO 1 --- [main] o.s.b.w.servlet.FilterRegistrationBean : Mapping filte
2018-12-05 13:28:49.437 INFO 1 --- [main] o.s.b.w.servlet.FilterRegistrationBean : Mapping filte
2018-12-05 13:28:49.437 INFO 1 --- [main] o.s.b.w.servlet.FilterRegistrationBean : Mapping filte
2018-12-05 13:28:50.913 INFO 1 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing
2018-12-05 13:28:52.097 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat starte
2018-12-05 13:28:52.108 INFO 1 --- [main] b.s.e.h.HelloScopusApplicationKt : Started Hello
2018-12-05 18:33:16.153 INFO 1 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing
2018-12-05 18:33:16.153 INFO 1 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet : Initializing
2018-12-05 18:33:16.173 INFO 1 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet : Completed ini
PS C:\Users\uval>
```

Deployment na forma declarativa

Além de criar Deployments por meio do comando "kubectl run", também é possível usar um arquivo no formato JSON ou YAML com as configurações desejadas.

É importante notar que essa forma de configuração não é exclusiva aos Deployments, sendo usada em outros objetos do Kubernetes como Pods, Services, Secrets, etc.

O exemplo a seguir descreve um Deployment com as mesmas características básicas que o criado anteriormente via "kubectl run" :

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: helloscopus
spec:
  selector:
    matchLabels:
      app: helloscopus
  replicas: 3
  template:
    metadata:
      labels:
        app: helloscopus
    spec:
      containers:
        - name: helloscopus
          image: vval/helloscopus:0.4
          ports:
            - containerPort: 8090
```

O texto acima deve ser salvo em um arquivo yaml (Ex.: deployment.yaml)

Antes de realizar o deploy deletamos o deploy antigo:

```
kubectl delete deploy helloscopus
```

Em seguida, para criar o novo deployment:

```
kubectl apply -f deployment.yaml
```


Criando um Service

Anteriormente conseguimos acessar a aplicação por meio de roteamento de portas. Entretanto esse método não disponibiliza a aplicação para a Internet ou nem mesmo para outras aplicações do cluster.

No Kubernetes os Pods não são imortais, de forma que quando morrem, um Pod novo é criado para substituí-lo. Isso impede que as aplicações possam usar os IPs dos Pods de forma confiável.

Um Service expõe um conjunto de Pod por meio de um endereço IP. A forma mais comum de selecionar os Pods para um service é por meio do conceito de **LabelSelectors**. O exemplo a seguir é usado para criar um serviço do tipo **NodePort** com nome "helloscopus". O atributo spec.selector seleciona os pods que tiverem como **Label** "app: helloscopus". Como pode ser visto na seção anterior, essa Label foi declarada na configuração do Deployment.

```
apiVersion: v1
kind: Service
metadata:
  name: helloscopus
spec:
  selector:
    app: helloscopus
  type: NodePort
  ports:
    - port: 80
      targetPort: 8090
```

Criando o Service:

```
kubectl apply -f service.yaml
```

Visualizar os serviços do cluster:

```
kubectl get services
```

```
PS C:\Users\oval> kubectl get svc
NAME            CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
helloscopus     10.0.0.169      <nodes>          80:31033/TCP     6s
kubernetes      10.0.0.1        <none>           443/TCP          2d
```

Por se tratar de um serviço do tipo **NodePort** o helloscopus possui além de um IP interno ao cluster (coluna "CLUSTER-IP"), um IP externo (coluna "EXTERNAL-IP") do tipo <nodes>, que indica que o serviço é acessado externamente por meio do IP de um dos nós do kubernetes. A coluna "PORT(S)" refere-se às portas interna (80) e externa (31033).

Para testar a aplicação usando o Service criado, deve-se primeiro obter o endereço de um dos nós do cluster. No nosso caso, como estamos usando o minikube:

```
minikube ip
```

Em seguida, acessar o IP do minikube na porta externa do Service helloscopus:



The screenshot shows a web browser window with the address bar displaying "192.168.99.100:31033/maria". The page content shows a greeting "Olá maria !!", the host IP "Host: 119984", and the number of accesses "Acessos: 1".

Tipos de Service

- *ClusterIP* (default) - Expõe o Serviço por meio de um IP interno ao cluster. Esse tipo faz com que o serviço seja acessível apenas dentro do cluster.
- *NodePort* - Abre uma porta em cada nó do cluster e qualquer tráfego enviado a essa porta é redirecionado ao serviço. Os serviços podem ser acessados por meio do endereço `<IP_do_nó>:<Porta>`.
- *LoadBalancer* - Depende do suporte do serviço cloud que hospeda o cluster (como AWS, Google Cloud, etc..). Um serviço desse tipo cria um load balancer externo com um IP fixo e o atribui ao serviço.

Service Discovery

Dada a natureza dinâmica do Kubernetes, não é recomendado usar os IPs dos Services diretamente. O Kubernetes fornece dois métodos para obter os endereços dos Services:

Variável de Ambiente

Ao criar um Pod, o Kubernetes injeta nele variáveis de ambiente do tipo:

```
{nome do Service em maiúscula}_SERVICE_HOST  
{nome do Service em maiúscula}_SERVICE_PORT
```

Por exemplo, para acessar o serviço "helloscopus":

[http://\\$HELLOSCOPUS_SERVICE_HOST:\\$HELLOSCOPUS_SERVICE_PORT/](http://$HELLOSCOPUS_SERVICE_HOST:$HELLOSCOPUS_SERVICE_PORT/)

Um problema com esse método é que um Pod só consegue acessar variáveis de ambiente de serviços criados antes dele.

DNS

É possível instalar um servidor DNS no Kubernetes. As distribuições de Kubernetes geralmente incluem um servidor desse tipo. O servidor DNS observa se novos Services são criados e cria registros DNS. Esses registros seguem a seguinte especificação:

```
{nome do service}.{namespace}.svc.cluster.local
```

Exemplo:

<http://my-svc.my-namespace.svc.cluster.local:80/>

Kubernetes Object Model

terça-feira, 24 de julho de 2018 09:56

- Pods
- ReplicaSets
- Deployments
- Namespaces
- Secret
- Outros

To create an object, we need to provide the **spec** field to the Kubernetes API server.

```
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:1.7.9
          ports:
            - containerPort: 80
```

With the **apiVersion** field in the example above, we mention the API endpoint on the API server which we want to connect to.

With the **kind** field, we mention the object type.

With the **metadata** field, we attach the basic information to objects, like the name.

With **spec**, we define the desired state of the deployment. In our example, we want to make sure that, at any point in time, at least 3 Pods are running, which are created using the Pods Template defined in **spec.template**.

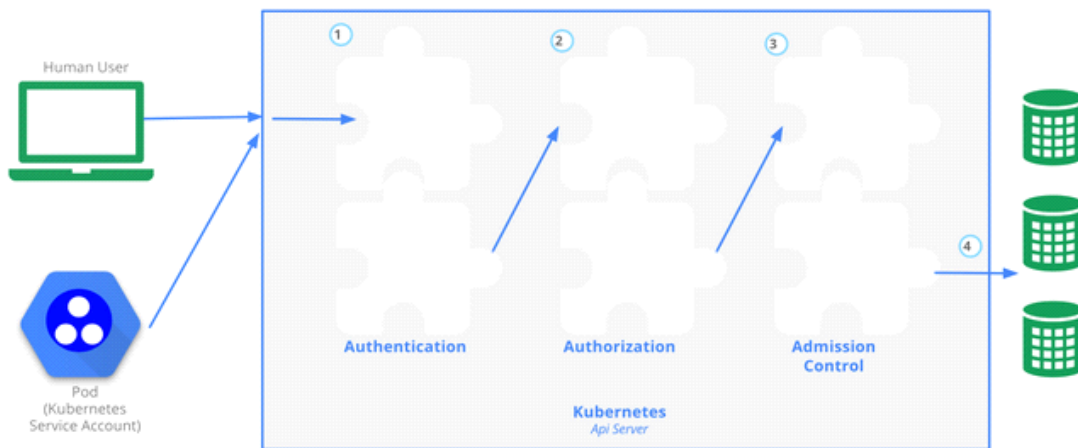
In **spec.template.spec**, we define the desired state of the Pod.

Autenticação, Autorização, Controle de Admissão

quinta-feira, 26 de julho de 2018

11:01

- **Authentication**
Logs in a user.
- **Authorization**
Authorizes the API requests added by the logged-in user.
- **Admission Control**
Software modules that can modify or reject the requests based on some additional checks, like **Quota**.



Accessing the API

Autenticação

quinta-feira, 26 de julho de 2018 11:03

O kubernetes tem dois tipos de usuário:

- **Normal Users**
They are managed outside of the Kubernetes cluster via independent services like User/Client Certificates, a file listing usernames/passwords, Google accounts, etc.
- **Service Accounts**
With Service Account users, in-cluster processes communicate with the API server to perform different operations. Most of the Service Account users are created automatically via the API server, but they can also be created manually. The Service Account users are tied to a given Namespace and mount the respective credentials to communicate with the API server as Secrets.

