

THE DEVELOPER'S  
CONFERENCE

# Deployando Hyperledger Fabric com Kubernetes

Cláudio Ramos



## **Cláudio Ramos** **Senior Developer, SAP**

*Graduado em Ciências da Computação - UNISINOS*

*Treinamentos sobre Blockchain as a Service em SAP Cloud Platform*  
*Avaliação de viabilidade de projetos referentes a blockchains permissionadas*

*Membro da Hyperledger Community POA*

*Contribuidor no projeto Hyperledger Fabric da Linux Foundation*



# HLF → K8S

A hitchhiker's guide to deploying  
Hyperledger Fabric on Kubernetes

<https://event.on24.com/wcc/r/1819160/1D0ED8AC50AC8CB6E0158C1BE45B25E7>



# Três passos para ir LIVE com HLF & K8S



## 1 Hyperledger Fabric

Projeto Hyperledger Fabric e como seus componentes interagem

## 2 Kubernetes

Framework Kubernetes, Helm Charts e building blocks básicos

## 3 Deployando Hyperledger Fabric com Kubernetes

Passo-a-passo para deploy de uma rede usando Helm charts para CA, Orderer e Peers

## 4 Próximos passos

Melhorias futuras e como contribuir



# HFL direcionado a KS8

## Fabric CA

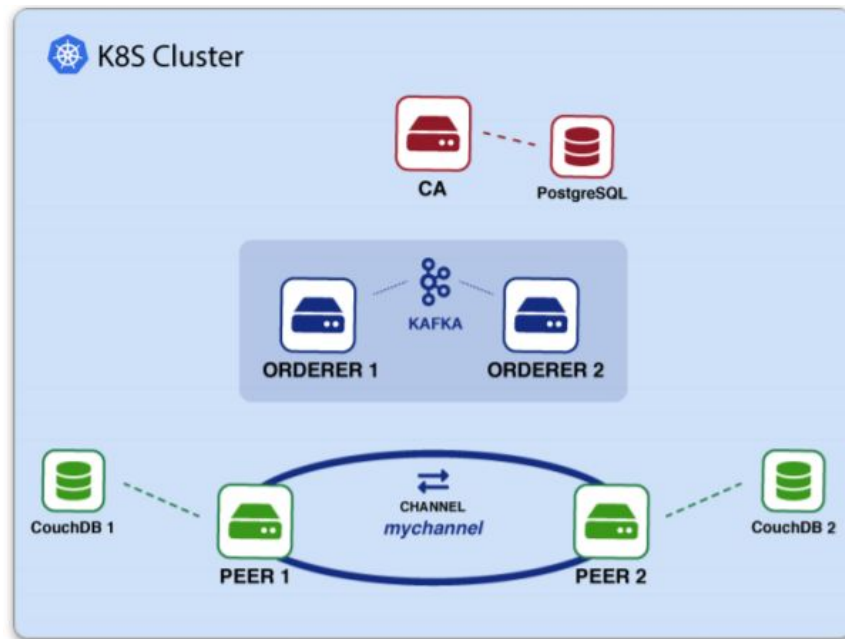
Fabric Certificate Authority  
registro & definição de identidades

## Fabric Orderer

Serviço Fabric Ordering provém  
consenso em redes para desenvolvimento  
(solo) e produção (Kafka).

## Fabric Peer

Fabric Peer gerencia a blockchain  
através da comunicação com o  
serviço de ordenação (ordering)





THE  
DEVELOPER'S  
CONFERENCE

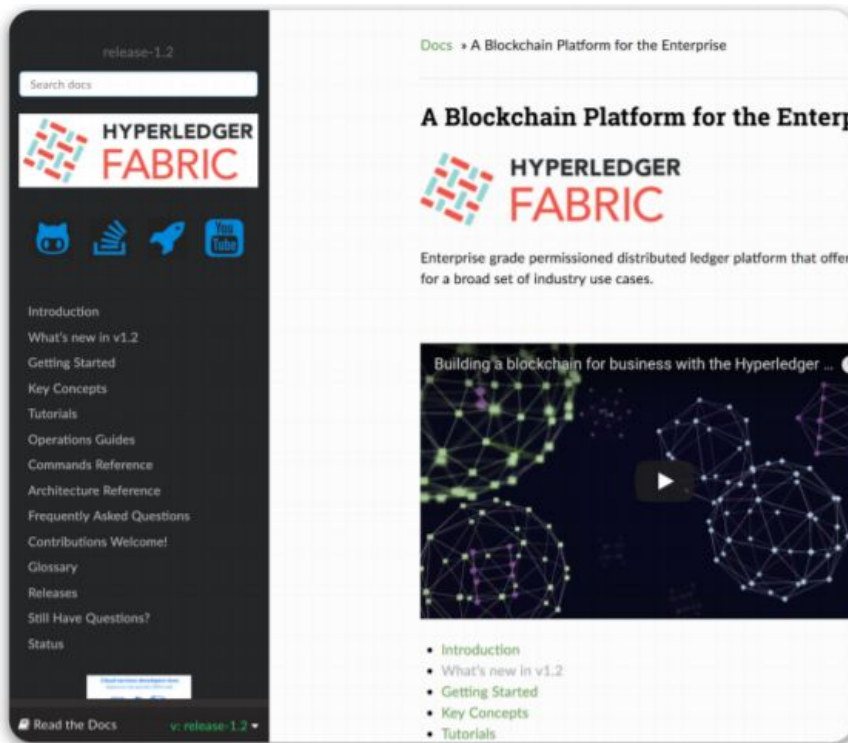
# Hyperledger Fabric

# Hyperledger Fabric

Fabric é um dos frameworks do projeto Hyperledger.

Permite a construção de redes de **blockchain permissionadas**.

<https://hyperledger-fabric.readthedocs.io/>





**Hyperledger Fabric** precisa de 3 tipos de nodo para ser deployado em produção



Fabric **Certificate Authority** para registro e definição de identidades



Fabric **Ordering Service** provendo consenso para redes solo e KAFKA



Fabric **Peer** para gerenciar a blockchain pela comunicação com Ordering Service





# Certificate Authority

Fabric CA gerencia identidades, uma vez que:

- Registra **identidades**
- Emite **ECerts**
- Renova e revoga certificados

<https://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html>

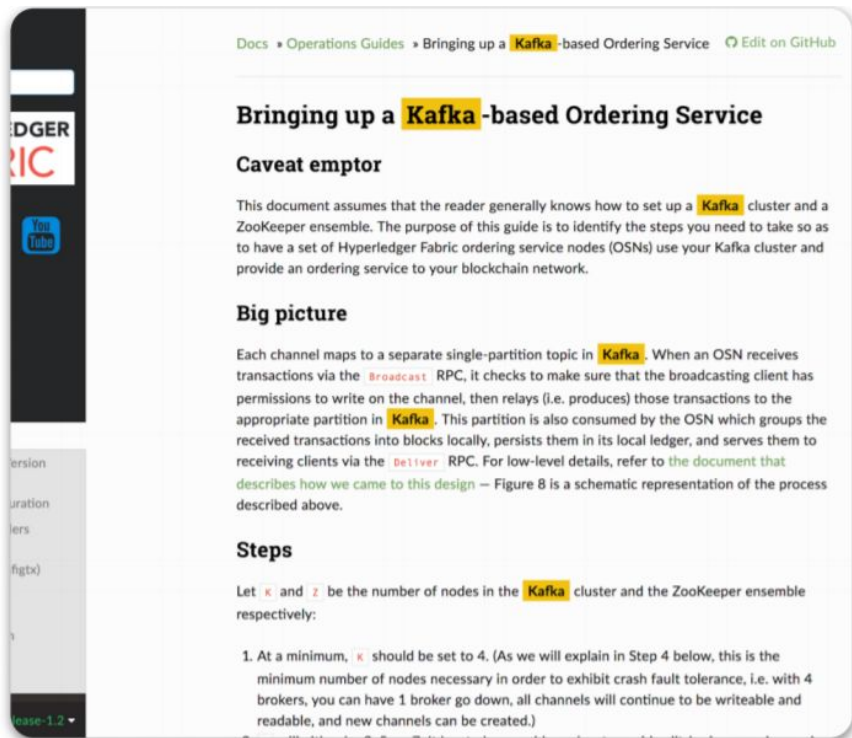
The image shows a screenshot of the Hyperledger Fabric CA User's Guide documentation page. The page is titled "Fabric CA User's Guide" and is part of the "latest" version of the documentation. The page content includes a search bar, navigation links for "Rocket Chat", "CI", and "StackOverflow", and a "GETTING STARTED" section. The "GETTING STARTED" section includes a "Table of Contents" with links to "Overview", "Getting Started", "Fabric CA Server", "Fabric CA Client", "Getting Idemix CRI (Certificate Revocation Information)", "HSM", "File Formats", and "Troubleshooting". The main content area includes a "Table of Contents" with links to "1. Overview", "2. Getting Started", "3. Configuration Settings", and "4. Fabric CA Server". The "Getting Started" section includes a list of features such as "registration of identities, or connects to LDAP as the user registry", "issuance of Enrollment Certificates (ECerts)", and "certificate renewal and revocation". The "Configuration Settings" section includes a link to "1. A word on file paths".



