Building a Fault-Tolerant Distributed System with zookeepertcl

Tcl Conference 2018

Garrett McGrath

/whois

/whois

- Developer at FlightAware
    - Work on Hyperfeed

/whois

- Developer at FlightAware
  - Work on Hyperfeed
- Current focus on distribution and reliability
  - Talk based on this work

System Definition

System Definition

- Multiple **components** (process)
  - All need to run concurrently
  - Too many to run on a single machine

System Definition

- Multiple **components** (process)
  - All need to run concurrently
  - Too many to run on a single machine
- Spread across multiple machines (nodes)
  - Egalitarian system
    - In terms of compute resources

System Definition

- Multiple **components** (process)
    - All need to run concurrently
    - Too many to run on a single machine
- Spread across multiple machines (nodes)
    - Egalitarian system
        - In terms of compute resources
- Each **component**
    - Runs on one machine at a time
    - Allow a node to run multiple components

Faults and Failures

Faults and Failures

- Expect temporary and permanent failures
    - Of components
    - And nodes

Faults and Failures

- Expect temporary and permanent failures
  - Of components
  - And nodes
- Want to tolerate
  - Crash failures
  - Omission failures

Faults and Failures

- Expect temporary and permanent failures
  - Of components
  - And nodes
- Want to tolerate
  - Crash failures
  - Omission failures
- Consistency-Availability-Partition
  - Address A and P

Recovery and Failover

Recovery and Failover

- Since failure expected, when it happens

Recovery and Failover

- Since failure expected, when it happens
  - To a component
    - Want it to run on another node

Recovery and Failover

- Since failure expected, when it happens
  - To a component
    - Want it to run on another node
  - To a node
    - Want its components to run on other nodes

Recovery and Failover

- Since failure expected, when it happens
    - To a component
        - Want it to run on another node
    - To a node
        - Want its components to run on other nodes
- Want a system that
    - Supports automated failover
        - For common failure conditions

Scope and Limitations

Scope and Limitations

- Cannot protect against all failures

Scope and Limitations

- Cannot protect against all failures
- Consistency / integrity faults unaddressed

Scope and Limitations

- Cannot protect against all failures
- Consistency / integrity faults unaddressed
- Byzantine Failure not touched
    - Arbitrary and/or malicious responses
        - Possibly from unintentional bugs
        - Or, collusion among nodes to deceive

Scope and Limitations

- Cannot protect against all failures
- Consistency / integrity faults unaddressed
- Byzantine Failure not touched
    - Arbitrary and/or malicious responses
        - Possibly from unintentional bugs
        - Or, collusion among nodes to deceive
- Partial addressing of network partitions

Implementation

Implementation

- Fault tolerant distributed system
  - With Tcl and Zookeeper

Implementation

- Fault tolerant distributed system
  - With Tcl and Zookeeper
- Based on leader election recipe
  - Use term in a peculiar way

Implementation

- Fault tolerant distributed system
  - With Tcl and Zookeeper
- Based on leader election recipe
  - Use term in a peculiar way
- Each component will have a leader

Implementation

- Fault tolerant distributed system
    - With Tcl and Zookeeper
- Based on leader election recipe
    - Use term in a peculiar way
- Each component will have a leader
    - Who is running the component

Implementation

- Fault tolerant distributed system
  - With Tcl and Zookeeper
- Based on leader election recipe
  - Use term in a peculiar way
- Each component will have a leader
  - Who is running the component
- With other nodes ready to step in

Per Node Implemention

Per Node Implemention

- Each node runs a supervisor

Per Node Implemention

- Each node runs a supervisor
    - Communicates with Zookeeper

Per Node Implemention

- Each node runs a supervisor
  - Communicates with Zookeeper
  - Elects components
    - Starts them if win election
    - Or if current leader fails