Para autenticação são usados diferentes módulos de autenticação:

- **Static Token File:** Um arquivo é passado por meio de `--token-auth-file=SOMEFILE`. Os tokens não possuem validade.
- **Static Password File:** Opção `--basic-auth-file=SOMEFILE`. Semelhante ao Token, dura pra empre e só pode ser mudado restartando o API Server
- **Client Certificate:** Opção `--client-ca-file=SOMEFILE`. Autoridades certificadoras presentes no arquivo validam o certificado.
- **Service Account Token:** É usado por serviços e configurado automaticamente na sua criação.

Existem muitos outros módulos:

<https://kubernetes.io/docs/reference/access-authn-authz/authentication/#authentication-strategies>

Múltiplos autenticadores podem ser usados.

Autorização

quinta-feira, 26 de julho de 2018 10:52

Autorização permite:

- Secure your cluster by granting privileged operations (accessing secrets, for example) only to admin users.
- Force user authentication in your cluster.
- Limit resource creation (such as pods, persistent volumes, deployments) to specific namespaces. You can also use quotas to ensure that resource usage is limited and under control.
- Have a user only see resources in their authorized namespace. This allows you to isolate resources within your organization (for example, between departments).

Assim com em Autenticação, existem vários módulos de Autorização:

Node Authorizer

Attribute-Based Access Control (ABAC) Authorizer

Webhook Authorizer

O módulo mais usado é o RBAC:

Role-Based Access Control (RBAC) Authorizer

- 1) Definir permissões por meio do objeto **Role**

Exmeplo:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: lfs158
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

- 2) Vincular usuário(s) a Roles por meio de **RoleBinding**

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: pod-read-access
  namespace: lfs158
subjects:
- kind: User
  name: victor
```

```
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Tutorial mais completo de como configurar RBAC:

<https://docs.bitnami.com/kubernetes/how-to/configure-rbac-in-your-kubernetes-cluster/>

Dashboard

quarta-feira, 1 de agosto de 2018 16:38

<https://github.com/kubernetes/dashboard>

<https://github.com/kubernetes/dashboard/wiki/Creating-sample-user>

minikube

kubect! proxy

Acessar esta URL:

<http://localhost:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy/#!/workload?namespace=default>

Liveliness

terça-feira, 7 de agosto de 2018 12:06

<https://medium.com/spire-labs/utilizing-kubernetes-liveness-and-readiness-probes-to-automatically-recover-from-failure-2fe0314f2b2e>

Readiness Probes e Liveness Probes

Quando um deployment for atualizado o Kubernetes sabe quando um novo Pod está **rodando** antes de terminar um Pod antigo. O Kubernetes também sabe quando um Pod parou de rodar para que possa reiniciar. Entretanto o Kubernetes **não sabe quando a sua aplicação está funcionando**. Existe um intervalo de tempo entre o instante em que o container está pronto e o instante em que a aplicação está de fato no ar. Ainda, é possível que a aplicação pare de funcionar sem que o container falhe.

Readiness Probe é passivo. Liveness Probe é ativo.

Em Readiness Probe o Kubernetes não irá enviar tráfego até que o probe seja bem sucedido. Quando um deployment é atualizado os Pods antigos irão funcionar até que os novos Pods confirmem que estão prontos.

Em Liveness Probe o Kubernetes irá tentar restartar o container caso a verificação falhe

Existem 3 tipos de Probes:

- Liveness command
- Liveness HTTP request
- TCP Liveness Probe.

Comando:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
  livenessProbe:
    exec:
      command:
      - cat
      - /tmp/healthy
    initialDelaySeconds: 3
    periodSeconds: 5
```

HTTP Request:

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
    httpHeaders:
      - name: X-Custom-Header
        value: Awesome
    initialDelaySeconds: 3
    timeoutSeconds: 1
    periodSeconds: 3
```

TCP (attempts to open the TCP Socket to the container which is running the application) :

```
livenessProbe:
  tcpSocket:
    port: 8080
    initialDelaySeconds: 15
    periodSeconds: 20
```

Debugging

sexta-feira, 3 de agosto de 2018 20:14

Ver qual imagem o deployment está usando

```
kubectl get deployments -l app=facerecognition -o wide
```

Logs

```
kubectl logs POD
```

Executar comando

```
kubectl exec -it POD COMANDO
```

Exemplo:

```
kubectl exec -it facerecognition-7f7d6456bb-29m7l /bin/bash
```

(abre o bash de um dos containers do serviço facerecognition, possibilitando o seu controle)

Proxy

O comando **kubectl proxy** permite navegar a Kubernetes API e acessar serviços diretamente, por meio de seus IPs internos, sem passar por, por exemplo, o Ingress

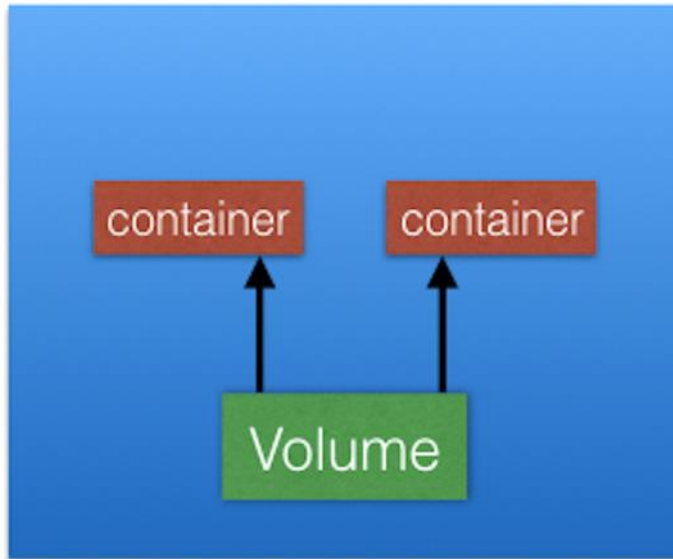
Port-Forwarding

Volume

terça-feira, 7 de agosto de 2018 15:58

Volumes permitem persistência de dados.

Um Volume se liga a um Pod e é compartilhado entre os seus containers



Existem diferentes tipos de volume, que podem ser conferidos aqui:

<https://kubernetes.io/docs/concepts/storage/volumes/>

Usando variáveis de ambiente e volumes no Kubernetes por meio de ConfigMaps e Secrets

terça-feira, 24 de julho de 2018 14:33

O programa deve utilizar variáveis de ambiente ou arquivos de configuração ao invés de deixar em aberto (variáveis hardcoded) informações confidenciais ou que dependem de configuração.

O Kubernetes permite desacoplar a configuração das aplicações containerizadas, aumentando a portabilidade e segurança. ConfigMaps e Secrets são arquivos de configuração contendo pares chave-valor e que ficam guardados em um local central do cluster e podem ser acessados pelos Pods.

A diferença entre esses dois tipos de configuração é que ConfigMaps são usados para dados não confidenciais e são armazenados em puro texto, enquanto Secrets são usados para dados como chaves e credencias.

Secrets e ConfigMaps podem ser expostos aos Pods por meio de variáveis de ambiente ou volumes.

Exemplos de como usar variável de ambiente:

- Em Java:

```
db.username = System.getenv("SQL_DB_USERNAME")
db.password = System.getenv("SQL_DB_PASS")
```

- Em Python:

```
import os
username=os.getenv("SQL_DB_USERNAME")
password=os.getenv("SQL_DB_PASS")
```

Criando objetos do tipo ConfigMap

Objetos ConfigMaps podem ser criados de duas maneiras:

- 1) Arquivo yaml:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-de-regiao
  namespace: default
data:
  pais: Brasil
  idioma: PT-BR
  moeda: BRL
```

Assim como outros objetos do Kubernetes, para criar o objeto usando um arquivo .yaml:

```
kubectl apply -f config-de-regiao
```

Para examinar o ConfigMap criado:

```
kubectl get configmap exemplo-config -o yaml
```

```

PS C:\Users\oval> kubectl apply -f configmap.yaml
configmap "config-de-regiao" configured
PS C:\Users\oval> kubectl get configmap config-de-regiao -o yaml
apiVersion: v1
data:
  idioma: PT-BR
  moeda: BRL
  pais: Brasil
kind: ConfigMap
metadata:
  annotations:
    kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"idioma":"PT-BR","moeda":"BRL","pais":"B
  creationTimestamp: 2018-12-11T20:51:00Z
  name: config-de-regiao
  namespace: default
  resourceVersion: "707911"
  selfLink: /api/v1/namespaces/default/configmaps/config-de-regiao
  uid: 773d15b5-fd86-11e8-b7de-08002780e198

```

2) Também é possível a criação de ConfigMaps a partir de arquivos texto.

Exemplo de um arquivo de configuração config-de-regiao.txt:

```

pais: Brasil
idioma: PT-BR
moeda: BRL

```

O seguinte comando criará o ConfigMap a partir do arquivo:

```
kubectl create configmap config-de-regiao2 --from-file=config-de-regiao.txt
```

Examinando o ConfigMap criado:

```

PS C:\Users\oval> kubectl create configmap config-de-regiao2 --from-file=config-de-regiao.txt
configmap "config-de-regiao2" created
PS C:\Users\oval> kubectl get configmap config-de-regiao2 -o yaml
apiVersion: v1
data:
  config-de-regiao.txt: |-
    pais: Brasil
    idioma: PT-BR
    moeda: BRL
kind: ConfigMap
metadata:
  creationTimestamp: 2018-12-11T21:06:29Z
  name: config-de-regiao2
  namespace: default
  resourceVersion: "708990"
  selfLink: /api/v1/namespaces/default/configmaps/config-de-regiao2
  uid: a131b3fe-fd88-11e8-b7de-08002780e198

```

A diferença entre as duas maneiras de gerar um ConfigMap é que **a primeira gera múltiplos pares chave-valor**, enquanto que **a segunda gera um único par**, cuja chave é o nome do arquivo e o valor o seu conteúdo.

Isso pode ser observado a seguir na coluna DATA:

```

PS C:\Users\oval> kubectl get configmap
NAME          DATA      AGE
config-de-regiao  3         28m
config-de-regiao2 1         13m

```

Essa característica faz com que seja preferível usar o **primeiro método em variáveis de ambiente**, e o **segundo em volume**.

Criando objetos do tipo Secret

No caso dos Secrets, os valores das variáveis devem estar em base64.

Exemplo de um objeto **Secret**:

```

apiVersion: v1
kind: Secret
metadata:
  name: senha-bd
type: Opaque
data:

```

BD_PASS: c2VuaGExMjM=

- Encoding para base64:

PowerShell:

```
[System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("qwerty123"))
```

Resultado: **cXdlcnR5MTIz**

- Decoding:

PowerShell:

```
[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String("cXdlnR5MTIz"))
```

Resultado: **qwerty123**

Criando o Secret no cluster:

```
kubectl apply -f senha.yaml
```

Verificand o Secret criado:

```
kubectl get secret senha-bd -o yaml
```

```
PS C:\Users\voval> kubectl get secret senha-bd -o yaml
apiVersion: v1
data:
  BD_HOST: aG9zdGRvYmFuY29kZWRhZG9zLnNubQ==
  BD_PASS: c2UuaGEzMjM=
  BD_PORT: MzMwNg==
  BD_USER: dXN1YXJpbw==
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration:
      {"apiVersion":"v1","data":{"BD_HOST":"aG9zdGRvYmFuY29kZWRhZG9zLnNubQ==","BD_PASS":"c2UuaGEzMjM=","BD_PORT":"MzMwNg==","BD_USER":"dXN1YXJpbw=="},"kind":"Secret","metadata":{"name":"senha-bd","namespace":"default"},"type":"Opaque"}
```

Usando ConfigMaps e Secrets nos Pods

Com a ConfigMap e o Secret criados agora devemos configurar os Pods para usá-los.

Essa configuração será feita em um **Deployment** de uma aplicação:

apiVersion: apps/v1beta1

kind: Deployment

```
metadata:
```

```
name: helloscopus
```

spec:

selector:

matchLabels:

app: helloscopus

replicas: 3

template:

```

metadata:
  labels:
    app: helloscopus
spec:
  containers:
  - name: helloscopus
    image: vval/helloscopus:0.4
    ports:
    - containerPort: 8090
    volumeMounts:
    - name: volume-config
      mountPath: /configuracoes
    envFrom:
    - configMapRef:
        name: config-de-regiao
    - secretRef:
        name: senha-bd
    volumes:
    - name: volume-config
      configMap:
        name: config-de-regiao2

```

Os pontos a ficar atento para o arquivo acima são:

- volumeMounts: especifica volumes a serem montados.
- volumes: declara os volumes a serem usados pelo "volumeMounts" e a origem, como "configMap", "secret", "emptyDir", "awsElasticBlockStore", e outros.
- envFrom: gera variáveis de ambiente a partir

Para testar vamos "entrar" dentro de um dos Pods do Deployment configurado, por meio do comando "kubectl exec":

```
kubectl exec -it {nome de um pod} bin/sh
```

Usando o comando "ls", vemos que o volume "configuracoes" foi montado com sucesso:

```

/ # ls
bin          etc          lib          proc         sbin         tmp
configuracoes helloscopus.jar media        root         srv          usr
dev          home         mnt         run          sys          var
/ #

```

Vamos acessar o conteúdo do arquivo contido dentro do volume:

```

cd configuracoes
ls configuracoes
cat config-de-regiao.txt

```

```

/ # cd configuracoes
/configuracoes # ls
config-de-regiao.txt
/configuracoes #
/configuracoes # cat config-de-regiao.txt
pais: Brasil
idioma: PT-BR
moeda: BRL/configuracoes #

```

Verificamos ainda que podemos acessar as variáveis de ambiente proveniente do ConfigMap e Secret:

```

/configuracoes # echo $idioma
PT-BR
/configuracoes # echo $moeda
BRL
/configuracoes # echo $BD_HOST
hostdobancodedados.com
/configuracoes # echo $BD_PORT
3306

```

É importante notar que volumes são alterados dinamicamente ao atualizar um configmap ou secret, mas variáveis de ambiente não, necessitando matar o Pod e criar um novo.

Observação

É possível que a aplicação também rode em ambientes fora de clusters Kubernetes, como para testes. Pode ser necessário configurar variáveis de ambiente na ferramenta de CI/CD usada, como GitLab-CI, Jenkins, Bamboo, etc.
Exemplo: No caso do GitLab-CI, é possível setar as variáveis em Settings->CI/CD->Variables.

Variables ?

Variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use variables for passwords, secret keys, or whatever you want.

AWS_ACCESS_KEY_ID	*****	Protected	<input checked="" type="checkbox"/>	⊖
aws_access_key_id1	*****	Protected	<input checked="" type="checkbox"/>	⊖
AWS_SECRET_ACCESS_KEY	*****	Protected	<input checked="" type="checkbox"/>	⊖
aws_secret_access_key1	*****	Protected	<input checked="" type="checkbox"/>	⊖
CI_USER_NAME	*****	Protected	<input checked="" type="checkbox"/>	⊖
CI_USER_PASS	*****	Protected	<input checked="" type="checkbox"/>	⊖
dbname_env	*****	Protected	<input checked="" type="checkbox"/>	⊖
DOCKER_AUTH_CONFIG	*****	Protected	<input checked="" type="checkbox"/>	⊖
host_env	*****	Protected	<input checked="" type="checkbox"/>	⊖
password_env	*****	Protected	<input checked="" type="checkbox"/>	⊖
port_env	*****	Protected	<input checked="" type="checkbox"/>	⊖
s3_bucket_env	*****	Protected	<input checked="" type="checkbox"/>	⊖
teste	*****	Protected	<input checked="" type="checkbox"/>	⊖
user_env	*****	Protected	<input checked="" type="checkbox"/>	⊖
Input variable key	Input variable value	Protected	<input checked="" type="checkbox"/>	

Save variables

Reveal values

Leitura Adicional

<https://kubernetes.io/docs/tutorials/configuration/>
<https://kubernetes.io/docs/concepts/configuration/secret/>
<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>
<https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/>
<https://kubernetes.io/docs/concepts/storage/volumes/>
<https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/>

Kubernetes CSI

sexta-feira, 18 de janeiro de 2019 16:50

Understanding the Container Storage Interface (CSI)

De <<https://medium.com/google-cloud/understanding-the-container-storage-interface-csi-ddbeb966a3b>>

Continuous Integration / Continuous Build

quarta-feira, 25 de julho de 2018 11:42

1. **Continuous Build:** automated build on every check-in, which is the first step, but does nothing to prove functional integration of new code.
2. **Continuous Integration (CI):** automated build and execution of at least unit tests to prove integration of new code with existing code, but preferably integration tests (end-to-end).
3. **Continuous Deployment (CD):** automated deployment when code passes CI at least into a test environment, preferably into higher environments when quality is proven either via CI or by marking a lower environment as PASSED after manual testing. I.E., testing may be manual in some cases, but promoting to next environment is automatic.
4. **Continuous Delivery:** automated publication and release of the system into production. This is CD into production plus any other configuration changes like setup for A/B testing, notification to users of new features, notifying support of new version and change notes, etc.

De <<https://stackoverflow.com/questions/28608015/continuous-integration-vs-continuous-delivery-vs-continuous-deployment>>

Ferramentas

quarta-feira, 25 de julho de 2018

11:43

Jenkins
Gitlab CI

Criação de Cluster no AWS com kops

quinta-feira, 26 de julho de 2018 09:33

A ferramenta kops facilita a criação e manutenção de cluster Kubernetes no AWS por meio de linha de comando.

1) É necessário instalar:

- Kops CLI Tool

```
wget -O kops https://github.com/kubernetes/kops/releases/download/\$\(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4\)/kops-linux-amd64  
chmod +x ./kops  
sudo mv ./kops /usr/local/bin/
```

- Kubectl

```
wget -O kubectl https://storage.googleapis.com/kubernetes-release/release/\$\(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt\)/bin/linux/amd64/kubectl  
chmod +x ./kubectl  
sudo mv ./kubectl /usr/local/bin/kubectl
```

- AWS CLI Tols