# Ordering Service

Ordering Service consiste em um grupo de nodos Orderer que estabelecem **consenso** através da troca de mensagens via cluster KAFKA

<https://hyperledger-fabric.readthedocs.io/en/release-1.2/kafka.html>



The screenshot shows a web browser displaying the Hyperledger Fabric documentation page for "Bringing up a Kafka-based Ordering Service". The page title is "Bringing up a Kafka-based Ordering Service" and it includes a "Caveat emptor" section, a "Big picture" section, and a "Steps" section. The "Steps" section begins with "Let  $k$  and  $z$  be the number of nodes in the Kafka cluster and the ZooKeeper ensemble respectively:" and lists a single step: "1. At a minimum,  $k$  should be set to 4. (As we will explain in Step 4 below, this is the minimum number of nodes necessary in order to exhibit crash fault tolerance, i.e. with 4 brokers, you can have 1 broker go down, all channels will continue to be writeable and readable, and new channels can be created.)"

Docs » Operations Guides » Bringing up a Kafka-based Ordering Service [Edit on GitHub](#)

## Bringing up a Kafka-based Ordering Service

### Caveat emptor

This document assumes that the reader generally knows how to set up a Kafka cluster and a ZooKeeper ensemble. The purpose of this guide is to identify the steps you need to take so as to have a set of Hyperledger Fabric ordering service nodes (OSNs) use your Kafka cluster and provide an ordering service to your blockchain network.

### Big picture

Each channel maps to a separate single-partition topic in Kafka. When an OSN receives transactions via the Broadcast RPC, it checks to make sure that the broadcasting client has permissions to write on the channel, then relays (i.e. produces) those transactions to the appropriate partition in Kafka. This partition is also consumed by the OSN which groups the received transactions into blocks locally, persists them in its local ledger, and serves them to receiving clients via the Deliver RPC. For low-level details, refer to the document that describes how we came to this design — Figure 8 is a schematic representation of the process described above.

### Steps

Let  $k$  and  $z$  be the number of nodes in the Kafka cluster and the ZooKeeper ensemble respectively:

1. At a minimum,  $k$  should be set to 4. (As we will explain in Step 4 below, this is the minimum number of nodes necessary in order to exhibit crash fault tolerance, i.e. with 4 brokers, you can have 1 broker go down, all channels will continue to be writeable and readable, and new channels can be created.)

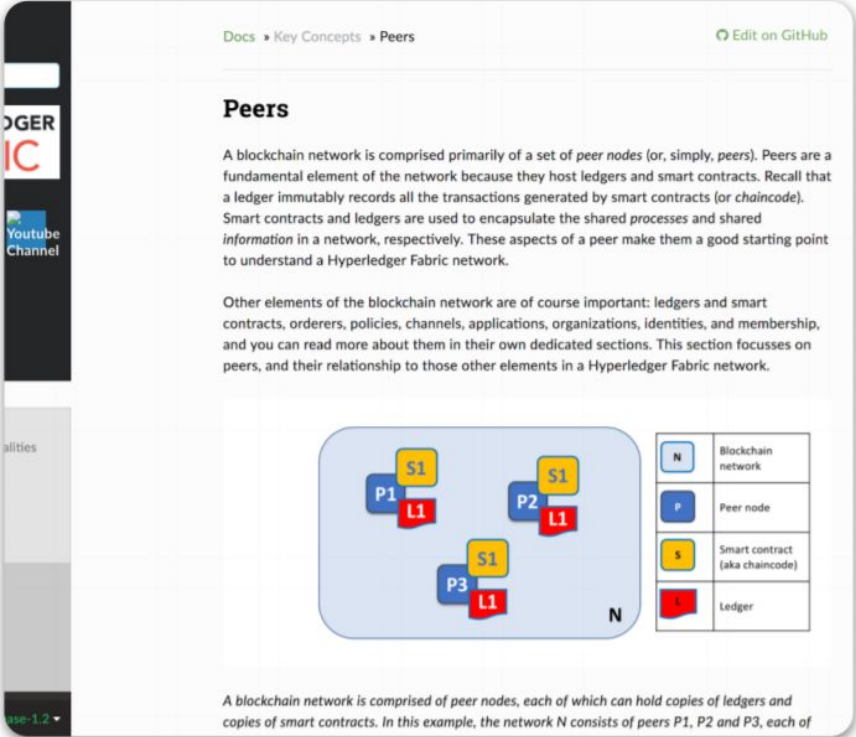


# Peer

Peers são os únicos nodos que realmente gravam a blockchain ledger.

Podemos anexar CouchDB para manter e fazer pesquisas sobre o **estado geral** da blockchain

<https://hyperledger-fabric.readthedocs.io/en/release-1.2/peers/peers.html>



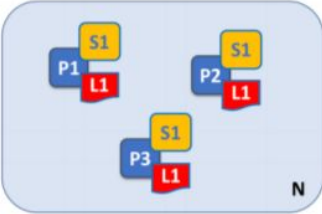
The screenshot shows a web page titled "Peers" from the Hyperledger Fabric documentation. The page includes a navigation bar with "Docs" and "Key Concepts" menus, and a link to "Edit on GitHub". The main content explains that a blockchain network is primarily composed of peer nodes, which host ledgers and smart contracts. It also lists other important elements like orderers, policies, channels, applications, organizations, identities, and membership. A diagram illustrates a network 'N' containing three peer nodes (P1, P2, P3), each with a smart contract (S1) and a ledger (L1). A legend table defines the symbols used in the diagram.

Docs » Key Concepts » Peers Edit on GitHub

## Peers

A blockchain network is comprised primarily of a set of *peer nodes* (or, simply, *peers*). Peers are a fundamental element of the network because they host ledgers and smart contracts. Recall that a ledger immutably records all the transactions generated by smart contracts (or *chaincode*). Smart contracts and ledgers are used to encapsulate the shared *processes* and shared *information* in a network, respectively. These aspects of a peer make them a good starting point to understand a Hyperledger Fabric network.

Other elements of the blockchain network are of course important: ledgers and smart contracts, orderers, policies, channels, applications, organizations, identities, and membership, and you can read more about them in their own dedicated sections. This section focuses on peers, and their relationship to those other elements in a Hyperledger Fabric network.