Per Node Implemention

- Each node runs a supervisor
  - Communicates with Zookeeper
  - Elects components
    - Starts them if win election
    - Or if current leader fails
  - Monitors components, e.g., **SIGCHLD**

Per Node Implemention

- Each node runs a supervisor
    - Communicates with Zookeeper
    - Elects components
        - Starts them if win election
        - Or if current leader fails
    - Monitors components, e.g., **SIGCHLD**
- Supervisor Knows
    - How to start and stop each component
    - Other nodes in the system

ZOO

Zookeeper

Zookeeper

- Distributed coordination service

Zookeeper

- Distributed coordination service
- Developed at Yahoo
  - Maintained by the ASF

Zookeeper

- Distributed coordination service
- Developed at Yahoo
    - Maintained by the ASF
- Written in Java

Zookeeper

- Distributed coordination service
- Developed at Yahoo
    - Maintained by the ASF
- Written in Java
- Runs
    - Standalone (dev / testing)

Zookeeper

- Distributed coordination service
- Developed at Yahoo
  - Maintained by the ASF
- Written in Java
- Runs
  - Standalone (dev / testing)
  - Replicated
    - Handle **k** failures
    - With **2k + 1** servers

Coordination

Coordination

- Notoriously difficult to get right
    - Deadlocks
    - Race conditions

Coordination

- Notoriously difficult to get right
  - Deadlocks
  - Race conditions
- Examples

Coordination

- Notoriously difficult to get right
  - Deadlocks
  - Race conditions
- Examples
  - Barriers

Coordination

- Notoriously difficult to get right
  - Deadlocks
  - Race conditions
- Examples
  - Barriers
  - Queues

Coordination

- Notoriously difficult to get right
  - Deadlocks
  - Race conditions
- Examples
  - Barriers
  - Queues
  - Locks (read or write)

Coordination

- Notoriously difficult to get right
  - Deadlocks
  - Race conditions
- Examples
  - Barriers
  - Queues
  - Locks (read or write)
  - Two-phase commit (atomic transactions)

Coordination

- Notoriously difficult to get right
    - Deadlocks
    - Race conditions
- Examples
    - Barriers
    - Queues
    - Locks (read or write)
    - Two-phase commit (atomic transactions)
    - Leader election

API

API

- Does **not** come with pre-baked primitives based on coordination task

API

- Does **not** come with pre-baked primitives based on coordination task
- Exposes a simple API instead
  - More flexible
  - Use it to implement coordination tasks
  - Provides consistency and availability guarantees

API, Cont.

API, Cont.

- Based on a file-system like abstraction

API, Cont.

- Based on a file-system like abstraction
  - ***znode***
    - Combination of file and directory

API, Cont.

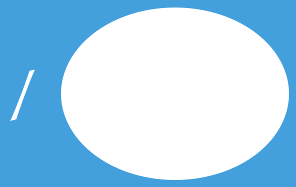
- Based on a file-system like abstraction
  - ***znode***
    - Combination of file and directory
  - Provides hierarchical namespace
    - Enables process communication

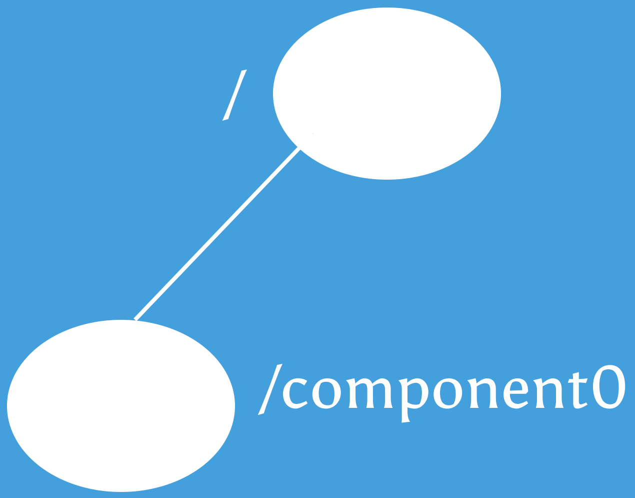
API, Cont.