```
sudo apt-get install pip  
pip install awscli
```

2) Criar um usuário dedicado para o kops no AWS IAM, com as permissões:

- AmazonEC2FullAccess
- AmazonRoute53FullAccess
- AmazonS3FullAccess
- IAMFullAccess
- AmazonVPCFullAccess

Add user



Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

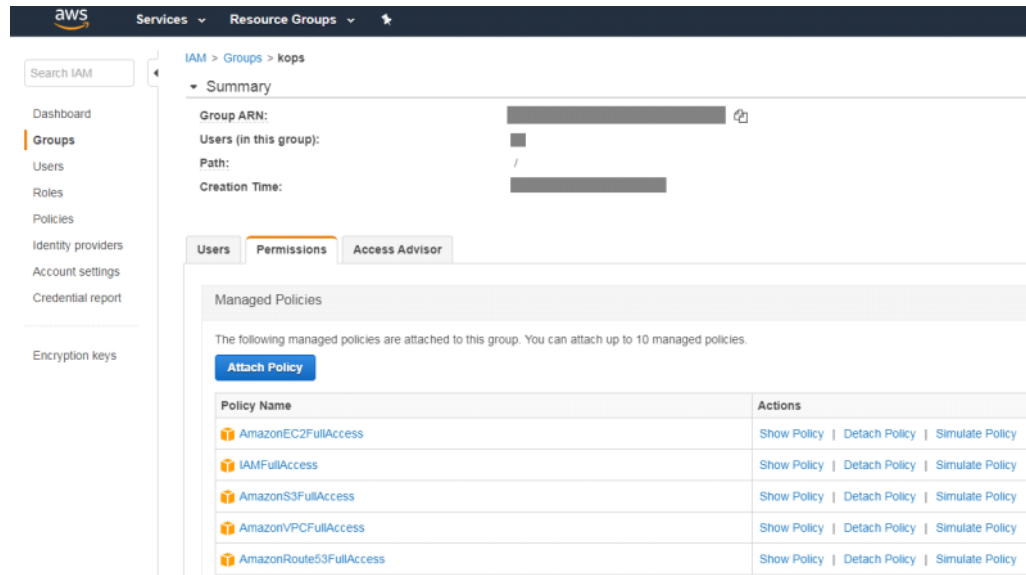
Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

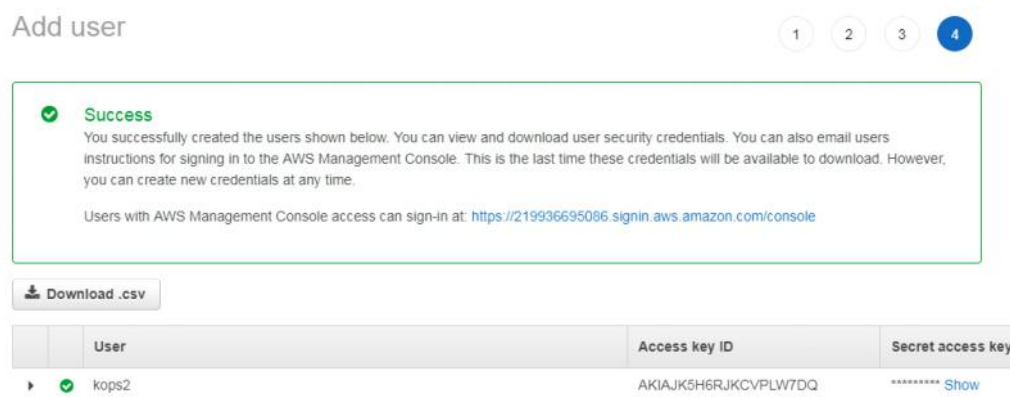
- Access type* ☒ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

Ao selecionar a opção **Programmatic access**, nas últimas etapas de criação de usuário você obterá uma **Access Key**.

Na próxima etapa, crie um grupo com as permissões necessárias e adicione o usuário ao grupo:



Guarde a Access Key (ID e Secret) em um lugar seguro:



Deve-se configurar o AWS CLI com a Access Key:

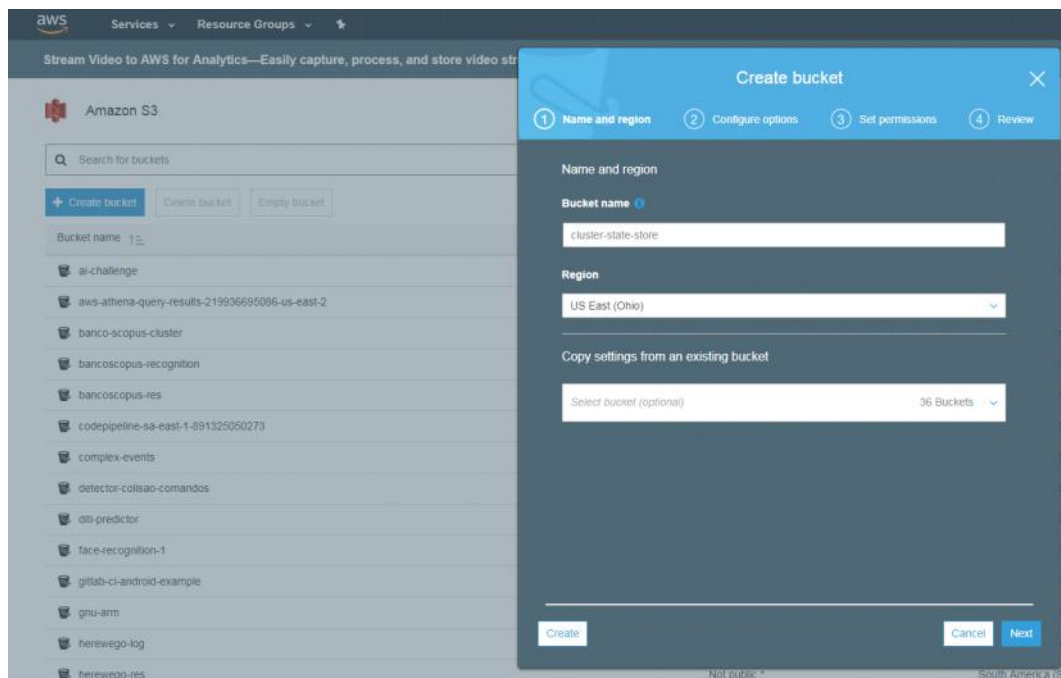
```
aws configure
```

Em seguida a configuração é exportada para que o kops use ela

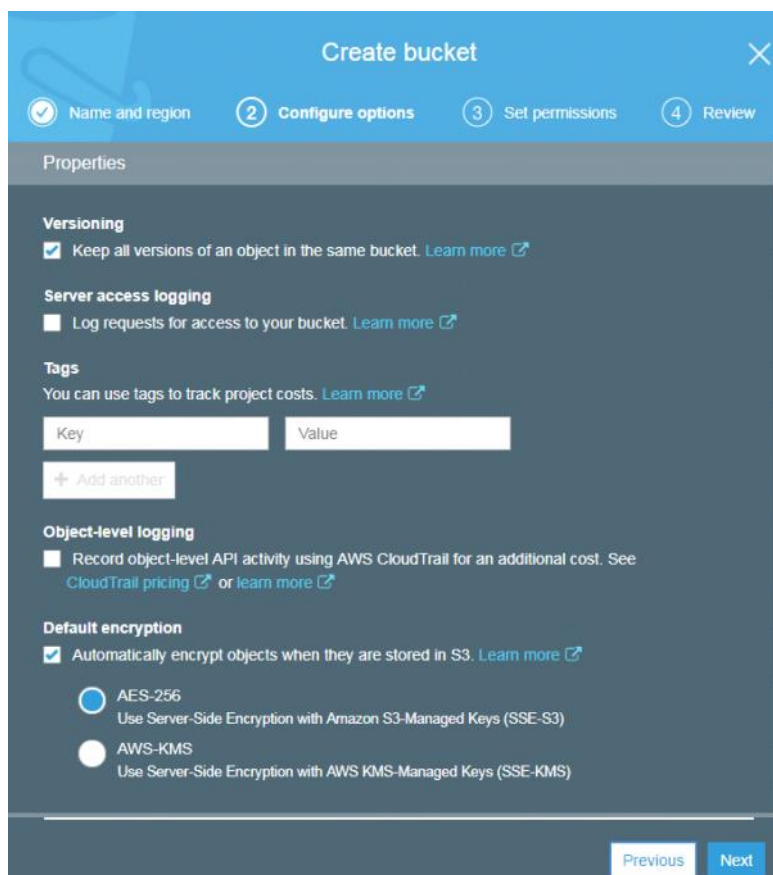
```
export AWS_ACCESS_KEY_ID=$(aws configure get aws_access_key_id)
export AWS_SECRET_ACCESS_KEY=$(aws configure get aws_secret_access_key)
```

1) Configurar um domínio DNS OU na hora de criar o cluster, usar como nome "nomedocluster.k8s.local", o que faz com que os nós do cluster usem comunicação baseada em gossip.

4) Criar bucket S3 para o kops. Ele irá utilizá-lo para guardar o estado e representação do cluster



Ficar atento à configuração de encriptação do bucket:



Não esquecer de dar permissão de acesso ao usuário do kops:

5) Criar cluster:

Criar par de chaves RSA
ssh-keygen

Obs.: Usar valores default durante criação da chave, o que irá salvá-la em "~/.ssh/id_rsa.pub"

Preparar Ambiente:

```
export NAME=nome_do_cluster.k8s.local
export KOPS_STATE_STORE=s3://nome_do_bucket
```

Criar configuração do cluster:

```
kops create cluster --zones us-west-2a ${NAME}
```

Editar configuração do cluster:

```
kops edit cluster ${NAME}
```

Construir cluster:

```
kops update cluster ${NAME} --yes
```


Pipeline I: .gitlab-ci

quinta-feira, 26 de julho de 2018 09:34

Pipeline II: Runner

quinta-feira, 26 de julho de 2018 10:34

Deployment

quinta-feira, 26 de julho de 2018

10:38

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: facerecognition
5  spec:
6    selector:
7      matchLabels:
8        app: facerecognition
9    replicas: 2 # tells deployment to run 2 pods matching the template
10   template: # create pods using pod definition in this template
11     metadata:
12       labels:
13         app: facerecognition
14     spec:
15       containers:
16         - name: facerecognition
17           image: registry.scpsinovacao.net/vval/face-recognition-test:v0.2
18           command: ["/usr/bin/supervisord"]
19           ports:
20             - containerPort: 80
21           env:
22             - name: aws_secret_access_key1
23               valueFrom:
24                 secretKeyRef:
25                   name: facerecognition-secret
26                   key: aws_secret_access_key1
27             - name: aws_access_key_id1
28               valueFrom:
29                 secretKeyRef:
30                   name: facerecognition-secret
31                   key: aws_access_key_id1
32           imagePullSecrets:
33             - name: regcred
```

command

Permite realizar um comando dentro da imagem, após seu deployment. No caso, o campo está executando o script "supervisord", que iniciará o programa

env

Contém as variáveis de ambiente utilizadas pelo programa. O valor das variáveis está declarado em um secret (no caso "facerecognition-secret").

Maiores detalhes encontram-se em [Usando variáveis de ambiente](#)

imagePullSecrets

Caso a imagem do container esteja em um registro privado, é necessário configurar credenciais para acessar esse registro no Kubernetes. O **imagePullSecrets** aponta para o **secret** que contém as credenciais do registro privado.

Para criar esse **secret**, use como tipo **docker-registry** e preferencialmente como nome **regcred**:

```
kubectl create secret docker-registry regcred --docker-server=registry.scpsinovacao.net --
```

docker-username=**ldDoUsuario** --docker-password=**Senha123** --docker-
email=**vval@scopus.com.br**

Maiores detalhes no link:

<https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/>

Service

quinta-feira, 26 de julho de 2018

10:38

Ingress

quinta-feira, 26 de julho de 2018

10:39

Admin.config

quinta-feira, 26 de julho de 2018

10:31

Usando variáveis de ambiente por meio de Secrets no Kubernetes

terça-feira, 24 de julho de 2018 14:33

<https://medium.com/google-cloud/kubernetes-configmaps-and-secrets-68d061f7ab5b>

O programa deve utilizar variáveis de ambiente ao invés de deixar em aberto (variáveis hardcoded) informações confidenciais ou que dependem de configuração.

Exemplo:

Spring Boot:

```
db.username = System.getenv("SQL_DB_USERNAME")
db.password = System.getenv("SQL_DB_PASS")
```

Python:

```
import os
username=os.getenv("SQL_DB_USERNAME")
password=os.getenv("SQL_DB_PASS")
```

Para setar as variáveis de ambiente no kubernetes para um dado serviço:

1) Criar um objeto do tipo ("kind") secret.

O objeto deve ter um nome, declarado em metadata.secret. Os valores das variáveis devem estar em base64.

Dica:

Encoding para base64:

```
echo -n qwerty123 | base64
```

Decoding:

```
echo "cXdlcnR5MTIz" | base64 --decode
```

Exemplo de um objeto **secret**:


```

1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: facerecognition-secret
5  type: Opaque
6  data:
7    aws_secret_access_key1: VGVHcGa25pdV5hd3MuEdvQlpa25pdVlmUXc4ZGlya25pdVGhd3MuHfkmZn25pd5cQ==
8    aws_access_key_id1: QUtJQUa25pdVhd3bi0xlcGVHklbi0x0U0E=
9    teste: 25pdNDU2
10   host_env: 25pdwbi0xW5zdGFuYa25pdVdC5jZDl6bbi0xW5qdHQudXMtZbi0xC0xLnJkcy5hbWF6bi0x3MuY29t
11   port_env: hd3MuGVbi0x==
12   dbname_env: ZGJmGVHNLmQ==
13   user_env: bbi0xpdzcWhd3MuG1pbg==
14   password_env: c25pdaGVHMjM=
15   s3_bucket_env: hd3MudjZS1GVHWNvZ2hd3Mulvbi0x

```

2) Aplicar o arquivo .yaml criado no cluster kubernetes:

```
kubectl apply -f facerecognition-secret.yaml
```

Para verificar todos os secretos aplicados no cluster:

```
kubectl get secrets
```

3) Configurar as variáveis de ambiente no deployment.yaml do App que utilizará-las

Exemplo: Para a variável de ambiente usada pelo programa "aws_secret_access_key1", o kubernetes atribuirá o valor declarado no objeto chamado "facerecognition-secret" para a key "aws_secret_access_key1".

Ps.: Não é necessário, mas usei o mesmo nome para a key do secret e variável do deployment.

```

1  apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
2  kind: Deployment
3  metadata:
4    name: facerecognition
5  spec:
6    selector:
7      matchLabels:
8        app: facerecognition
9    replicas: 2 # tells deployment to run 2 pods matching the template
10   template: # create pods using pod definition in this template
11     metadata:
12       # unlike pod-nginx.yaml, the name is not included in the meta data
13       # generated from the deployment name
14       labels:
15         app: facerecognition
16     spec:
17       containers:
18         - name: facerecognition
19           image: registry.scpsoinovacao.net/vval/face-recognition-test:v0.2
20           command: ["/usr/bin/supervisord"]
21           ports:
22             - containerPort: 80
23           env:
24             - name: aws_secret_access_key1
25               valueFrom:
26                 secretKeyRef:
27                   name: facerecognition-secret
28                   key: aws_secret_access_key1
29             - name: aws_access_key_id1
30               valueFrom:
31                 secretKeyRef:
32                   name: facerecognition-secret
33                   key: aws_access_key_id1
34             - name: teste
35               valueFrom:
36                 secretKeyRef:
37                   name: facerecognition-secret
38                   key: teste
39             - name: host_env
40               valueFrom:
41                 secretKeyRef:
42                   name: facerecognition-secret

```

4) Caso o app use variáveis de ambiente na fase de testes, é necessário setar as variáveis no ambiente apropriado,

já que a configuração anterior só é válida para os apps que estejam em produção no kubernetes.

O ideal é usar uma ferramenta de CI/CD, como GitLab-CI, Jenkins, Bamboo, etc...

No caso do GitLab-CI, é possível setar as variáveis em **Settings->CI/CD->Variables**

Variables ?

Variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use variables for passwords, secret keys, or whatever you want.

AWS_ACCESS_KEY_ID	*****	Protected	<input checked="" type="checkbox"/>	-
aws_access_key_id1	*****	Protected	<input checked="" type="checkbox"/>	-
AWS_SECRET_ACCESS_KEY	*****	Protected	<input checked="" type="checkbox"/>	-
aws_secret_access_key1	*****	Protected	<input checked="" type="checkbox"/>	-
CI_USER_NAME	*****	Protected	<input checked="" type="checkbox"/>	-
CI_USER_PASS	*****	Protected	<input checked="" type="checkbox"/>	-
dbname_env	*****	Protected	<input checked="" type="checkbox"/>	-
DOCKER_AUTH_CONFIG	*****	Protected	<input checked="" type="checkbox"/>	-
host_env	*****	Protected	<input checked="" type="checkbox"/>	-
password_env	*****	Protected	<input checked="" type="checkbox"/>	-
port_env	*****	Protected	<input checked="" type="checkbox"/>	-
s3_bucket_env	*****	Protected	<input checked="" type="checkbox"/>	-
teste	*****	Protected	<input checked="" type="checkbox"/>	-
user_env	*****	Protected	<input checked="" type="checkbox"/>	-
Input variable key	Input variable value	Protected	<input checked="" type="checkbox"/>	

Save variables

Reveal values

Versionamento

quarta-feira, 25 de julho de 2018 14:50

Existem inúmeras formas de controlar a versão de um app. No serviço facerecognition isso foi feito da seguinte forma:

1) Utilização de um arquivo VERSION.

Como conteúdo do arquivo, um texto no formato "v0.1.2".
O arquivo é adicionado na pasta raíz do repositório.

2) Configuração do pipeline

A ferramenta do pipeline usa versão nos momentos de criação(build), controle(push no registro) e execução(deploy) da imagem docker

.gitlab-ci.yml (simplificado)

Docker-build:

artifacts:

paths:

- **shared_variables**

script:

- **export version='cat VERSION'**

- **docker build -t registry.scpsinovacao.net/banco-scopus/face-recognition-test:\$version .**

Deploy:

script:

- source **shared_variables**

- **kubectl --kubeconfig=admin.config set image deployment/facerecognition facerecognition=registry.scpsinovacao.net/banco-scopus/face-recognition-test:\$version**

O arquivo contendo a versão não é acessível na fase deploy. Para tornar a versão acessível, ela é exportada no arquivo **shared_variables**, compartilhado com os outros estagios via **artifacts.paths**

Registro de Containers

Estão localizados em "Registry"

F

face-recognition-test

Project

Repository

Issues0

Merge Requests0

CI / CD

Operations

Registry

Wiki

Snippets

Settings

Container Registry

With the Docker Container Registry integrated into GitLab, every project can have its own space to store its Docker images.