N	Blockchain network
P	Peer node
S	Smart contract (aka chaincode)
L	Ledger

A blockchain network is comprised of peer nodes, each of which can hold copies of ledgers and copies of smart contracts. In this example, the network N consists of peers P1, P2 and P3, each of





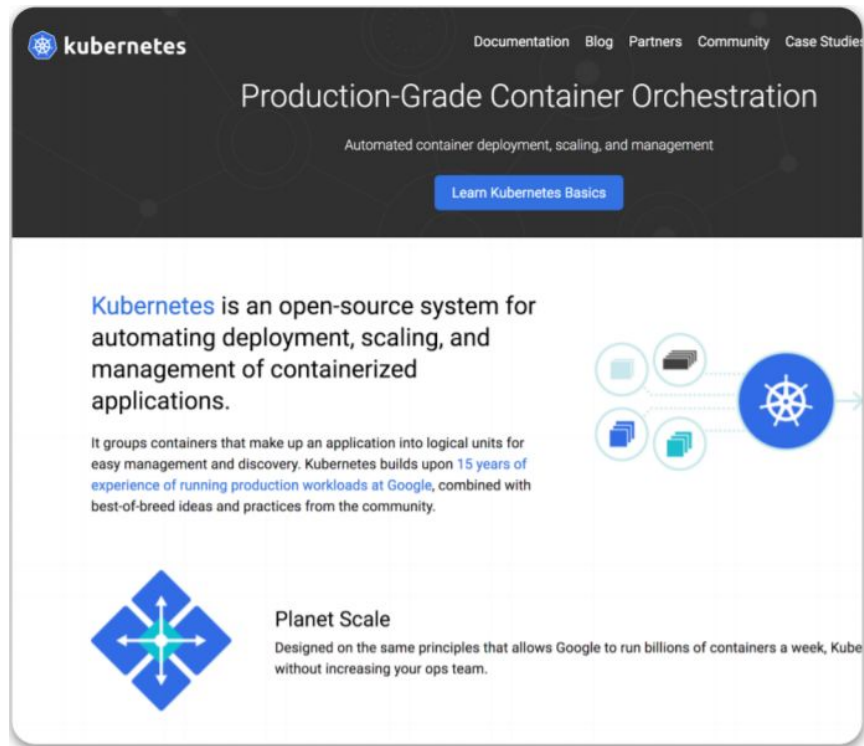
THE  
DEVELOPER'S  
CONFERENCE

# Kubernetes

# Kubernetes

Kubernetes é um sistema para orquestrar o deploy e gerenciamento de containers (e. g. Docker)

<https://kubernetes.io/>



The image shows a screenshot of the Kubernetes website homepage. At the top, there is a navigation bar with the Kubernetes logo and the word "kubernetes" in lowercase. To the right of the logo, there are links for "Documentation", "Blog", "Partners", "Community", and "Case Studies". Below the navigation bar, the main heading reads "Production-Grade Container Orchestration" in a large, white font. Underneath this heading, a smaller line of text says "Automated container deployment, scaling, and management". A blue button with the text "Learn Kubernetes Basics" is positioned below the text. The main content area has a white background and features a large blue heading: "Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications." To the right of this text is a diagram consisting of four circular icons (a folder, a server rack, a document, and a server) connected by dotted lines to a central blue circle containing the Kubernetes logo. Below the main heading, there is a paragraph of text: "It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community." At the bottom left of the main content area, there is a blue diamond-shaped logo with a white cross and four arrows pointing outwards. To the right of this logo, the text "Planet Scale" is displayed in a bold font, followed by a paragraph: "Designed on the same principles that allows Google to run billions of containers a week, Kubernetes without increasing your ops team."





Para trabalhar com **Kubernetes**, precisamos entender 3 conceitos principais



O Kubernetes **Cluster** gerencia conjunto de abstrações que tornam fácil orquestrar containers



**Charts** são pacotes junto a templates de Kubernetes, atuando como facilitadores



**Helm** (e seu modo server-side Tiller) permitem a instalação de Charts

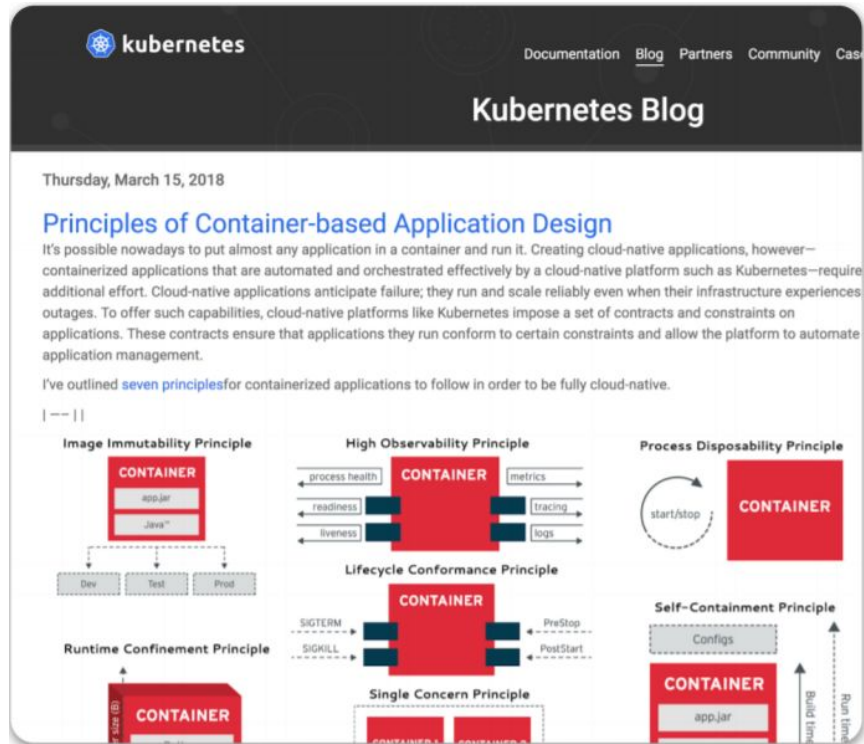


# Kubernetes cluster

Kubernetes (K8S) nos permite agrupar, fazer deploy, escalar, realizar auto-ajuste e conectar a containers

K8S orquestra containers

<https://kubernetes.io/blog/2018/03/principles-of-container-app-design/>



The image shows a screenshot of the Kubernetes Blog page. At the top, the Kubernetes logo and name are visible, along with navigation links for Documentation, Blog, Partners, Community, and Cases. The main heading is "Kubernetes Blog" and the date is "Thursday, March 15, 2018". The article title is "Principles of Container-based Application Design". The text discusses the challenges of containerized applications and the need for standardized principles. It lists seven principles: Image Immutability Principle, High Observability Principle, Process Disposability Principle, Lifecycle Conformance Principle, Runtime Confinement Principle, Single Concern Principle, and Self-Containment Principle. Each principle is accompanied by a diagram illustrating its concept.

Thursday, March 15, 2018

## Principles of Container-based Application Design

It's possible nowadays to put almost any application in a container and run it. Creating cloud-native applications, however—containerized applications that are automated and orchestrated effectively by a cloud-native platform such as Kubernetes—require additional effort. Cloud-native applications anticipate failure; they run and scale reliably even when their infrastructure experiences outages. To offer such capabilities, cloud-native platforms like Kubernetes impose a set of contracts and constraints on applications. These contracts ensure that applications they run conform to certain constraints and allow the platform to automate application management.

I've outlined [seven principles](#) for containerized applications to follow in order to be fully cloud-native.

| — ||

- Image Immutability Principle**: A container (red box) contains 'app.jar' and 'java'. It is deployed to 'Dev', 'Test', and 'Prod' environments (grey boxes).
- High Observability Principle**: A container (red box) has bidirectional arrows for 'process health', 'readiness', and 'liveness' on the left, and 'metrics', 'tracing', and 'logs' on the right.
- Process Disposability Principle**: A container (red box) with a circular arrow labeled 'start/stop' around it.
- Lifecycle Conformance Principle**: A container (red box) with bidirectional arrows for 'SIGTERM' and 'SIGKILL' on the left, and 'PreStop' and 'PostStart' on the right.
- Runtime Confinement Principle**: A container (red box) with a dashed box around it labeled 'runtime (R)'.
- Single Concern Principle**: A container (red box) containing 'app.jar' and 'java', with a dashed box around it labeled 'CONTAINER 1' and 'CONTAINER 2' below it.
- Self-Containment Principle**: A container (red box) containing 'app.jar' and 'java', with a dashed box around it labeled 'Configs' above it. Arrows indicate 'Build time' and 'Run time' phases.

