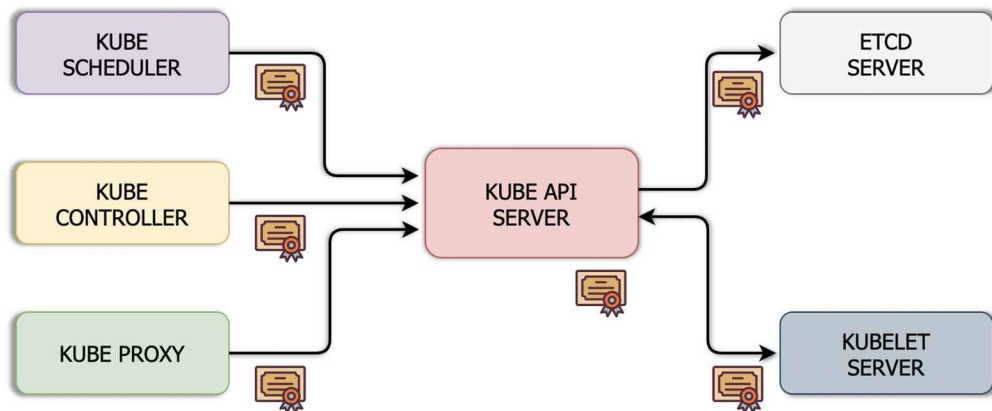


Kubernetes

Basic Security

Paulo Vigne

TLS in Kubernetes



```
[Service]
ExecStart=/usr/local/bin/kube-controller-manager \
--address=0.0.0.0 \
--service-cluster-ip-range=10.0.0.0/24 \
--cluster-cidr=10.244.0.0/16 \
--allocate-node-cidrs=true \
--kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \
--authentication-kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \
--authorization-kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \
--leader-elect=true \
--cluster-signing-cert-file=/var/lib/kubernetes/ca.crt \
--cluster-signing-key-file=/var/lib/kubernetes/ca.key \
--root-ca-file=/var/lib/kubernetes/ca.crt \
--service-account-private-key-file=/var/lib/kubernetes/service-account.key \
--use-service-account-credentials=true \
--v=2
```

```
[Service]
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${SERVER_IP} \
--allow-privileged=true \
--authorization-mode=Node,RBAC \
--client-ca-file=/var/lib/kubernetes/ca.crt \
--enable-admission-plugins=NamespaceLifecycle,NodeRestriction,LimitRange \
--enable-bootstrap-token-auth=true \
--etcd-cafile=/var/lib/kubernetes/ca.crt \
--etcd-certfile=/var/lib/kubernetes/etcd.crt \
--etcd-keyfile=/var/lib/kubernetes/etcd.key \
--etcd-servers=https://127.0.0.1:2379 \
--kubelet-certificate-authority=/var/lib/kubernetes/ca.crt \
--kubelet-client-certificate=/var/lib/kubernetes/kube-api.crt \
--kubelet-client-key=/var/lib/kubernetes/kube-api.key \
--kubelet-https=true \
--service-account-key-file=/var/lib/kubernetes/service-account.crt \
--service-cluster-ip-range=10.0.0.0/24 \
--tls-cert-file=/var/lib/kubernetes/kube-api.crt \
--tls-private-key-file=/var/lib/kubernetes/kube-api.key \
--requestheader-client-ca-file=/var/lib/kubernetes/ca.crt \
```

```
kubectl config set-cluster kubernetes-from-scratch \
--certificate-authority=ca.crt \
--embed-certs=true \
--server=https://127.0.0.1:6443 \
--kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-cluster kubernetes-from-scratch \
--certificate-authority=ca.crt \
--embed-certs=true \
--server=https://127.0.0.1:6443 \
--kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-credentials system:kube-controller-manager \
--client-certificate=kube-controller-manager.crt \
--client-key=kube-controller-manager.key \
--embed-certs=true \
--kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-context default \
--cluster=kubernetes-from-scratch \
--user=system:kube-controller-manager \
--kubeconfig=kube-controller-manager.kubeconfig
```

Autenticação no Kubernetes

Quem pode Acessar ???