Learn more about Container Registry.

^ banco-scopus/face-recognition-test

Tag	Tag ID	Size	Created
v0.5	995cd9f45	1.03 GiB	22 hours ago
v0.6	e05cd8f80	1.03 GiB	22 hours ago
v0.3	55884cb84	1.03 GiB	1 day ago
v0.4	a4961a527	1.03 GiB	22 hours ago

Monitorização

sexta-feira, 27 de julho de 2018

08:54

Apresentacao

sexta-feira, 10 de agosto de 2018 16:55

DevOps

Artigo: The three ways devops explicado:

<https://www.ibm.com/developerworks/br/library/se-devops/part1/index.html>

<https://www.ibm.com/developerworks/library/se-devops/part2/part2-pdf.pdf>

São três princípios básicos da cultura DevOps

"DevOps applies agile and lean principles across the entire software supply chain. It enables a business to maximize the speed of its delivery of a product or service, from initial idea to production release to customer feedback to enhancements based on that feedback."

"DevOps improves the way that a business delivers value to its customers, suppliers, and partners"

(trecho do livro [DevOps for Dummies](#))

Relação entre DevOps e Agile

De certa forma o DevOps é para a operações o que o Agile é para o desenvolvimento.

Um dos princípios mais importantes do processo de desenvolvimento Ágil é oferecer software funcional em pequenas quantidades. Esse processo em geral gerou resultados muito positivos para as empresas que o implementaram.

Entretanto ele se esqueceu da parte de operações, criando um "gargalo" na cadeia logística do desenvolvimento até o consumidor final. Quando o código não é enviado para produção conforme é desenvolvido, as implementações se acumulam no estágio de operações, de forma que mesmo que haja geração frequente de código, isso leva tempo a chegar no usuário.

O DevOps tenta solucionar esse "gap" ao permitir um fluxo de trabalho muito mais contínuo para Operações de TI. DevOps estende e completa o processo contínuo de integração e release garantindo que o código está pronto para produção e que irá proporcionar valor para o cliente.

Problemas que enfrentamos atualmente

- Demora na entrega
- Bugs frequentes e que nos pegam de surpresa
- ???

Vantagens de DevOps

- Automatização
- Menor tempo de entrega
- Menor abertura a erros humanos
- Detecção de bugs
- "Rollback" de versão
- Testes A/B

Retorno sobre o Investimento:

- Melhor experiência: Capacidade de responder rapidamente a feedbacks e demandas de mercado
- Aumento de receita: Rapidez na entrega de valor
- Redução de custos de infraestrutura e mão-de-obra
- Redução de riscos
- Confiabilidade e estabilidade
- Maior capacidade de inovação

DevOps facilita a computação em nuvem

-Vantagens da computação em nuvem (não sei se cabe isso)

- Disponibilidade
- Escalabilidade
- Custos

Boas práticas de programação que favorecem DevOps (talvez tenha mais a ver com microserviços ou cloud)

- Gerenciador de dependências
- Variáveis de ambiente
- Testes automatizados

Como medir o impacto do DevOps?

KPIs

<https://puppet.com/blog/5-kpis-make-case-for-devops>

1. Deployment frequency
2. Speed of deployment
3. Deployment success rate
4. How quickly service can be restored after a failed deployment
5. Culture: Difícil de mensurar. Possivelmente por meio de pesquisas com os funcionários.

Casos de sucesso e Como implementar a cultura DevOps numa organização

-Alguns links a respeito:

<https://techbeacon.com/10-companies-killing-it-devops>

<https://devopsagenda.techtarget.com/opinion/Four-key-lessons-to-apply-from-DevOps-success-stories>

<https://searchitoperations.techtarget.com/news/450297784/QA-DevOps-transformation-is-not-just-for-devs-and-unicorns>

<https://www1.pega.com/system/files/resources/2017-11/DevOps%20Infographic%20FINAL.pdf>

<http://servicevirtualization.com/how-amazon-made-the-leap-to-a-devops-culture/>

-Falar do Inovabank??? (Como a primeira experiência da Scopus com DevOps)

-Falar o que estamos fazendo atualmente na Arquitetura??

O que precisa adicionar no Agile?

You must modify the Agile sprint policy so that instead of having just shippable, viable code at the end of each sprint, you also require a rebuildable environment that the code deploys into. Ideally, this should happen at the earliest sprints (sprint 0 and sprint 1).

<https://www.ibm.com/developerworks/library/se-devops/part2/part2-pdf.pdf>

Links

quarta-feira, 15 de agosto de 2018 16:06

cases

<https://techbeacon.com/10-companies-killing-it-devops>

<https://devopsagenda.techtarget.com/opinion/Four-key-lessons-to-apply-from-DevOps-success-stories>

<https://searchitoperations.techtarget.com/news/450297784/QA-DevOps-transformation-is-not-just-for-devs-and-unicorns>

<https://www1.pega.com/system/files/resources/2017-11/DevOps%20Infographic%20FINAL.pdf>

<http://servicevirtualization.com/how-amazon-made-the-leap-to-a-devops-culture/>

Ferramentas

quinta-feira, 16 de agosto de 2018

17:19

ElasticSearch

Logstash

Kibana

Grafana

Graphite

Filebeat

Prometheus

<https://indexoutofrange.com/Choosing-centralized-logging-and-monitoring-system/>

Comparação API Gateways

quarta-feira, 5 de setembro de 2018 18:30

<https://blog.getambassador.io/selecting-an-api-gateway-for-continuous-delivery-of-cloud-native-applications-8ba05fa1c74>

Ver Countour

Ambassador

- Kubernetes-Native: O estado do Ambassador fica guardado no próprio kubernetes, ao invés de precisar de um banco de dados externo.
Pode ser configurado usando a mesma sintaxe que outros recursos do Kubernetes como "Deployment" e "Service", adicionando recursos como "Mapping", "AuthService", "TracingService".
- Self-Service: Assim como um time de desenvolvedores de um microserviço é capaz de mantê-lo no kubernetes, é possível que esse mesmo time configure regras de mapeamento do Ambassador sem se preocupar com a manutenção de um serviço ou arquivo de configuração centralizado.

Problema com Ambassador:

A parte de autenticação é fraca. Ele delega tudo para um único serviço externo. Isso obriga múltiplos times a contribuírem com o mesmo serviço.

Isso vai contra a filosofia de "self-service" e microserviços.

"This is what I was expecting Ambassador to have after experiencing the "self service" style configuration for routing. How auth currently works goes totally against the self service ideology IMO. Right now, all teams/individuals hav to ship changes to two independent services (their service + auth service) whenever they need to change how their service is protected. Also, multiple teams will end up contributing to the same auth service and the auth service might end up being much more complex than it would make sense as business logic might creep in to the auth service handlers."

De <<https://github.com/datawire/ambassador/issues/216>>

API Gateway Vs Reverse Proxy

segunda-feira, 3 de setembro de 2018 10:47

It is easier to think about them if you realize they aren't mutually exclusive. Think of an API gateway as a specific type reverse proxy implementation.

In regards to your questions, it is not uncommon to see the both used in conjunction where the API gateway is treated as an application tier that sits behind a reverse proxy for load balancing and health checking. An example would be something like a WAF sandwich architecture in that your Web Application Firewall/API Gateway is sandwiched by reverse proxy tiers, one for the WAF itself and the other for the individual microservices it talks to. Regarding the differences, they are very similar. It's just nomenclature. As you take a basic reverse proxy setup and start bolting on more pieces like authentication, rate limiting, dynamic config updates, and service discovery, people are more likely to call that an API gateway.

De <<https://stackoverflow.com/questions/35756663/api-gateway-vs-reverse-proxy>>

I believe, API Gateway is a reverse proxy that can be configured dynamically via API and potentially via UI, while traditional reverse proxy (like Nginx, HAProxy or Apache) is configured via config file and has to be restarted when configuration changes. Thus, API Gateway should be used when routing rules or other configuration often changes. To your questions:

1. It makes sense as long as every component in this sequence serves its purpose.
2. Differences are not in feature list but in the way configuration changes applied.

De <<https://stackoverflow.com/questions/35756663/api-gateway-vs-reverse-proxy>>

Nginx vs Envoy vs Haproxy

segunda-feira, 3 de setembro de 2018 10:09

<https://blog.getambassador.io/envoy-vs-nginx-vs-haproxy-why-the-open-source-ambassador-api-gateway-chose-envoy-23826aed79ef>

Envoy was designed from the ground up for microservices, with features such as hitless reloads (called [hot restart](#)), observability, resilience, and advanced load balancing. Envoy also embraced distributed architectures, adopting [eventual consistency](#) as a core design principle and [exposing dynamic APIs for configuration](#). Traditionally, proxies have been configured using static configuration files. Envoy, while supporting a static configuration model, also allows configuration via gRPC/protobuf APIs. This simplifies management at scale, and also allows Envoy to work better in environments with ephemeral services.

