Apache ZooKeeper

Pedro Ferreira, Vinicius Cogo

Tópicos

1. Apache ZooKeeper
2. Modelo de Dados do ZooKeeper
3. Exemplos de Utilização
4. Instalação
5. API ZooKeeper
6. Ferramenta GUI: zooinspector

1. Apache ZooKeeper

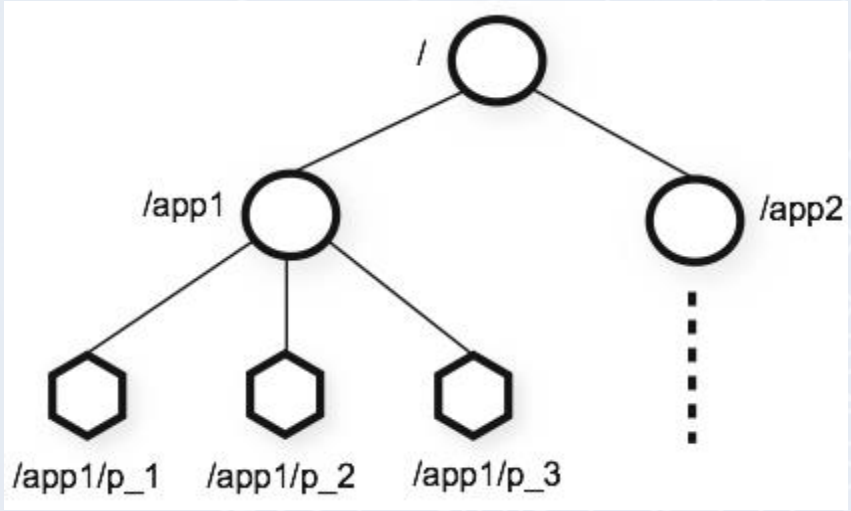
- **Serviço** para **Coordenação** de Sistemas Distribuídos
  - **Fiável**, **escalável** e de alta disponibilidade
  - Não serve para **guardar** dados, mas sim **metadados**
    - Ex: IDs, IPs, timestamps, versões, etc.
  - Usado por outros serviços, como:
    - Base dados replicadas, sistemas de sincronização, Blockchain, etc.

- **Multi-cliente**
  - pode ser usado por múltiplos clientes simultaneamente

- **Bibliotecas** e **APIs** em múltiplas linguagens
  - C, Java, Perl, **Python**
  - Command Line Interface (CLI) também disponível

2. Modelo de Dados do ZooKeeper

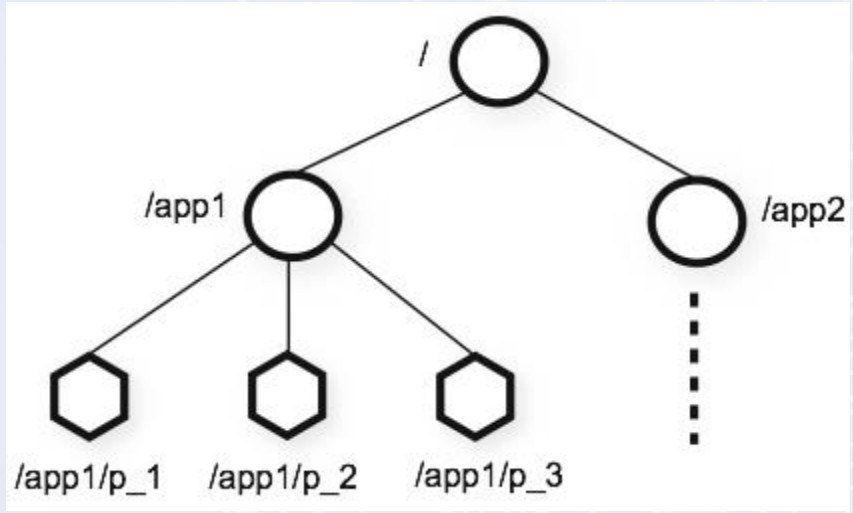
- Como é que os **metadados** escritos pelos clientes são **guardados** e organizados **no ZooKeeper**?