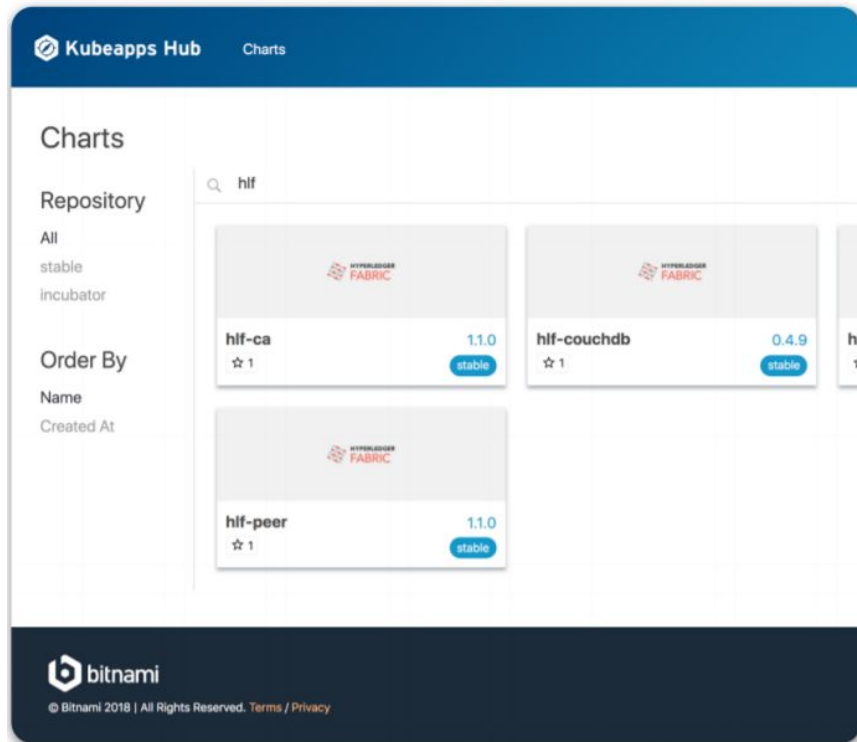


# Charts

Charts são pacotes de templates que definem componentes de Kubernetes

É possível buscar por Charts no Kubeapps Hub

<https://hub.kubeapps.com/charts?q=hlf>



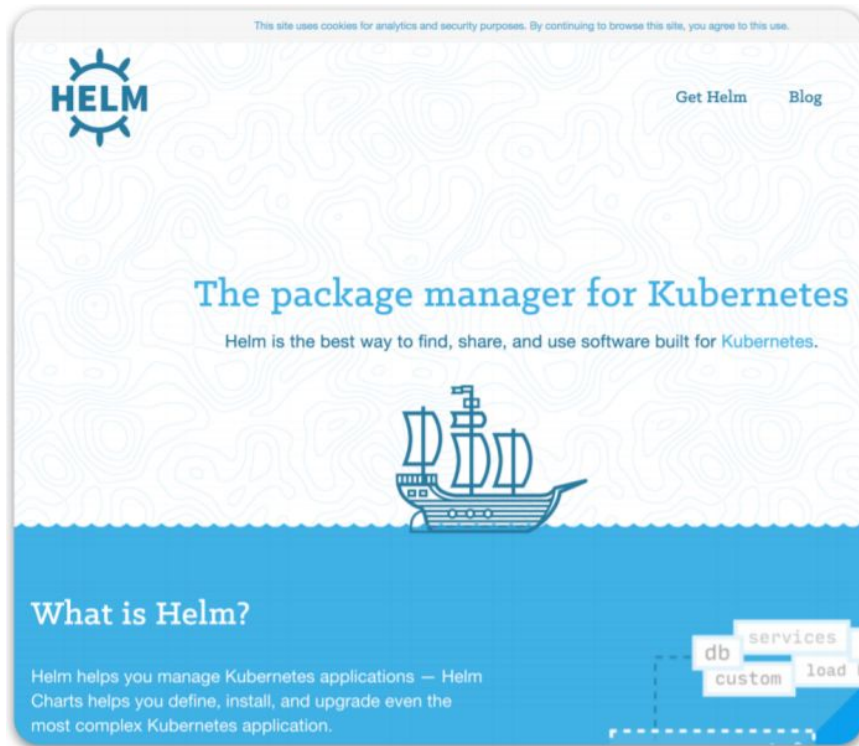


# Helm

Helm permite o empacotamento, pesquisa e deploy de Charts

Seu componente server-side, Tiller, gerencia o deployment em K8S

<https://www.helm.sh/>





Antes de começar, são necessárias 3 coisas



Setup do **Kubernetes** cluster e instalação do Helm/Tiller nele



Obtenção de um **domain name** (gratuito / baratinho) para seu cluster



Download do **repositório** no endereço  
<https://github.com/aidtechnology/lf-k8s-hlf-webinar>



# Instale Ingress controller e cert manager

```
helm install stable/nginx-ingress -n nginx-ingress --namespace ingress-controller
```

```
helm install stable/cert-manager -n cert-manager --namespace cert-manager
```

```
kubectl create -f ./extra/certManagerCI_staging.yaml
```

```
kubectl create -f ./extra/certManagerCI_production.yaml
```





THE  
DEVELOPER'S  
CONFERENCE

# Deployando HLF no K8S

Começando a festa, certo meow!



# HFL direcionado a KS8

## Fabric CA

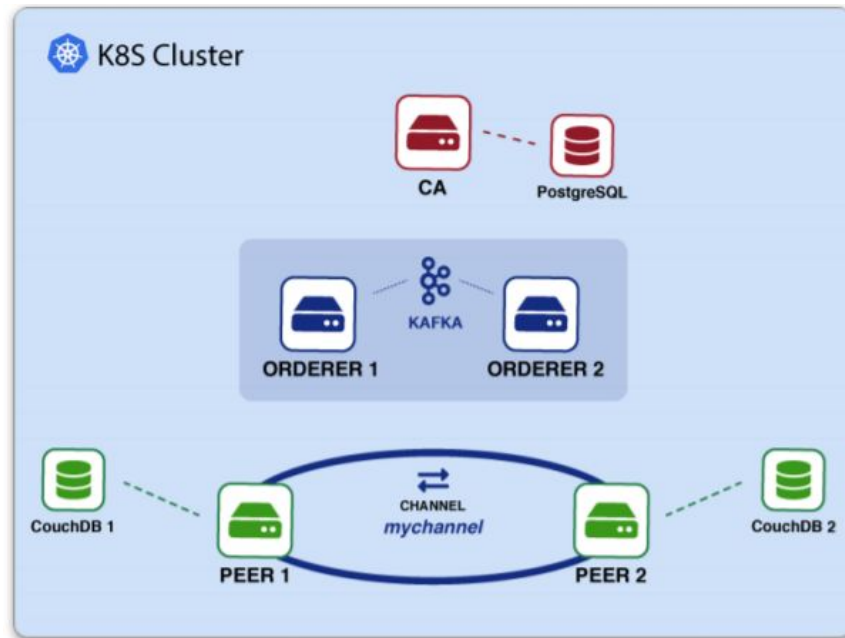
Fabric Certificate Authority  
registro & definição de identidades

## Fabric Orderer

Serviço Fabric Ordering provém  
consenso em redes para desenvolvimento  
(solo) e produção (Kafka).

## Fabric Peer

Fabric Peer gerencia a blockchain  
através da comunicação com o  
serviço de ordenação (ordering)