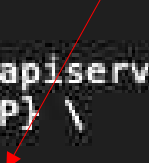
Mechanism	Secret Source	Usage
X509 Client Certs	CSR generated externally and signed with the cluster CA key	Enterprise CA / PKI
	Via Kubernetes API CertificateSigningRequest	Kubernetes cluster admin
Bearer token	Bootstrap token	Internal use
	Node authentication token	Internal use
	Static token file	Insecure
	ServiceAccount token	Pods, containers, applications, users
	OIDC token	Users
HTTP Basic auth	Static password file	Insecure
Auth proxy	N/A (trust proxy)	Integration
Impersonate	N/A (trust account)	Integration and administration

Autorização no Kubernetes

O que eles podem fazer ???

Mechanism	Decision source	Usage
Node	API Server built-in	Internal use (kubelets)
ABAC	Static file	Insecure, deprecated
RBAC	API Objects	Users and administrators
WebHook	External services	Integration
AlwaysDeny AlwaysAllow	API Server built-in	Testing

```
[Service]
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${SERVER_IP} \
--allow-privileged=true \
--authorization-mode=Node,RBAC \
--client-ca-file=/var/lib/kubernetes/ca.crt
--enable-admission-plugins=NamespaceLifecycle
--enable-bootstrap-token-auth=true \
```



Tipos de Accounts no Kubernetes

Temos Users e ServiceAccounts

- **Users:** Acesso por humanos, temos a Account admin, geralmente criada no deploy do cluster (rke, kubeadm) e os demais usuários.

- **ServiceAccounts:** Acesso programático por serviços internos ou externos ao cluster.