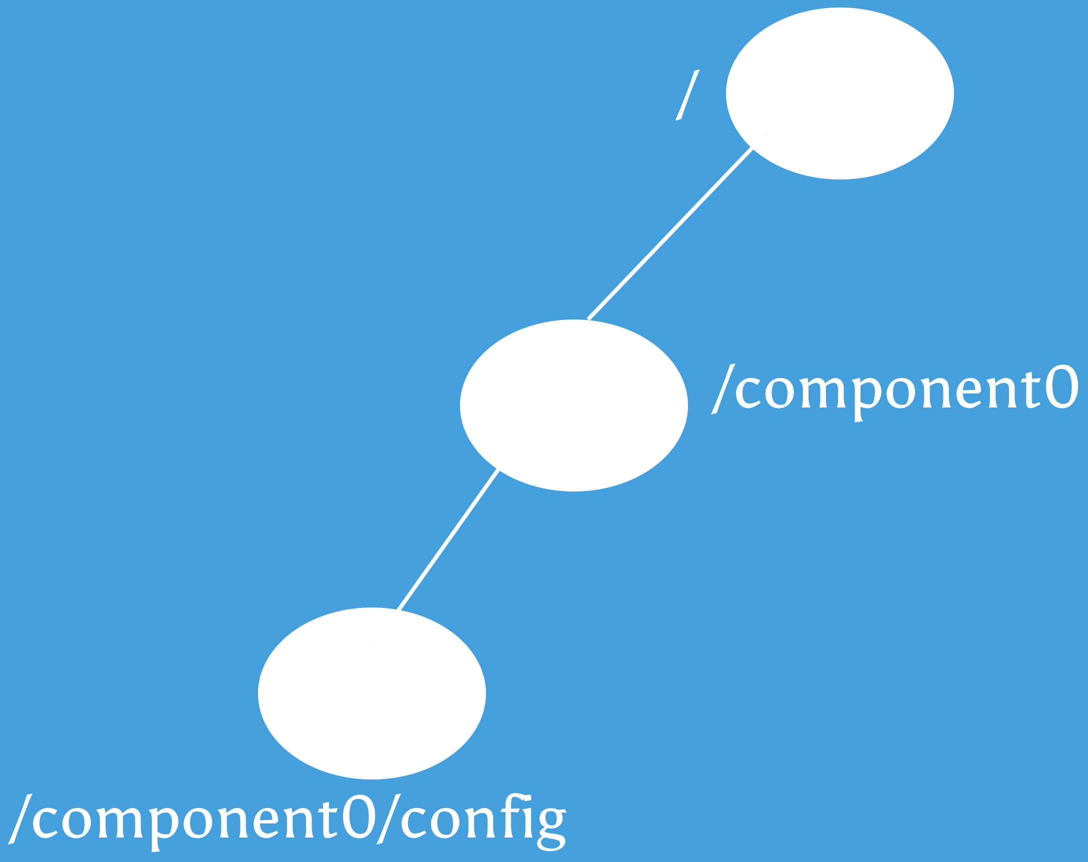
- Based on a file-system like abstraction
  - ***znode***
    - Combination of file and directory
  - Provides hierarchical namespace
    - Enables process communication
- ***znodes*** contain
  - Data (small amount, typically 1MB max)