THE  
DEVELOPER'S  
CONFERENCE

# Fabric CA

# Fabric CA

 Instalar CA Database Helm Chart

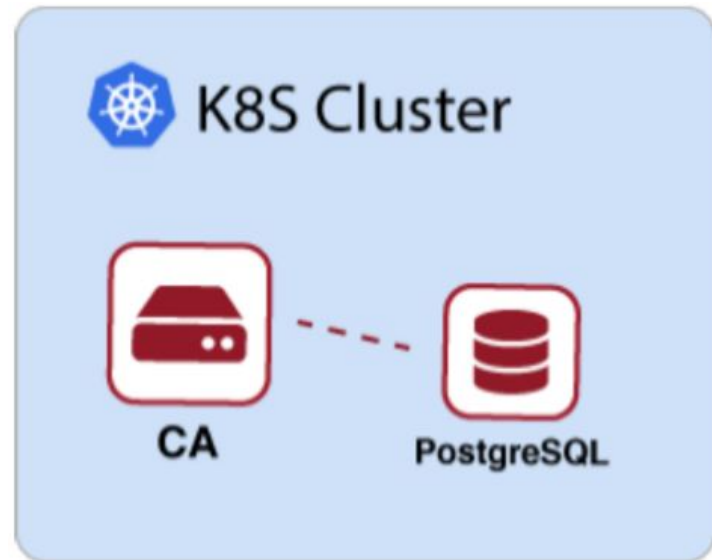
 Instalar Fabric CA Helm Chart

 Gerar Fabric CA Identity

 Obter Crypto Material

 Salvar Crypto Material no K8S

 Gerar Genesis e Channel



# Instalar CA Database Helm Chart



Instalar CA Database  
Helm Chart

```
helm install stable/postgresql -n ca-pg --namespace  
blockchain -f ./helm_values/ca-pg_values.yaml
```





```
EXPLORER
! ca-pg_values.yaml helm_values
LF-K8S-HLF-WEBINAR
  .idea
  config
  extra
  helm_values
    ! ca_values.yaml
    ! ca-pg_values.yaml
    ! cdb-peer1_values.yaml
    ! cdb-peer2_values.yaml
    ! kafka-hlf_values.yaml
    ! ord1_values.yaml
    ! ord2_values.yaml
    ! peer1_values.yaml
    ! peer2_values.yaml
  .gitignore
  LICENSE
  README.md

! ca-pg_values.yaml x
1 imageTag: "9.6.2"
2
3 # postgresPassword:
4 postgresDatabase: fabric_ca
5
6 persistence:
7   size: 1Gi
8   storageClass: "managed-premium"
9
10 affinity:
11   podAntiAffinity:
12     requiredDuringSchedulingIgnoredDuringExecution:
13       - topologyKey: "kubernetes.io/hostname"
14         labelSelector:
15           matchLabels:
16             app: postgresql
17
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF YAML



# Instalar Cabric CA Helm Chart



Instalar CA Database  
Helm Chart



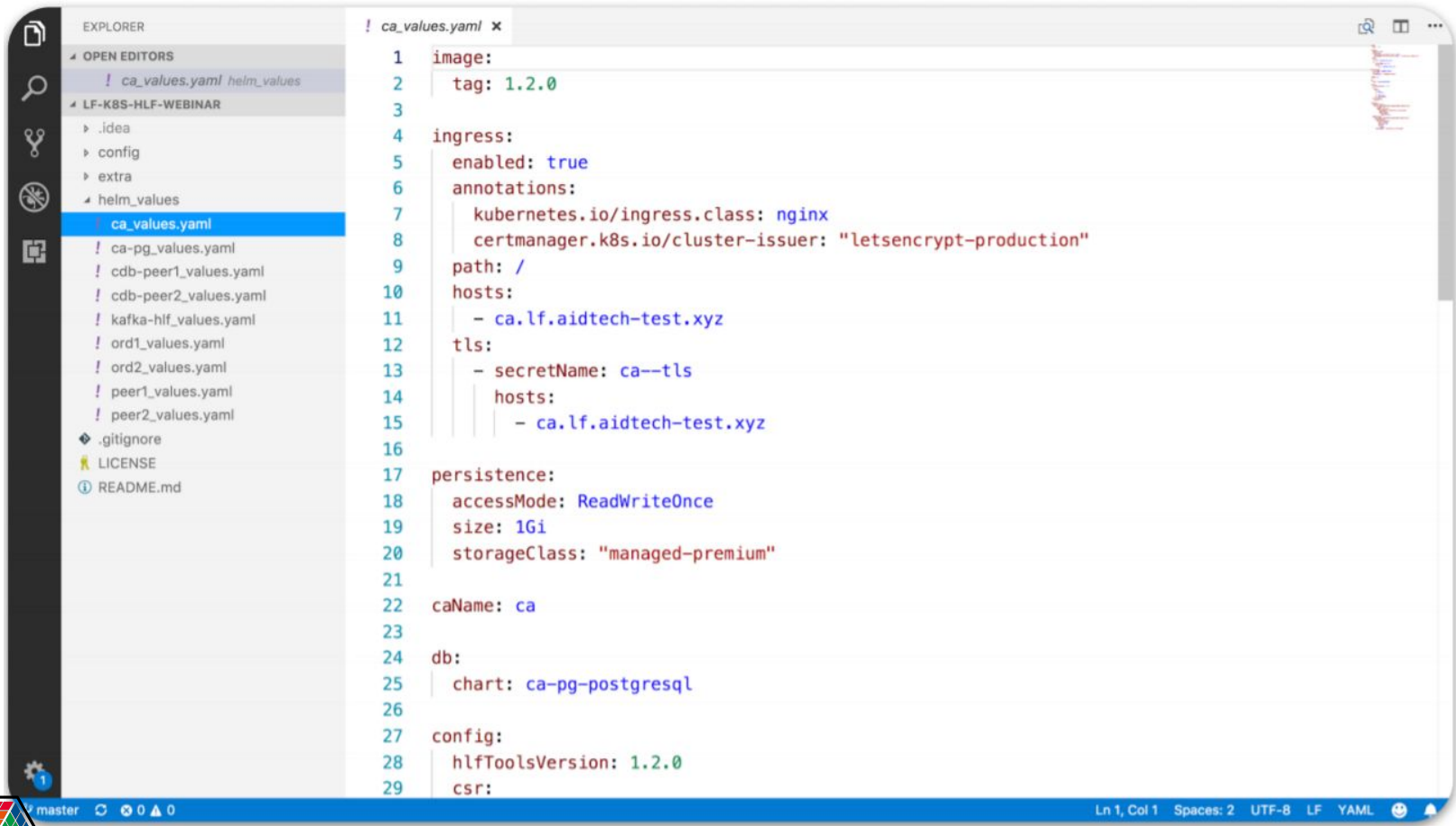
Instalar Fabric CA  
Helm Chart

```
helm install stable/hlf-ca -n ca --namespace blockchain -f  
./helm_values/ca_values.yaml
```

```
CA_POD=$(kubectl get pods -n blockchain -l "app=hlf-  
ca,release=ca" -o jsonpath="{.items[0].metadata.name}")
```

```
kubectl logs -n blockchain $CA_POD | grep 'Listening on'
```





```
1 image:
2   tag: 1.2.0
3
4 ingress:
5   enabled: true
6   annotations:
7     kubernetes.io/ingress.class: nginx
8     certmanager.k8s.io/cluster-issuer: "letsencrypt-production"
9   path: /
10  hosts:
11    - ca.lf.aidtech-test.xyz
12  tls:
13    - secretName: ca--tls
14      hosts:
15        - ca.lf.aidtech-test.xyz
16
17 persistence:
18   accessMode: ReadWriteOnce
19   size: 1Gi
20   storageClass: "managed-premium"
21
22 caName: ca
23
24 db:
25   chart: ca-pg-postgresql
26
27 config:
28   hlfToolsVersion: 1.2.0
29  csr:
```



```
27 config:
28   hlfToolsVersion: 1.2.0
29   csr:
30     names:
31       c: IE
32       st: Dublin
33       l:
34       o: "AID:Tech"
35       ou: Blockchain
36   affiliations:
37     aidtech: []
38
39 affinity:
40   podAntiAffinity:
41     preferredDuringSchedulingIgnoredDuringExecution:
42       - weight: 95
43         podAffinityTerm:
44           topologyKey: "kubernetes.io/hostname"
45           labelSelector:
46             matchLabels:
47               app: hlf-ca
48   podAffinity:
49     requiredDuringSchedulingIgnoredDuringExecution:
50       - labelSelector:
51         matchExpressions:
52           - key: release
53             operator: In
54             values:
55               - ca-pg
```



# Gerar Fabric CA Identity



Instalar CA Database  
Helm Chart



Instalar Fabric CA  
Helm Chart



Gerar Fabric CA  
Identity

```
kubectl exec -n blockchain $SCA_POD -- cat  
/var/hyperledger/fabric-ca/msp/signcerts/cert.pem
```

```
kubectl exec -n blockchain $SCA_POD -- bash -c 'fabric-ca-client  
enroll -d -u  
http://$SCA_ADMIN:$SCA_PASSWORD@$SERVICE_DNS:7054'
```

```
CA_INGRESS=$(kubectl get ingress -n blockchain -l "app=hlf-  
ca,release=ca" -o jsonpath="{.items[0].spec.rules[0].host}")
```

```
curl https://$CA_INGRESS/cainfo
```



# Obter crypto material



Instalar CA Database  
Helm Chart



Instalar Fabric CA  
Helm Chart



Gerar Fabric CA  
Identity



Obter Crypto Material

```
FABRIC_CA_CLIENT_HOME=./config fabric-ca-client getcert -u  
http://$SCA_INGRESS -M ./AidTechMSP
```

```
kubectl exec -n blockchain $SCA_POD -- fabric-ca-client register --  
id.name org-admin --id.secret OrgAdminPW --id.attrs 'admin=true:ecert'
```