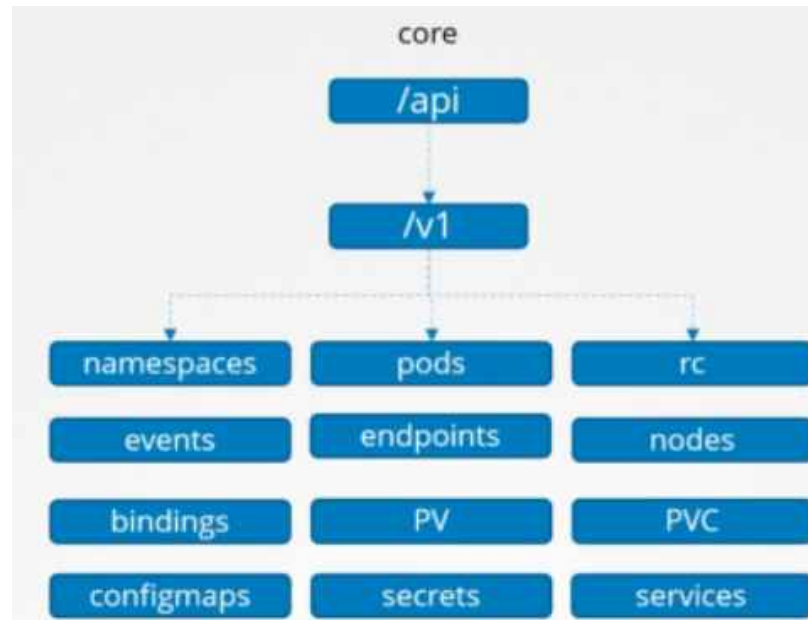
Kubernetes API

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/>

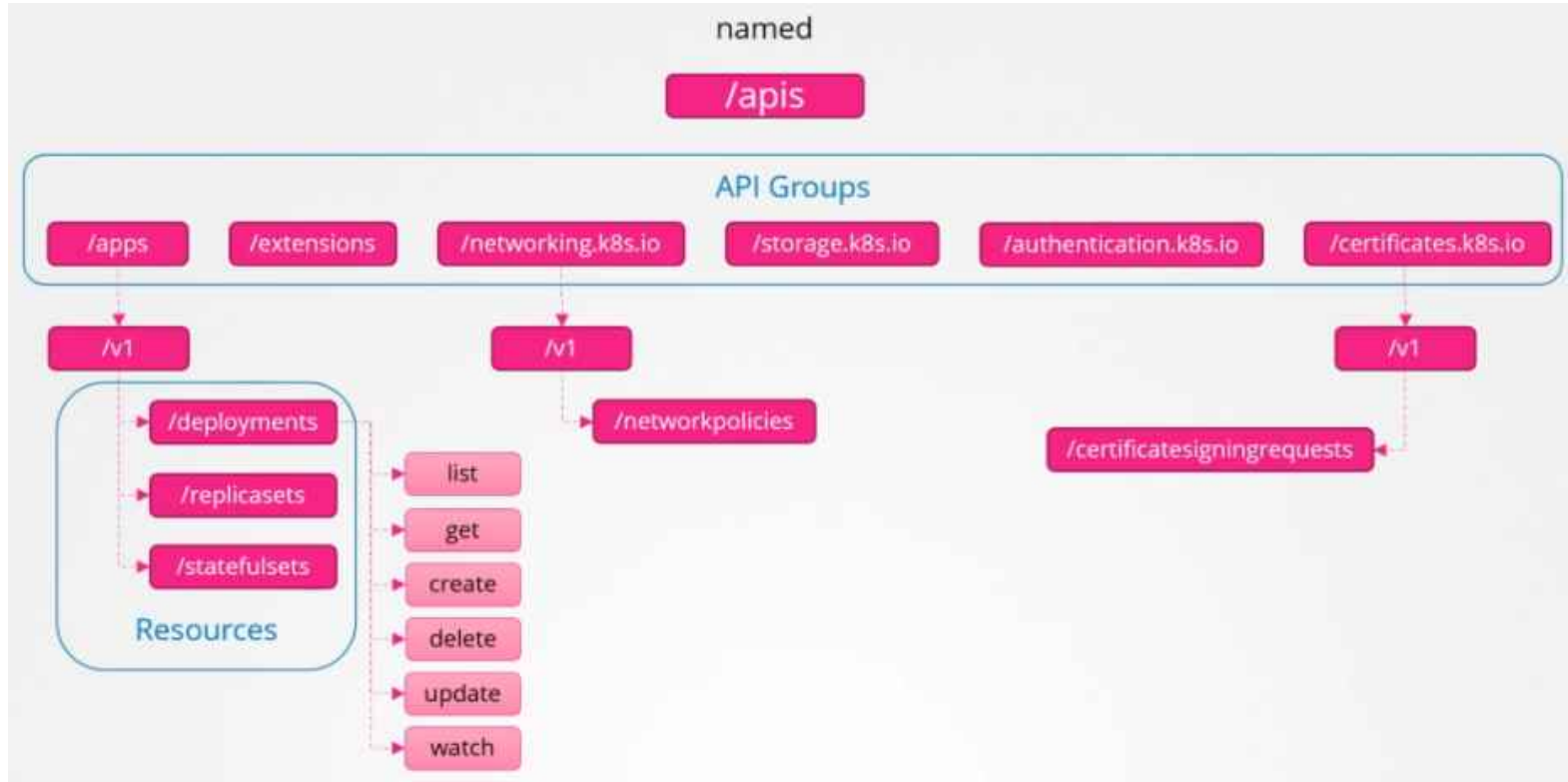
Responsáveis pela funcionalidade do Cluster



Kubernetes API



Kubernetes API



Kubernetes RBAC (Role Base Access Control)

É uma maneira de definir o que usuários ou serviceaccounts podem fazer no cluster. São definidos papéis (roles) e permissões, podem ser aplicados de maneira declarativa ou por diversas extensões.

O Kubernetes provê um API endpoint do qual os usuários podem gerenciar contêineres através de um ambiente distribuído. A API segue o padrão REST e o Kubernetes manipula tudo como um recurso.

Kubernetes RBAC (Role Base Access Control)