2. Modelo de Dados do ZooKeeper

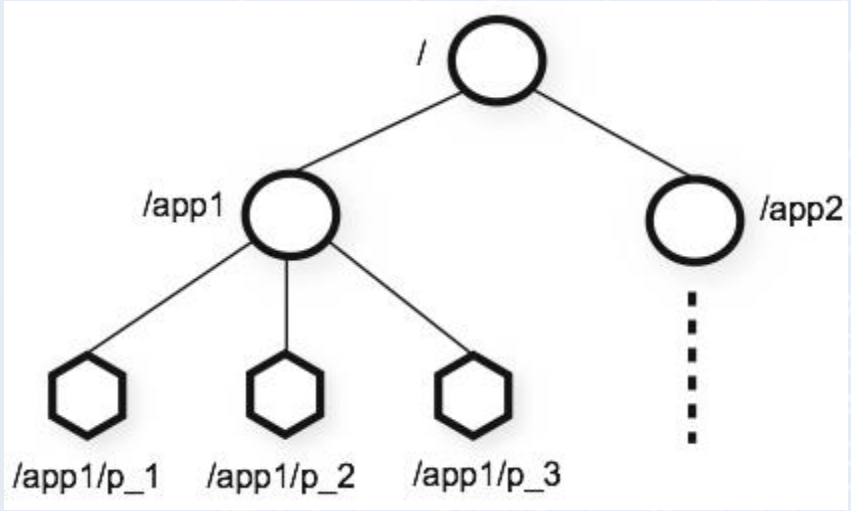
- **Znode**
  - Guarda **metadados** em **memória** (max 1 MB por znode)
  - Organização **hierárquica**, i.e. um nó pode ter vários "filhos"
  - **Nomes** usam notação estilo sistema de **ficheiros** UNIX. Ex: /app1/p_1/...

- **Tipos de Znode**
  - **Persistente:** desaparece somente após um delete
  - **Efémero:** não pode ter filhos e só existe enquanto a sessão cliente que o criou existir
  - **Sequencial:** é atribuído automaticamente um número de sequência ao nó
    - Ex: /app1/p_1
    - Tanto nós persistentes como efémeros podem ser simultaneamente sequenciais



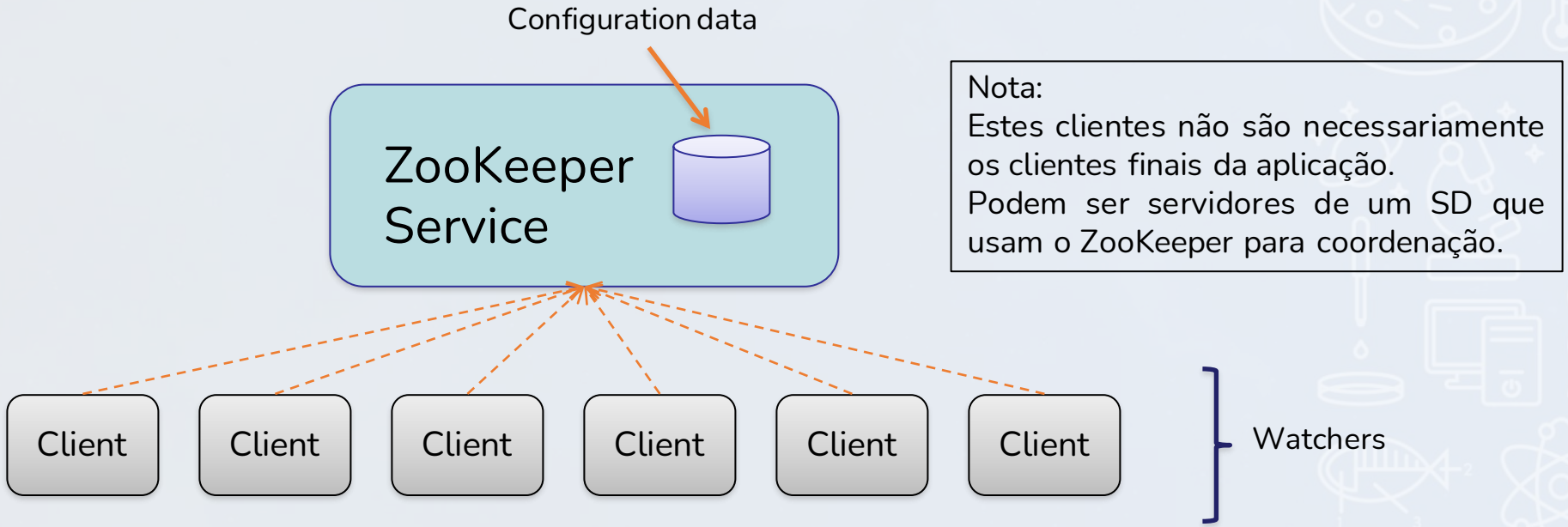
2. Modelo de Dados do ZooKeeper

- **Mecanismo de *Watch***
  - Permite **notificar clientes** quando há **alterações** num nó ou nos seus filhos
  - Ex: *get_children* `(... , "/app1", watch=my_func)`
    - Método invocado quando se quer ser notificado da adição ou remoção de filhos ao nó "/app1"