```
FABRIC_CA_CLIENT_HOME=./config fabric-ca-client enroll -u  
http://org-admin:OrgAdminPW@$SCA_INGRESS -M ./AidTechMSP
```

```
mkdir -p ./config/AidTechMSP/admincerts
```

```
cp ./config/AidTechMSP/signcerts/* ./config/AidTechMSP/admincerts
```



```
49 #####
50 # TLS section for secure socket connection
51 #
52 #
53 # certfiles - PEM-encoded list of trusted root certificate files
54 # client:
55 #   certfile - PEM-encoded certificate file for when client authentication
56 #   is enabled on server
57 #   keyfile - PEM-encoded key file for when client authentication
58 #   is enabled on server
59 #####
60 tls:
61   enabled: false
62   # TLS section for secure socket connection
63   certfiles:
64     - ./Lets_Encrypt_Authority_X3.pem
65   client:
66     certfile:
67     keyfile:
68
69 #####
70 # Certificate Signing Request section for generating the CSR for an
71 # enrollment certificate (ECert)
72 #
73 # cn - Used by CAs to determine which domain the certificate is to be generated for
74 #
75 # serialnumber - The serialnumber field, if specified, becomes part of the issued
76 #   certificate's DN (Distinguished Name). For example, one use case for this is
77 #   a company with its own CA (Certificate Authority) which issues certificates
```



# Salvar crypto material no K8S



Instalar CA Database  
Helm Chart



Instalar Fabric CA  
Helm Chart



Gerar Fabric CA  
Identity



Obter Crypto Material



Salvar Crypto Material  
no K8S

```
ORG_CERT=$(ls ./config/AidTechMSP/admincerts/cert.pem)
```

```
kubectl create secret generic -n blockchain hlf--org-admincert  
--from-file=cert.pem=$ORG_CERT
```

```
ORG_KEY=$(ls ./config/AidTechMSP/keystore/*_sk)
```

```
kubectl create secret generic -n blockchain hlf--org-adminkey  
--from-file=key.pem=$ORG_KEY
```





```
EXPLORER
! configtx.yaml x
OPEN EDITORS
! configtx.yaml config
LF-KBS-HLF-WEBINAR
  .idea
  config
  AidTechMSP
  config
  configtx.yaml
fabric-ca-client-config.yaml
genesis.block
Lets_Encrypt_Authority_X3.pem
mychannel.tx
extra
helm_values
.gitignore
LICENSE
README.md

7 #####
8 #
9 # Profile
10 #
11 # - Different configuration profiles may be encoded here to be specified
12 # as parameters to the configtxgen tool
13 #
14 #####
15 Profiles:
16
17   OrdererGenesis:
18     Orderer:
19       <<: *OrdererDefaults
20     Organizations:
21       - *AidTech
22     Consortiums:
23       MyConsortium:
24         Organizations:
25           - *AidTech
26   MyChannel:
27     Consortium: MyConsortium
28     Application:
29       <<: *ApplicationDefaults
30     Organizations:
31       - *AidTech
32
33 #####
34 #
35 # Section: Organizations
```



```
EXPLORER                                ! configtx.yaml x
├── OPEN EDITORS
│   └── ! configtx.yaml config
├── LF-K8S-HLF-WEBINAR
│   ├── .idea
│   ├── config
│   │   ├── AidTechMSP
│   │   └── config
│   └── configtx.yaml
├── ! fabric-ca-client-config.yaml
├── genesis.block
├── Lets_Encrypt_Authority_X3.pem
├── mychannel.tx
├── extra
├── helm_values
├── .gitignore
├── LICENSE
└── README.md

38 # be referenced later in the configuration.
39 #
40 #####
41 Organizations:
42
43 # Organization controlling both peers and Orderers
44 - &AidTech
45   Name: AidTech
46
47   # ID to load the MSP definition as
48   ID: AidTechMSP
49
50   MSPDir: ./AidTechMSP
51
52   # turn off security for the peer
53   AdminPrincipal: Role.MEMBER
54
55   AnchorPeers:
56     # AnchorPeers defines the location of peers that can be used
57     # for cross org gossip communication. Note, this value is only
58     # encoded in the genesis block in the Application section context
59     - Host: peer1-hlf-peer.blockchain.svc.cluster.local
60       Port: 7051
61     - Host: peer2-hlf-peer.blockchain.svc.cluster.local
62       Port: 7051
63
64 #####
65 #
66 # SECTION: Orderer
```



```
78 Addresses:
79   - ord1-hlf-ord.blockchain.svc.cluster.local:7050
80   - ord2-hlf-ord.blockchain.svc.cluster.local:7050
81
82 # Batch Timeout: The amount of time to wait before creating a batch
83 BatchTimeout: 2s
84
85 # Batch Size: Controls the number of messages batched into a block
86 BatchSize:
87
88   # Max Message Count: The maximum number of messages to permit in a batch
89   MaxMessageCount: 10
90
91   # Absolute Max Bytes: The absolute maximum number of bytes allowed for
92   # the serialized messages in a batch.
93   AbsoluteMaxBytes: 98 MB
94
95   # Preferred Max Bytes: The preferred maximum number of bytes allowed for
96   # the serialized messages in a batch. A message larger than the preferred
97   # max bytes will result in a batch larger than preferred max bytes.
98   PreferredMaxBytes: 512 KB
99
100 Kafka:
101   # Brokers: A list of Kafka brokers to which the orderer connects
102   # If using K8S, we specify the service exposing the brokers
103   # NOTE: Use Address/IP:port notation
104   Brokers:
105     - kafka-hlf.blockchain.svc.cluster.local:9092
106
```



# Gerar genesis & channel



Instalar CA Database  
Helm Chart



Instalar Fabric CA  
Helm Chart



Gerar Fabric CA  
Identity



Obter Crypto Material



Salvar Crypto Material  
no K8S



Gerar Genesis e  
Channel



```
cd ./config
```

```
configtxgen -profile OrdererGenesis -outputBlock ./genesis.block
```

```
configtxgen -profile MyChannel -channelID mychannel -  
outputCreateChannelTx ./mychannel.tx
```

```
kubectl create secret generic -n blockchain hlf--genesis --from-  
file=genesis.block
```