Three groups

Subjects

User



Group



Service account



Operations

list

get

create

update

delete

watch

patch

get

post

put

delete

Resources



Pods

Nodes

ConfigMaps

Secrets

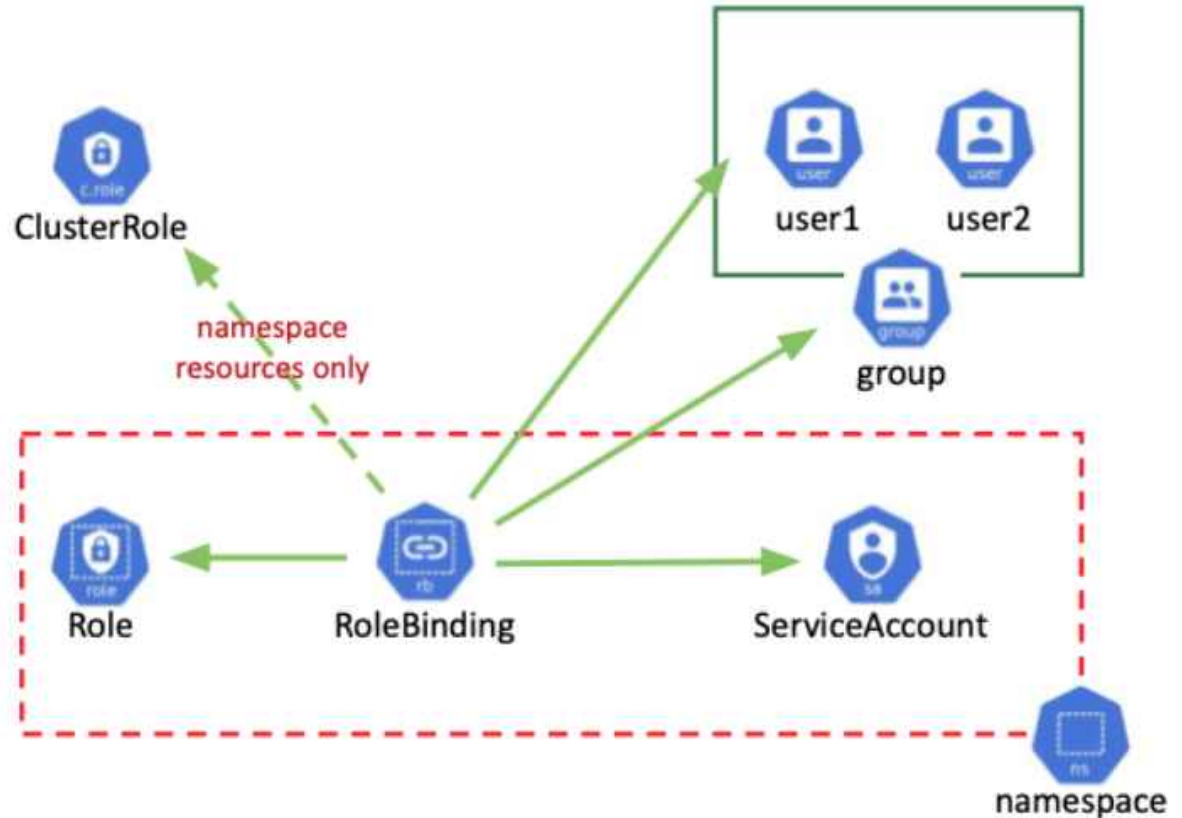
Deployments

...

Connected through access control

Kubernetes RBAC (Role Base Access Control)

- Âmbito do Namespace
 - Role
 - RoleBinding
- Âmbito do Cluster
 - ClusterRole
 - ClusterRoleBinding



Kubernetes RBAC (Role Base Access Control)

kubectl api-resources --namespaced=true

```
[root@ip-172-31-36-246 ~]# kubectl api-resources --namespaced=true | egrep " v1 \"\|\"apps"
bindings                v1      true      Binding
configmaps              cm       v1      true      ConfigMap
endpoints               ep       v1      true      Endpoints
events                  ev       v1      true      Event
limitranges             limits   v1      true      LimitRange
persistentvolumeclaims  pvc      v1      true      PersistentVolumeClaim
pods                    po       v1      true      Pod
podtemplates            v1      true      PodTemplate
replicationcontrollers  rc       v1      true      ReplicationController
resourcequotas          quota    v1      true      ResourceQuota
secrets                 v1      true      Secret
serviceaccounts         sa       v1      true      ServiceAccount
services                svc      v1      true      Service
controllerrevisions     v1      true      ControllerRevision
daemonsets              ds       apps/v1  true      DaemonSet
deployments             deploy   apps/v1  true      Deployment
replicasets             rs       apps/v1  true      ReplicaSet
statefulsets            sts      apps/v1  true      StatefulSet
```

kubectl api-resources --namespaced=false

```
[root@ip-172-31-36-246 ~]# kubectl api-resources --namespaced=false | grep " v1 "
componentstatuses      cs       v1      false     ComponentStatus
namespaces             ns       v1      false     Namespace
nodes                  no       v1      false     Node
persistentvolumes      pv       v1      false     PersistentVolume
```