3. Exemplos de Utilização

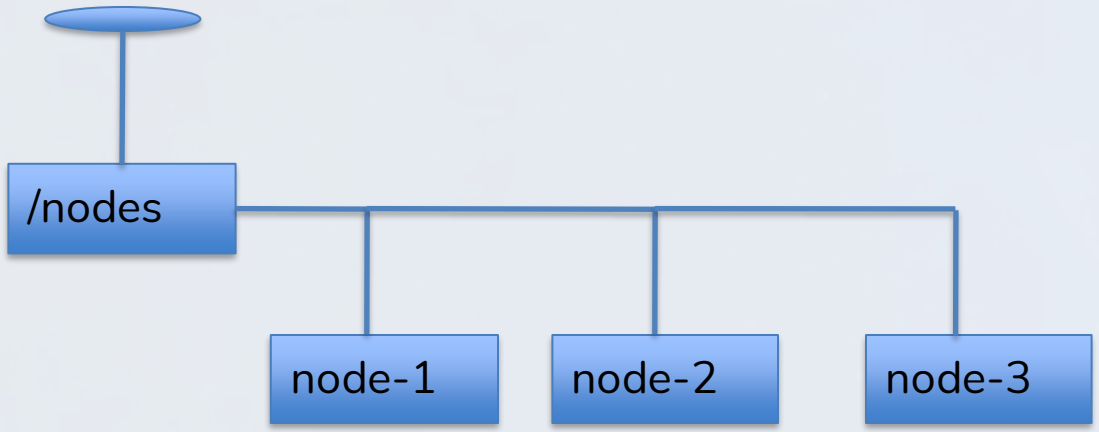
- **Gestão de Configuração Distribuída**
  - Um cliente cria um Znode **/conf** para guardar a configuração de um SD
  - Outros clientes fazem *watch* do Znode **/conf**
  - Quando há um update ao **/conf**, os clientes são notificados e verificam novo estado



Nota:
Estes clientes não são necessariamente os clientes finais da aplicação.
Podem ser servidores de um SD que usam o ZooKeeper para coordenação.

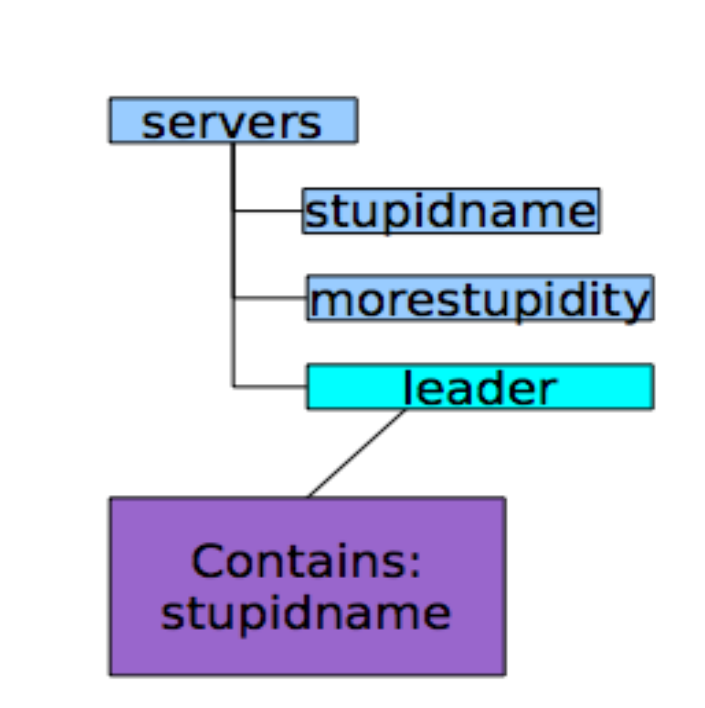
3. Exemplos de Utilização

- **Gestão de membros de grupo**
  - Cliente que gere o grupo cria um Znode **/nodes**
  - Cada um dos restantes clientes criam um Znode efémero filho de **/nodes**
  - Para receber atualizações sobre o grupo,
    os clientes fazem *watch* a **/nodes** e vêem os filhos