```
kubectl create secret generic -n blockchain hlf--channel --from-  
file=mychannel.tx
```



THE  
DEVELOPER'S  
CONFERENCE

# Fabric Orderer

# Fabric Orderer



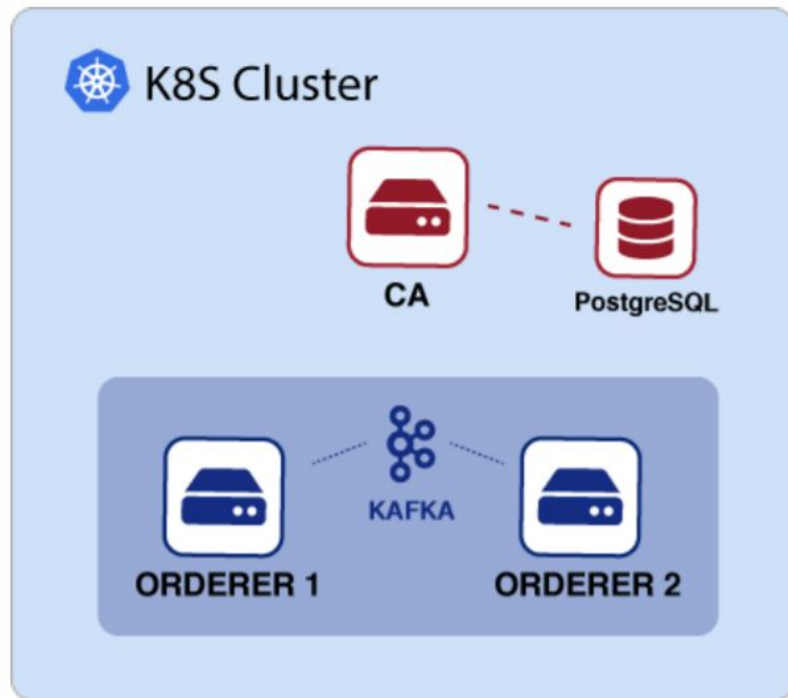
Instalar **Kafka Helm Chart**



Instalar **Fabric Orderer Helm Chart**



Definir **Fabric Orderer Identity**



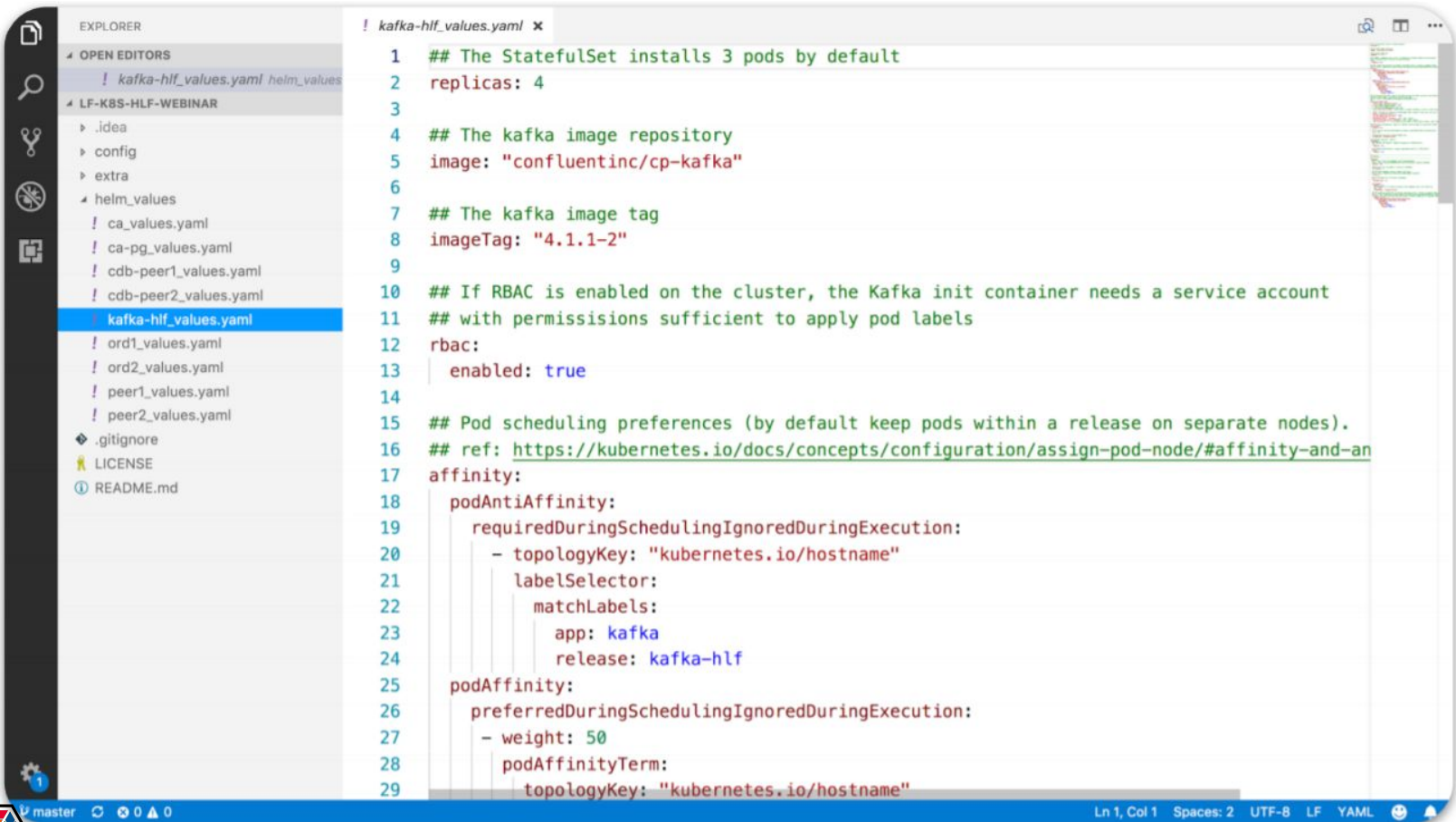
# Instalar Kafka Helm Chart



## Instalar Kafka Helm Chart

```
helm install incubator/kafka -n kafka-hlf --  
namespace blockchain -f ./helm_values/kafka-  
hlf_values.yaml
```





The image shows a code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like '.idea', 'config', 'extra', 'helm\_values', and files like 'ca\_values.yaml', 'ca-pg\_values.yaml', 'cdb-peer1\_values.yaml', 'cdb-peer2\_values.yaml', 'kafka-hlf\_values.yaml', 'ord1\_values.yaml', 'ord2\_values.yaml', 'peer1\_values.yaml', 'peer2\_values.yaml', '.gitignore', 'LICENSE', and 'README.md'. The code editor shows the content of 'kafka-hlf\_values.yaml' with the following text:

```
1  ## The StatefulSet installs 3 pods by default
2  replicas: 4
3
4  ## The kafka image repository
5  image: "confluentinc/cp-kafka"
6
7  ## The kafka image tag
8  imageTag: "4.1.1-2"
9
10 ## If RBAC is enabled on the cluster, the Kafka init container needs a service account
11 ## with permissions sufficient to apply pod labels
12 rbac:
13   enabled: true
14
15 ## Pod scheduling preferences (by default keep pods within a release on separate nodes).
16 ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-an
17 affinity:
18   podAntiAffinity:
19     requiredDuringSchedulingIgnoredDuringExecution:
20     - topologyKey: "kubernetes.io/hostname"
21       labelSelector:
22         matchLabels:
23           app: kafka
24           release: kafka-hlf
25   podAffinity:
26     preferredDuringSchedulingIgnoredDuringExecution:
27     - weight: 50
28       podAffinityTerm:
29         topologyKey: "kubernetes.io/hostname"
```





```
38 ##
39 configurationOverrides:
40   "offsets.topic.replication.factor": 3
41   # "auto.leader.rebalance.enable": true
42   # "controlled.shutdown.enable": true
43   # "controlled.shutdown.max.retries": 100
44   "auto.create.topics.enable": true # Useful to enable the Node.js client to create topics
45
46   # NOTE: The below are required for Hyperledger Fabric orderer to work (but last one is p
47   "default.replication.factor": 3
48   "unclean.leader.election.enable": false
49   "min.insync.replicas": 2
50   "message.max.bytes": "103809024" # 99 * 1024 * 1024 B
51   "replica.fetch.max.bytes": "103809024" # 99 * 1024 * 1024 B
52   "log.retention.ms": -1 # This should be only used for the HL Fabric Orderer (which need
53
54 ## Persistence configuration. Specify if and how to persist data to a persistent volume.
55 persistence:
56   enabled: true
57
58   ## The size of the PersistentVolume to allocate to each Kafka Pod in the StatefulSet
59   size: "1Gi"
60
61   ## Kafka data Persistent Volume Storage Class
62   storageClass: "managed-premium"
63
64 ## Prometheus Exporters / Metrics
65 prometheus:
66   ## Prometheus JMX Exporter: exposes the majority of Kafkas metrics
```



# Instalar Fabric Orgerer Helm Chart



Instalar Kafka Helm  
Chart



Instalar Fabric  
Orderer Helm Chart

```
export NUM=1
```

```
helm install stable/hlf-ord -n ord${NUM} --  
namespace blockchain -f  
./helm_values/ord${NUM}_values.yaml
```



```
1 image:
2   tag: 1.2.0
3
4 persistence:
5   accessMode: ReadWriteOnce
6   size: 1Gi
7   storageClass: "managed-premium"
8
9 caAddress: ca.lf.aidtech-test.xyz
10 caUsername: ord1
11
12 ord:
13   hlfToolsVersion: 1.2.0
14   type: kafka
15   ## MSP ID of the Orderer
16   mspID: AidTechMSP
17
18 secrets:
19   # This should contain "genesis" block derived from a configtx.yaml
20   # configtxgen -profile OrdererGenesis -outputBlock genesis.block
21   genesis: hlf--genesis
22   adminCert: hlf--org-admincert
23   caServerTls: ca--tls
24
25 affinity:
26   podAntiAffinity:
27     preferredDuringSchedulingIgnoredDuringExecution:
28       - weight: 95
29     podAffinityTerm:
```



# Definir Orderer Identity



Instalar Kafka Helm Chart



Instalar Fabric Orderer Helm Chart



Definir Fabric Orderer Identity