Kubernetes RBAC (Role Base Access Control)

defined locally

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: deployer
  namespace: ns1

rules:
- verbs: ["create"]
  apiGroups: ["apps"]
  resources: ["deployments"]
```

granted locally

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bob-deployer
  namespace: ns1

roleRef:
  kind: Role
  apiGroup: rbac.authorization.k8s.io
  name: deployer

subjects:
- kind: User
  apiGroup: rbac.authorization.k8s.io
  name: bob
```

Kubernetes RBAC (Role Base Access Control)

defined globally

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: deployer

rules:
- verbs: ["create"]
  apiGroups: ["apps"]
  resources: ["deployments"]
```

granted locally

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bob-deployer
  namespace: ns1

roleRef:
  kind: ClusterRole
  apiGroup: rbac.authorization.k8s.io
  name: deployer

subjects:
- kind: User
  apiGroup: rbac.authorization.k8s.io
  name: bob
```

Kubernetes RBAC (Role Base Access Control)

defined globally

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: deployer
```

```
rules:
- verbs: ["create"]
  apiGroups: ["apps"]
  resources: ["deployments"]
```

granted globally

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bob-deployer
```

```
roleRef:
  kind: ClusterRole
  apiGroup: rbac.authorization.k8s.io
  name: deployer

subjects:
- kind: User
  apiGroup: rbac.authorization.k8s.io
  name: bob
```

Security Context

Serve para restringir permissão de um container, como por exemplo restringir uso do root, escrita em file system, ou dar capacidades extras ao container como por exemplo acesso a rede do host. Pode ser declarada em:

- Pod Level
- Container Level (possui precedência)

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

Pod Level

Container Level

Network Policy

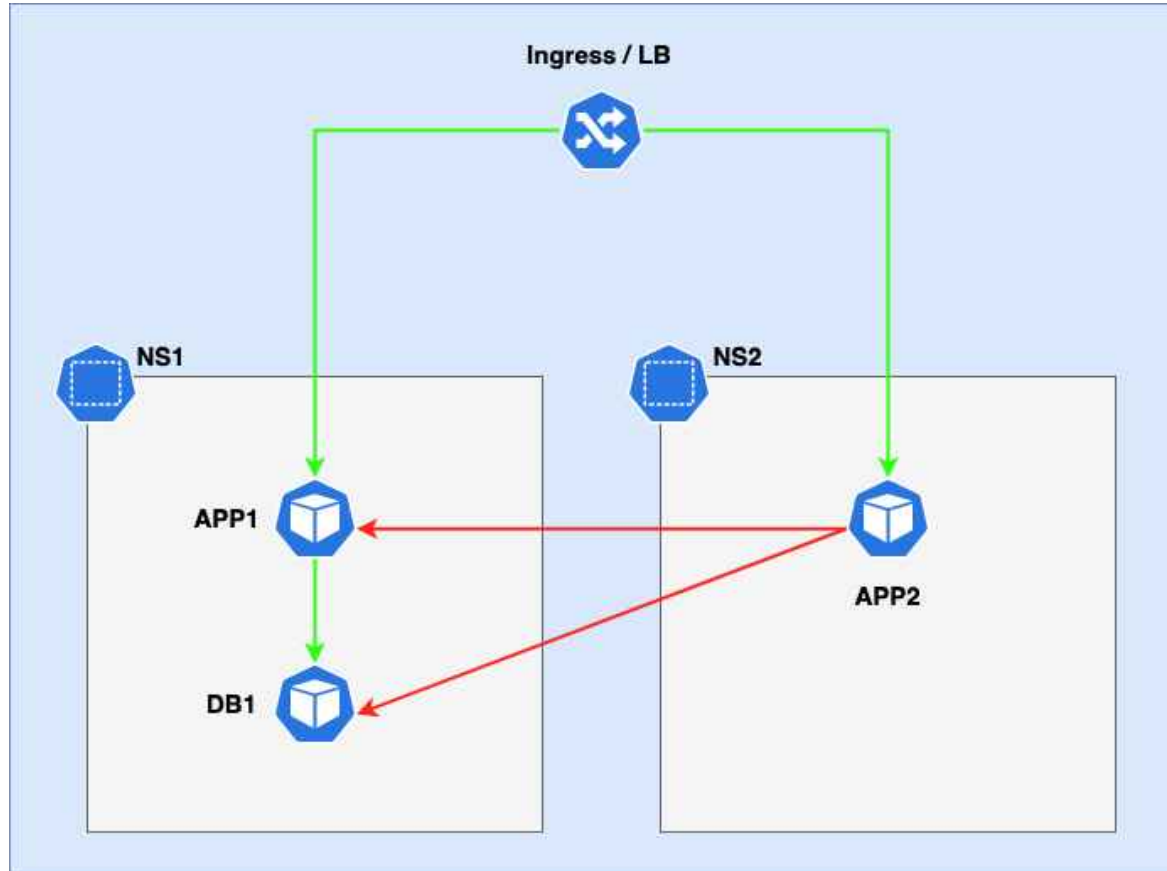
Naturalmente o Kubernetes foi projetado para permitir comunicação entre todos os pods de todos os namespaces independente do nó que os mesmo estiverem, porém a não restrição desta comunicação pode resultar numa enorme dor de cabeça com a segurança dos workloads ali envolvidos.