3. Exemplos de Utilização

- **Eleição de líder**
  - Verificar quem é líder
    - get (… , "/servers/leader", …)
  - Se houver um líder, seguir o líder
  - Se não houver líder, tentar registar como líder
    - create (… , "/servers/leader", ZOO_EPHEMERAL, …)
  - Se não conseguir seguir nem criar, voltar ao primeiro passo



4. Instalação do ZooKeeper na máquina virtual do DI

- Descarregar a **máquina virtual** equivalente aos labs **do DI** nas páginas da admin do DI (admin.di.fc.ul.pt)

- Iniciar a máquina virtual

- Num **terminal**:

```
$ sudo apt-get update
$ sudo apt-get install zookeeperd
$ pip3 install kazoo
$ exit
```

5. API ZooKeeper

- Cada **método** tem **duas versões**: **síncrono** e **assíncrono**
  - Métodos **assíncronos não bloqueiam**, eles definem antes uma função que é invocada quando chegar uma resposta do servidor
- Métodos síncronos/assíncronos
  - ***create***
  - ***delete***
  - ***exists***
  - ***get***
  - ***set***
  - ***get_children***
- Todos os métodos estão definidos no módulo **kazoo.client**

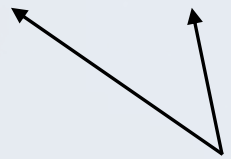
Os métodos assíncronos contêm o *prefixo* ***a*** *no nome da função (e.g., acreate)*

5. API ZooKeeper em Python – Kazoo Client

- Importar:
  - `from **kazoo.client** import **KazooClient**`

- Ver **métodos** disponíveis:
  - `print(dir(KazooClient))`

- Ver a **documentação** dos métodos:
  - `print(KazooClient.<método>.__doc__)`



Dois *underscores*

- Exemplos:

```
print(KazooClient.__init__.__doc__)
print(KazooClient.create.__doc__)
```

5. API ZooKeeper em Python – Kazoo Client

- `from kazoo.client import KazooClient`
- `zh = KazooClient(`**`hosts='127.0.0.1:2181', …`**`)`
  - Função que **inicia** uma **sessão** com o zookeeper e **devolve** um ***handler*** para essa sessão
- Parâmetros:
  - *hosts*: lista de pares host:port separados por vírgulas, cada um referente a um servidor zookeeper
  - Outros parâmetros opcionais: fazer `print(KazooClient.<método>.__doc__)` num interpretador Python (após `from kazoo.client import KazooClient`)

5. API ZooKeeper em Python – Overview

- Template geral para utilizar o Kazoo

```
from kazoo.client import KazooClient

# Criar um ZooKeeper handler
zh = KazooClient(hosts='127.0.0.1:2181', …)
zh.start()

# restante do programa

zh.stop()
zh.close()
```

5. API ZooKeeper em Python – create

- `zh.create(path, value='', acl=None, ephemeral=False, sequence=False, makepath=False)`
  - Cria um novo nó no caminho indicado por **path**, com valor **value**

- Parâmetros:
  - **path** – Path of node.
  - **value** – Initial bytes value of node.
  - acl – ACL list.
  - **ephemeral** – Boolean indicating whether node is ephemeral (tied to this session).
  - **sequence** – Boolean indicating whether path is suffixed with a unique index.
  - makepath – Whether the path should be created if it doesn't exist.

5. API ZooKeeper em Python – delete

- `zh.delete(path, version=-1, recursive=False)`
  - Apaga o nó indicado por **path**

- Parâmetros:
  - **path** – Path of node to delete.
  - version – Version of node to delete, or -1 for any.
  - recursive (bool) – Recursively delete node and all its children, defaults to False.