Ambassador Vs Ingress

segunda-feira, 3 de setembro de 2018 19:48

authentication, logging, rate limiting, and load balancing

<https://www.getambassador.io/reference/mappings>

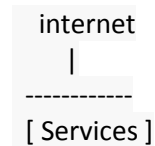
Oferece regras de mapeamento mais complexas, como mudar a URL

Prioridades de

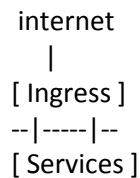
Ingress

quinta-feira, 23 de agosto de 2018

18:05



An Ingress is a collection of rules that allow inbound connections to reach the cluster services.



It can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting, and more. Users request ingress by POSTing the Ingress resource to the API server. An [Ingress controller](#) is responsible for fulfilling the Ingress, usually with a loadbalancer, though it may also configure your edge router or additional frontends to help handle the traffic in an HA manner

Ingress Controllers and Ingress Resources

<https://blog.getambassador.io/kubernetes-ingress-nodeport-load-balancers-and-ingress-controllers-6e29f1c44f2d>

Kubernetes supports a high level abstraction called [Ingress](#), which allows simple host or URL based HTTP routing. An ingress is a core concept (in beta) of Kubernetes, but is always implemented by a third party proxy. These implementations are known as ingress controllers. An ingress controller is responsible for reading the Ingress Resource information and processing that data accordingly. Different ingress controllers have extended the specification in different ways to support additional use cases.

Ingress is tightly integrated into Kubernetes, meaning that your existing workflows around kubectl will likely extend nicely to managing ingress. Note that an ingress controller typically doesn't eliminate the need for an external load balancer — the ingress controller simply adds an additional layer of routing and control behind the load balancer.

Sidecar

terça-feira, 11 de setembro de 2018 18:53

What are some things that a microservice sidecar could do for a microservice?

De <<http://ispyker.blogspot.com/2014/08/a-netflixoss-sidecar-in-support-of-non.html>>

Prana is conceptually “attached” to the main (aka Parent) application and complements it by providing “platform features” that are otherwise available as libraries within a JVM-based application.

<https://medium.com/netflix-techblog/prana-a-sidecar-for-your-netflix-paas-based-applications-and-services-258a5790a015>

A sidecar is a *utility* container in the Pod and its purpose is to support the main container. It is important to note that standalone sidecar does not serve any purpose, it must be paired with one or more main containers. Generally, sidecar container is reusable and can be paired with numerous type of main containers.

De <<https://abhishek-tiwari.com/a-sidecar-for-your-service-mesh/>>

Service Mesh

sexta-feira, 24 de agosto de 2018 10:52

<https://www.nginx.com/blog/what-is-a-service-mesh/>

<https://blog.buoyant.io/2017/04/25/whats-a-service-mesh-and-why-do-i-need-one/>

<https://thenewstack.io/which-service-mesh-should-i-use/>

A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the service mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware. (But there are variations to this idea, as we'll see.)

The concept of the service mesh as a separate layer is tied to the rise of the cloud native application. In the cloud native model, a single application might consist of hundreds of services; each service might have thousands of instances; and each of those instances might be in a constantly-changing state as they are dynamically scheduled by an orchestrator like Kubernetes. Not only is service communication in this world incredibly complex, it's a pervasive and fundamental part of runtime behavior. Managing it is vital to ensuring end-to-end performance and reliability.

The explicit goal of the service mesh is to move service communication out of the realm of the invisible, implied infrastructure, and into the role of a *first-class member of the ecosystem*—where it can be monitored, managed and controlled.

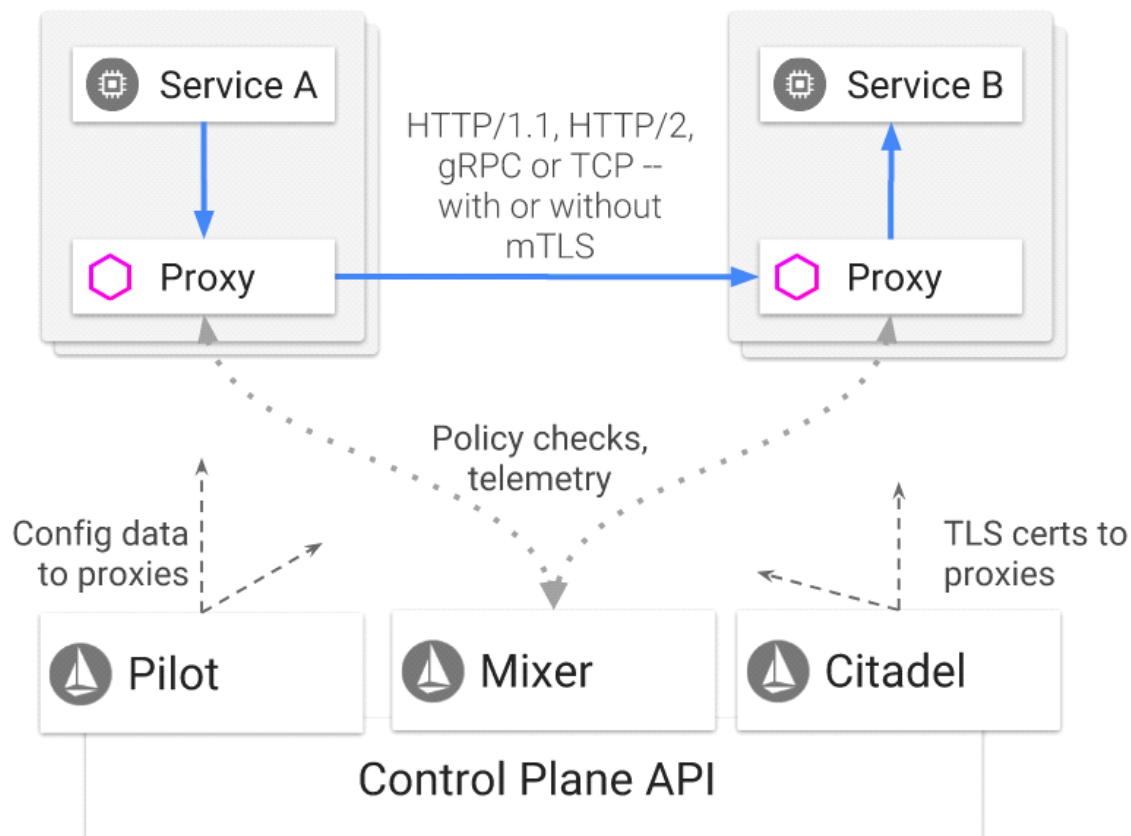
Istio

sexta-feira, 24 de agosto de 2018 13:26

<https://developers.redhat.com/blog/2018/03/13/istio-route-rules-service-requests/>

<https://medium.com/google-cloud/istio-why-do-i-need-it-18d122838ee3>

<https://istio.io/docs/concepts/what-is-istio/>



Implementing a Service Mesh with Istio to Simplify Microservices Communication

<https://kublr.com/blog/implementing-a-service-mesh-with-istio-to-simplify-microservices-communication/>

Ambassador and Istio: Edge Proxy and Service Mesh

De <https://www.getambassador.io/user-guide/with-istio/>

Istio Security

quarta-feira, 12 de setembro de 2018 11:17

<https://istio.io/docs/concepts/security/>

Ambassador

sexta-feira, 24 de agosto de 2018 15:28

Ambassador is an open source, Kubernetes-native [microservices API gateway](#) built on the [Envoy Proxy](#)

Ambassador provides all the functionality of a traditional ingress controller (i.e., path-based routing) while exposing many additional capabilities such as [authentication](#), URL rewriting, CORS, rate limiting, and automatic metrics collection (the [mappings reference](#) contains a full list of supported options)

Ambassador é um API Gateway open-source construído usando o Envoy Proxy. Os diferenciais mais significativos do Ambassador são a sua integração com o Kubernetes e gestão de configuração voltada para microsserviços.

- Kubernetes-Native: O estado do Ambassador fica guardado no próprio kubernetes, ao invés de precisar de um banco de dados externo. Pode ser configurado usando a mesma sintaxe que outros recursos do Kubernetes como "Deployment" e "Service", adicionando recursos como "Mapping", "AuthService", "TracingService".
- Self-Service: Assim como um time de desenvolvedores de um microsserviço é capaz de mantê-lo no kubernetes, é possível que esse mesmo time configure regras de mapeamento do Ambassador sem se preocupar com a manutenção de um serviço ou arquivo de configuração centralizado.

Ambassador + Istio

<https://www.getambassador.io/user-guide/with-istio>

Service Mesh vs API Gateway

Ambassador is a Kubernetes-native API gateway for microservices. Ambassador is deployed at the edge of your network, and routes incoming traffic to your internal services (aka "north-south" traffic). [Istio](#) is a service mesh for microservices, and is designed to add application-level Layer (L7) observability, routing, and resilience to service-to-service traffic (aka "east-west" traffic). Both Istio and Ambassador are built using [Envoy](#).

Ambassador and Istio can be deployed together on Kubernetes. In this configuration, incoming traffic from outside the cluster is first routed through Ambassador, which then routes the traffic to Istio-powered services. Ambassador handles authentication, edge routing, TLS termination, and other traditional edge functions.