A NetPol é um objeto do kubernetes focado em aplicações que permite especificar como é permitido a um pod comunicar-se com várias "entidades" de rede. Essas entidades são:

- Outros Pods
- Namespaces
- Blocos de IP (interno ou externo ao cluster)

As network policies possuem dois tipos de políticas: **ingress** e **egress**, não confundir com o ingress controller, pois neste caso os termos referem-se aos tráfegos entrantes e saíntes dos pods respectivamente.

Network Policy



Network Policy

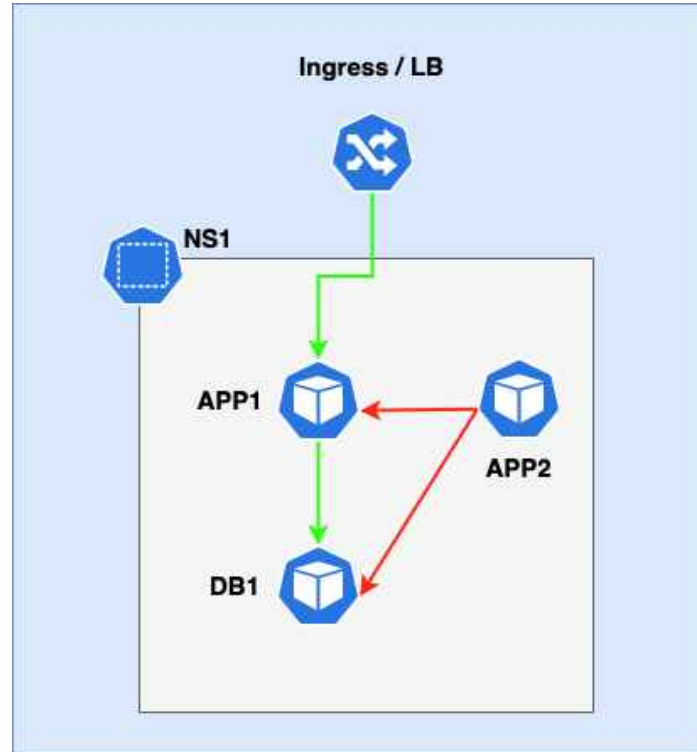


Image Security

Em casos em que a imagem fica em um repositório privado ao invés de público, devemos criar uma secret do tipo docker registry no namespace e referencia-la no manifesto de pod, desta maneira, o kubernetes no momento do pull oferecerá as credenciais necessárias para concluir o processo.

Temos dois atributos que influenciam diretamente no tratamento de imagem no manifesto do Pod:

imagePullSecrets: Contém as credenciais do registry remoto referenciado no campo image.

ImagePullPolicy: Diz em que condições o kubernetes (kubelet) deverá ir externamente no registry caso a imagem não esteja presente no cache do nó worker, poderá ser: **Always, Never, IfNotPresent.**

Resource Limits

Além de obviamente limitar o consumo de uma aplicação, podemos entender o resource limit como um recurso de segurança, visto que podemos ter uma aplicação mal comportada que poderá vir a consumir todos os recursos de um nó worker levando a uma comprometimento considerável do cluster. A limitação é feita da seguinte maneira:

CPU: Throttle

Mem: Terminated

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    - name: app
      image: images.my-company.example/app:v4
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
    - name: log-aggregator
      image: images.my-company.example/log-aggregator:v6
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

Aplicação de Referência

Fonte: <https://github.com/paulovigne/goapp>