5. API ZooKeeper em Python – Outros métodos

- `zh.get_children(path, watch=None, include_data=False)`
  - Obtém a lista de nós filhos do nó indicado em **path**

- `zh.get(path, watch=None)`
  - Obtém os dados guardados no nó indicado por **path**

- `zh.set(path, value, version=-1)`
  - Escreve no nó indicado por **path** os dados guardados em **value**

5. API ZooKeeper em Python – Outros métodos

- `zh.exists(path, watch=None)`
  - Verifica se o nó indicado por **path** existe,
    retornando uma estrutura chamada ZnodeStat (com metadados sobre o nó)

- `zh.ensure_path(path, acl=None)`
  - Cria recursivamente um **path** se este não existe

5. API ZooKeeper em Python – Watchers

- Funções **watcher** para **monitorizar** eventos em **nós** e filhos
  - `get`, `exists` e `get_children`

- Tipos de **eventos**:
  - CREATED (um nó foi criado)
  - DELETED (um nó foi removido)
  - CHANGED (o value de um nó foi modificado)
  - CHILD (um nó filho foi criado ou removido)
  - NONE (o estado da conexão alterou-se)

- Tipos de **estados**:
  - CONNECTED (a conexão está ativa e válida)
  - SUSPENDED (a conexão foi perdida mas ainda pode ser recuperada)
  - LOST (a conexão está confirmada como perdida)

5. API ZooKeeper em Python – Watchers

```
### Forma 1:
def my_func(event):
    # P.ex., verifica o que mudou nos nós filhos de /app

# Call my_func when the children change
children = zh.get_children("/app", watch=my_func)

### Forma 2:
@zh.ChildrenWatch("/app")
def watch_children(children):
    print("Children are now: %s" % children)
# Above function called immediately, and from then on

@zh.DataWatch("/app")
def watch_node(data, stat):
    print("Version: %s, data: %s" % (stat.version, data.decode()))
```

5. API ZooKeeper em Python – Transactions

- Permite **executar** uma sequência de **operações atomicamente**
- **Completa todas operações**, **ou nenhuma** surte efeito

```
transaction = zh.transaction()

transaction.check('/node/a', version=3)

transaction.create('/node/b', b"a value")

results = transaction.commit()
```