```
ORD_SECRET=$(kubectl get secret -n blockchain ord${NUM}-hlf-ord  
-o jsonpath="{.data.CA_PASSWORD}" | base64 --decode)
```

```
kubectl exec SCA_POD -n blockchain -- fabric-ca-client register --  
id.name ord${NUM} --id.secret SORD_SECRET --id.type orderer
```

```
ORD_POD=$(kubectl get pods -n blockchain -l "app=hlf-  
ord,release=ord${NUM}" -o jsonpath="{.items[0].metadata.name}")
```

```
kubectl logs -n blockchain SORD_POD | grep 'completeInitialization'
```





THE  
DEVELOPER'S  
CONFERENCE

# Fabric Peer

# Fabric Peer

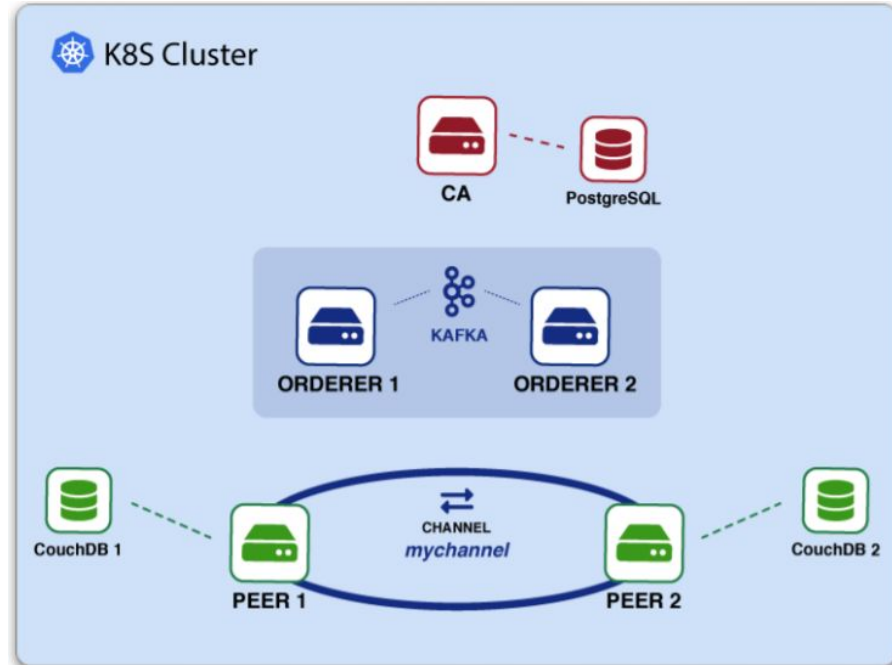
 Instalar CouchDB Helm Chart

 Instalar Fabric Peer Helm Chart

 Definir PeerIdentity

 Criar Channel

 Ingressar no Channel



# Instalar CouchDB Helm Chart



## Instalar CouchDB Helm Chart

```
export NUM=1
```

```
helm install stable/hlf-couchdb -n cdb-peer${NUM} --namespace  
blockchain -f ./helm_values/cdb-peer${NUM}_values.yaml
```

```
CDB_POD=$(kubectl get pods -n blockchain -l "app=hlf-  
couchdb,release=cdb-peer${NUM}" -o  
jsonpath="{.items[*].metadata.name}")
```

```
kubectl logs -n blockchain $CDB_POD | grep 'Apache CouchDB  
has started on'
```



```
EXPLORER
OPEN EDITORS
! cdb-peer1_values.yaml helm_val...
LF-K8S-HLF-WEBINAR
  .idea
  config
  extra
  helm_values
    ! ca_values.yaml
    ! ca-pg_values.yaml
    ! cdb-peer1_values.yaml
    ! cdb-peer2_values.yaml
    ! kafka-hlf_values.yaml
    ! ord1_values.yaml
    ! ord2_values.yaml
    ! peer1_values.yaml
    ! peer2_values.yaml
  .gitignore
  LICENSE
  README.md

! cdb-peer1_values.yaml x
1 image:
2   tag: 0.4.10
3
4 persistence:
5   size: 1Gi
6   storageClass: "managed-premium"
7
8 affinity:
9   podAntiAffinity:
10     preferredDuringSchedulingIgnoredDuringExecution:
11       - weight: 95
12         podAffinityTerm:
13           topologyKey: "kubernetes.io/hostname"
14           labelSelector:
15             matchLabels:
16               app: hlf-couchdb
17
```

master 0 0 0 Ln 1, Col 1 Spaces: 2 UTF-8 LF YAML





# Instalar Fabric Peer Helm Chart



Instalar CouchDB  
Helm Chart



Instalar Fabric Peer  
Helm Chart

```
helm install stable/hlf-peer -n peer${NUM} --  
namespace blockchain -f  
./helm_values/peer${NUM}_values.yaml
```



```
1 image:
2   tag: 1.2.0
3
4 persistence:
5   accessMode: ReadWriteOnce
6   size: 1Gi
7   storageClass: "managed-premium"
8
9 caAddress: ca.lf.aidtech-test.xyz
10 caUsername: peer1
11
12 peer:
13   hlfToolsVersion: 1.2.0
14   databaseType: CouchDB
15   couchdbInstance: cdb-peer1
16   mspID: AidTechMSP
17
18 secrets:
19   channel: hlf--channel
20   adminCert: hlf--org-admincert
21   adminKey: hlf--org-adminkey
22   caServerTls: ca--tls
23
24 affinity:
25   podAntiAffinity:
26     preferredDuringSchedulingIgnoredDuringExecution:
27     - weight: 95
28       podAffinityTerm:
29         topologyKey: "kubernetes.io/hostname"
```



# Definir Peer Identity



Instalar CouchDB  
Helm Chart



Instalar Fabric Peer  
Helm Chart



Definir Peer Identity

```
PEER_SECRET=$(kubectl get secret -n blockchain peer${NUM}-hlf-peer -o jsonpath="{.data.CA_PASSWORD}" | base64 --decode)
```

```
kubectl exec -n blockchain SCA_POD -- fabric-ca-client register --id.name peer${NUM} --id.secret SPEER_SECRET --id.type peer
```

```
PEER_POD=$(kubectl get pods -n blockchain -l "app=hlf-peer,release=peer${NUM}" -o jsonpath="{.items[0].metadata.name}")
```

```
kubectl logs -n blockchain SPEER_POD | grep 'Starting peer'
```



# Criar Channel



Instalar CouchDB  
Helm Chart



Instalar Fabric Peer  
Helm Chart



Definir Peer Identity



Criar Channel

```
kubectl exec -n blockchain SPEER_POD -- peer  
channel create -o ord1-hlf-  
ord.blockchain.svc.cluster.local:7050 -c mychannel -f  
/hl_config/channel/mychannel.tx
```



# Ingressar no Channel



Instalar CouchDB  
Helm Chart



Instalar Fabric Peer  
Helm Chart



Definir Peer Identity



Criar Channel



Ingressar Channel



```
kubectl exec -n blockchain $SPEER_POD -- peer channel fetch  
config /var/hyperledger/mychannel.block -c mychannel -o ord1-hlf-  
ord.blockchain.svc.cluster.local:7050
```

```
kubectl exec -n blockchain $SPEER_POD -- bash -c  
'CORE_PEER_MSPCONFIGPATH=$ADMIN_MSP_PATH  
peer channel join -b /var/hyperledger/mychannel.block'
```

```
kubectl exec -n blockchain $SPEER_POD -- peer channel list
```



THE  
DEVELOPER'S  
CONFERENCE

# Próximo Passos



2019...



Disponibilização de **Chart novo**  
para deploy de **Hyperledger  
Composer**



Colaboração na criação de  
**chart** para **Hyperledger  
Sawtooth**





## HYPERLEDGER

BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

*Community Stewardship and Technical, Legal, Marketing, Organizational Infrastructure*

### Frameworks



Permissionable smart contract machine (EVM)



Permissioned with channel support



Decentralized identity



Mobile application focus



Permissioned & permissionless support; EVM transaction family

### Tools



Blockchain framework benchmark platform



As-a-service deployment



Model and build blockchain networks



View and explore data on the blockchain



Ledger interoperability





# Perguntas?

**Cláudio Ramos**  
claudiovtramos@gmail.com





# THE DEVELOPER'S CONFERENCE