This allows the operator to have the best of both worlds: a high performance, modern edge service (Ambassador) combined with a state-of-the-art service mesh (Istio). Istio's basic [ingress controller](#) is very limited, and has no support for authentication or many of the other features of Ambassador.

Instalar Ambassador

segunda-feira, 3 de setembro de 2018 16:18

Baixar o seguinte arquivo, que contém o deploy "ambassador", serviço "ambassador-master" e recursos de controle de acesso "as configurações do kubernetes

wget <https://www.getambassador.io/yaml/ambassador/ambassador-rbac.yaml>

(opcional) Modificar o arquivo. É possível mudar o namespace usado, número de réplicas do deploy, etc...

Aplicá-lo no kubernetes:

```
kubectl apply -f ambassador-rbac.yaml
```

Criar arquivo ambassador-svc.yaml

```
nano ambassador-svc.yaml
```

Com o seguinte conteúdo:

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    service: ambassador
  name: ambassador
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    service: ambassador
```

Aplicá-lo no cluster:

```
kubectl apply -f ambassador-svc.yaml
```

Mapear Serviço

segunda-feira, 3 de setembro de 2018 19:45

O ambassador consegue mapear serviços por meio de uma annotation presente em suas configurações.

Exemplo:

```
---
apiVersion: v1
kind: Service
metadata:
  name: facerecognition
  labels:
    app: facerecognition
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      kind: Mapping
      name: facerecognition-mapping
      prefix: /face-recognition
      service: facerecognition
spec:
  selector:
    app: facerecognition
  ports:
    - port: 80
      targetPort: 80
```

Em **vermelho**, parte referente ao Ambassador adicionada no arquivo de descrição de um serviço.

Obs.: É possível adicionar as annotations de mapeamento referentes a um serviço no serviço principal do Ambassador.

O "prefix" é o caminho desejado do serviço no cluster. Exemplo: <http://inovabank.com/face-recognition>

É possível configurar profundamente o mapeamento. O ambassador suporta regras de regex, usando nome do host, método REST, funcionalidades de rewrite (troca o prefixo por outro). Há ainda funcionalidades de timeout, limite de requisições, redirecionamento usando HTTP 301, redirecionamento em caso de erro 404, definição de prioridade de requisições.

O link a seguir conta com uma documentação de como detalhar o Mapping além do prefixo: <https://www.getambassador.io/reference/mappings>

Mapear url externo

segunda-feira, 3 de setembro de 2018 18:15

O ambassador também pode mapear URLs fora do cluster a um dado prefixo. Nesses caso é ideal usar "host-rewrite", já que é comum que um serviço use informações de seu host. Um exemplo em que o host é usado é em Virtual Hosting, onde múltiplos domínios compartilham o mesmo Ip.

```
---
apiVersion: v1
kind: Service
metadata:
  name: google
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      kind: Mapping
      name: google_mapping
      prefix: /google
      service: https://google.com:443
      host_rewrite: google.com
spec:
  type: ClusterIP
  clusterIP: None
```

Nesse caso uma requisição feita para <http://urlCluster/google/search?q=TestePesquisa> seria redirecionada a <https://www.google.com/search?q=TestePesquisa>

Autenticação no Ambassador

terça-feira, 4 de setembro de 2018 11:41

O Ambassador delega a autenticação a um serviço externo. Ao declarar um tipo AuthService (ao invés de "Mapping") o Ambassador irá rotear todo o tráfego para esse serviço. No momento (setembro/2018) esse serviço de autenticação externo é responsável por todas as requisições ao Ambassador, entretanto existe no Roadmap do projeto do Ambassador a funcionalidade de declarar um ou mais autenticadores com diferentes escopos. Atualmente, se você quiser deixar um serviço "em aberto" é preciso tratar isso dentro do autenticador.

No exemplo a seguir o serviço de autenticação com o nome "ambassador-auth" foi declarado no service.yaml do Ambassador.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    service: ambassador
  name: ambassador
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      kind: AuthService
      name: authentication
      auth_service: "ambassador-auth:8080"
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    service: ambassador
```

É importante notar que o autenticador não passa de um serviço como outro qualquer dentro do cluster, como pode ser observado por meio das descrições de Deployment e Service do Ambassador-Auth:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ambassador-auth
spec:
  selector:
    matchLabels:
      app: ambassador-auth
  replicas: 1
  template:
    metadata:
      labels:
```

```
  app: ambassador-auth
spec:
  containers:
  - name: ambassador-auth
    image: registry.scpsinovacao.net/vval/ambassador-auth:v0.0.1
    ports:
    - containerPort: 8080
  imagePullSecrets:
  - name: regcred

---
apiVersion: v1
kind: Service
metadata:
  name: ambassador-auth
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
  selector:
    app: ambassador-auth
```

O processo de autenticação ocorre da seguinte maneira:

- 1) Ambassador recebe requisição do cliente e redireciona a requisição, idêntica à original porém sem o corpo (body) ao autenticador.
- 2.1) Autenticador recebe requisição e retorna HTTP status 200 (OK)
- 3.1) Ambassador recebe OK e envia requisição original ao serviço de destino, de acordo com o Mapping.
- 2.2) Autenticador retorna qualquer coisa que não seja um status OK
- 3.2) Ambassador não envia requisição ao destino, mas sim retorna ao cliente a mensagem enviada pelo autenticador

Monitorar Ambassador com Prometheus Operator

terça-feira, 4 de setembro de 2018 10:11

Debugging

quinta-feira, 18 de outubro de 2018 18:38

```
kubectl get pods  
kubectl exec -it ambassador-7f99d49b4b-c7589 /bin/sh  
procurar pela pasta ambassador-config-X
```

L4 vs L7 Load Balancer

terça-feira, 22 de janeiro de 2019 15:33

L4 vs L7 Showdown

De <<https://www.awsadvent.com/2016/12/10/l4-vs-l7-showdown/>>

Load Balancing Layer 4 and Layer 7

De <<https://freeloadbalancer.com/load-balancing-layer-4-and-layer-7/>>

Infrastructure-as-Code (IaC)

segunda-feira, 14 de janeiro de 2019 15:02

Tradicionalmente os administradores de sistemas faziam o deploy de infraestrutura, como servidores, bancos de dados, configurações, manualmente.

The idea behind IAC is to write code to define, provision, and manage your infrastructure. This has a number of benefits:

You can automate your entire provisioning and deployment process, which makes it much faster and more reliable than any manual process.

You can represent the state of your infrastructure in source files that anyone can read rather than a sysadmin's head.

You can store those source files in version control, which means the entire history of your infrastructure is now captured in the commit log, which you can use to debug problems, and if necessary, roll back to older versions.

You can validate each infrastructure change through code reviews and automated tests.

You can create reusable, documented, battle-tested infrastructure packages that make it easier to scale and evolve your infrastructure.

Terraform

segunda-feira, 14 de janeiro de 2019 19:03

<https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c>

Comparação entre o Terraform e outras soluções

Configuration Management vs Orchestration

Chef, Puppet, Ansible, and SaltStack are all “configuration management” tools, which means they are designed to install and manage software on existing servers. CloudFormation and Terraform are “orchestration tools”, which means they are designed to provision the servers themselves, leaving the job of configuring those servers to other tools

Mutável vs Imutável

Procedural (o que fazer) vs. Declarativo (estado)

O método procedural obriga o usuário a saber o estado atual da infraestrutura para poder tomar as ações corretas ao modificá-la. Já no método declarativo o usuário define a infraestrutura desejada e a ferramenta é a responsável por atingir esse estado. Outra vantagem do método declarativo é que, ao contrário do procedural, o código é reutilizável.

Arquitetura Client/Server vs. Client-only

Chef, Puppet, and SaltStack são Client/Server e necessitam lançar um servidor só para gestão de configuração, além de instalar software extra em cada servidor. CloudFormation, Ansible, and Terraform usam uma arquitetura com apenas cliente. O Ansible conecta diretamente aos servidores via SSH. Já o Terraform usa APIs do provedor cloud (como AWS, GCK, etc..) para provisionar infraestrutura.

```
provider "aws" {  
  region = "us-west-2"  
}
```

```
resource "aws_instance" "example" {  
  ami = "ami-0f9cf087c1f27d9b1"  
  instance_type = "t2.micro"  
}
```

terraform plan

terraform apply

Para dar um nome à instância criada, adicionar tag com nome da instância:

```
resource "aws_instance" "example" {  
  ami = "ami-076e276d85f524150"  
  instance_type = "t2.micro"
```



```
tags {  
  Name = "terraform-example"  
}  
}
```

Projeto NLU

quarta-feira, 30 de janeiro de 2019 12:19

Entrar maquina:
ssh root@IP-maquina

Cluster kubernetes:
<https://source.scpsinovacao.net/Isoares/NLU-K8S-CentOS>

MongoDB:
Controlar usuarios:
<https://medium.com/mongoaudit/how-to-enable-authentication-on-mongodb-b9e8a924efac>

A VER:
<https://blog.getspool.com/2012/04/18/fake-s3-save-time-money-and-develop-offline/>

Minio

sexta-feira, 15 de fevereiro de 2019 15:45

User minio as docker registry:

<https://medium.com/@enne/use-minio-as-docker-registry-storage-driver-c9c72c72cc87>