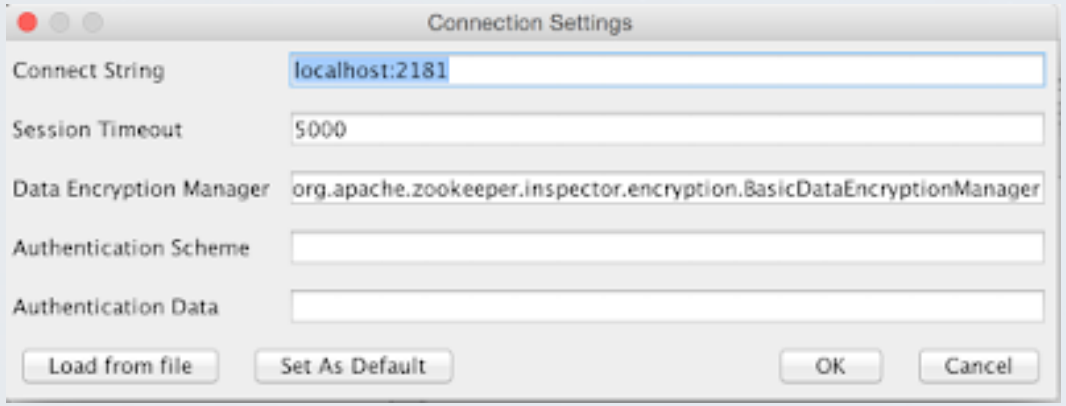
5. API ZooKeeper em Python – Recipes

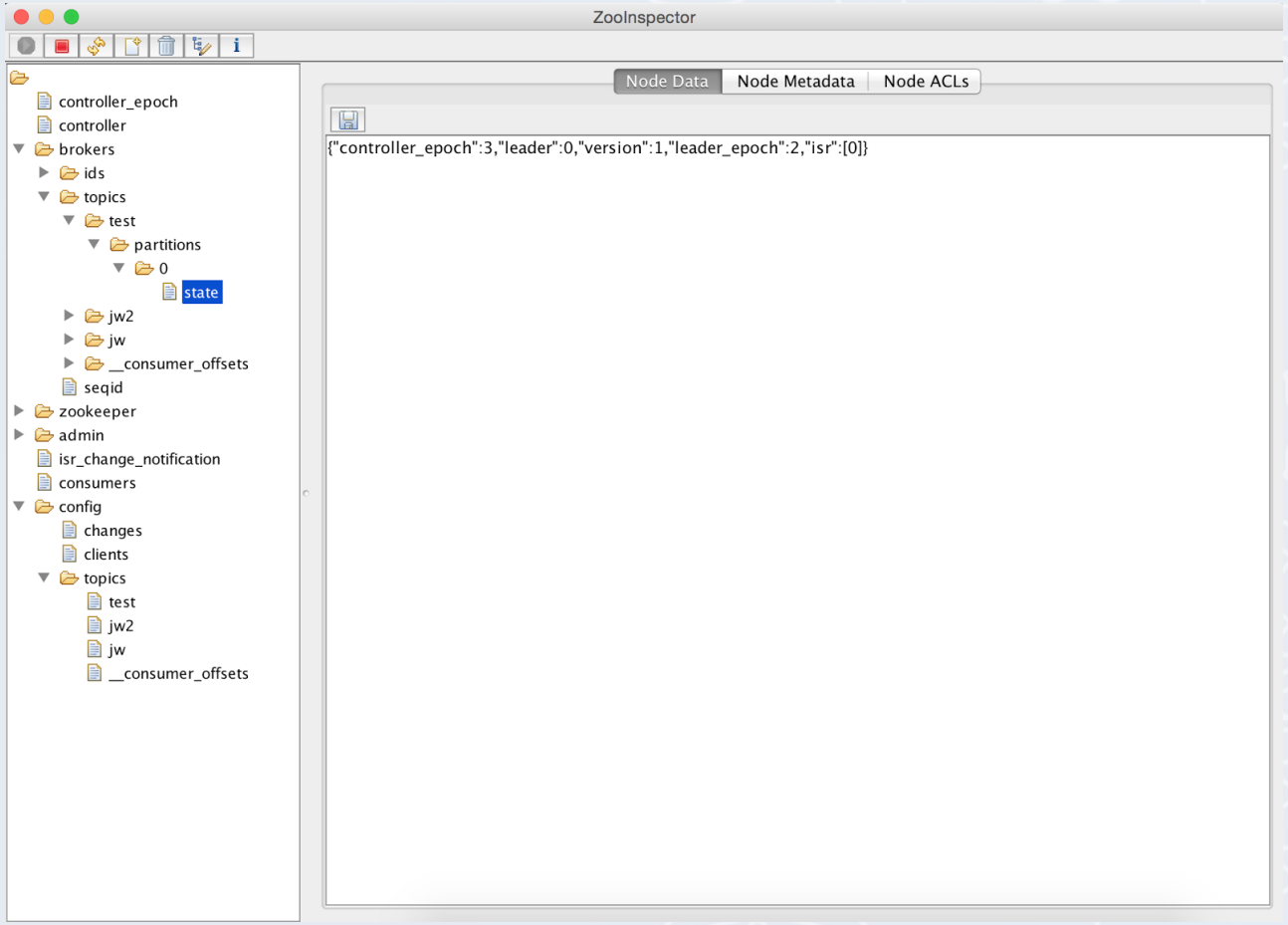
- `kazoo.recipe`
  - **Receitas** com diversas implementações de **sincronização prontas**
- Exemplos:
  - **Barrier: barreira** para bloquear um conjunto de **nós até** que **uma condição** seja atingida
  - **Counter: contador partilhado** (int ou float) onde **alterações** ocorrem **atomicamente**
  - **Election:** algoritmo de **eleição** de **líder**
  - **Lease: lease temporário** exclusivo não-bloqueante que pode ser renovado
  - **Lock: locks não-reentrantes**, **read** locks and **write** locks
  - **Partitioner: particiona** um conjunto de **nós** entre sub grupos
  - **Party: gestão** de **grupos** de processos participantes (**join**, **leave**, len, iter)
  - **Queue: filas** tradicionais
  - **Watchers: funções** de **monitorização** de **nós** e eventos nestes

6. GUI para o ZooKeeper - `zooinspector`

- **Ferramenta** com interface gráfica útil para:
  - **Criar** nós manualmente
  - **Remover** nós
  - **Inspecionar** nós

- `$ zooinspector`
- Deve-se primeiro conectar o programa à instância do ZooKeeper





Bibliografia

- Mais informações:
  - kazoo 2.6.0 documentation
    - https://kazoo.readthedocs.io/en/latest/
  - ZooKeeper Programmer's Guide
    - https://zookeeper.apache.org/doc/r3.3.6/zookeeperProgrammers.html#ZooKeeper+C+client+API
  - ZooKeeper Book
    - https://www.oreilly.com/library/view/zookeeper/9781449361297/
  - Apache ZooKeeper Essentials Book
    - https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784391324