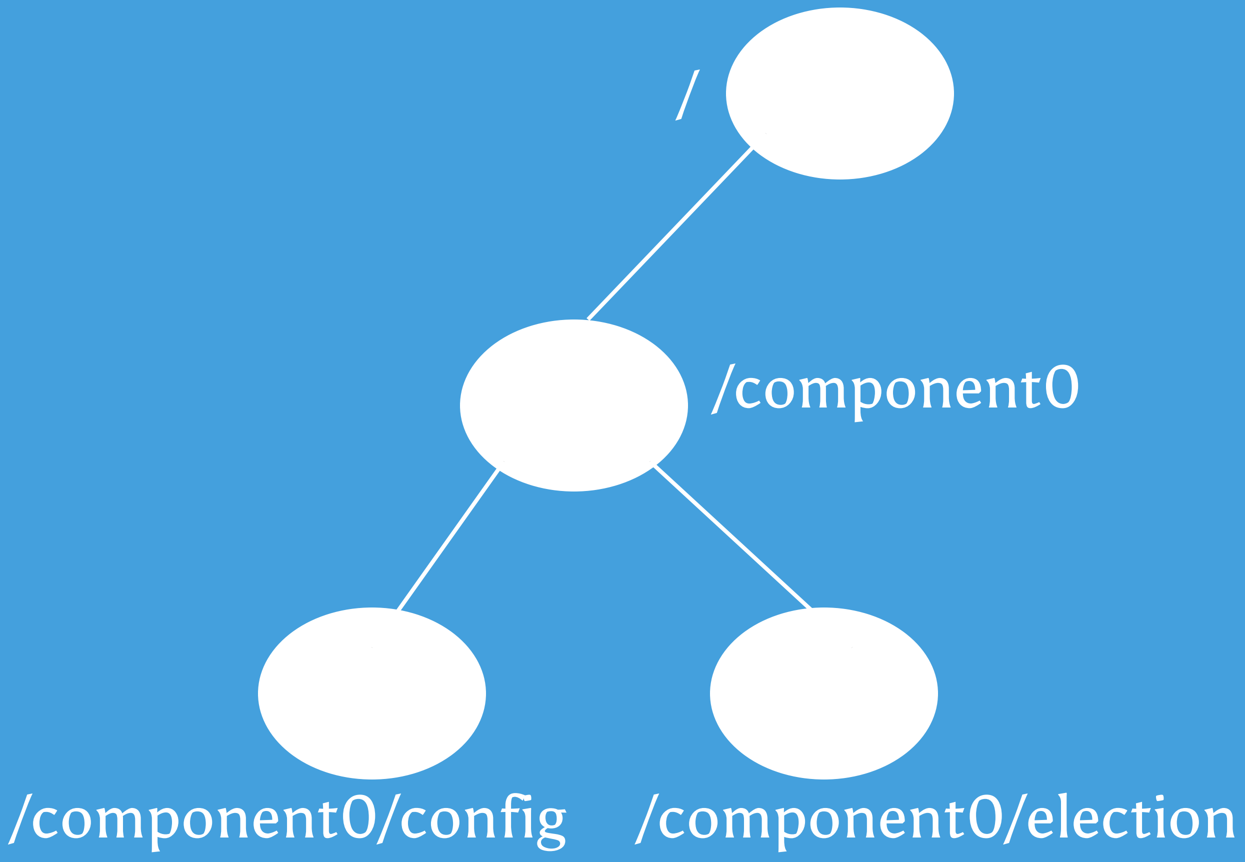
API, Cont.

- Based on a file-system like abstraction
  - *znode*
    - Combination of file and directory
  - Provides hierarchical namespace
    - Enables process communication
- *znodes* contain
  - Data (small amount, typically 1MB max)
  - Metadata (ACLs, ctime, mtime, atime)




/
/component0


/
/component0
/component0/config


/
/component0
/component0/config
/component0/election

API Operations

What Can We Do

API Operations

What Can We Do

- Create new ***znodes***

API Operations

What Can We Do

- Create new ***znodes***
    - Durable or ephemeral

API Operations

What Can We Do

- Create new ***znodes***
  - Durable or ephemeral
  - Sequential

API Operations

What Can We Do

- Create new ***znodes***
  - Durable or ephemeral
  - Sequential
- Delete existing ***znodes***

API Operations

What Can We Do

- Create new ***znodes***
  - Durable or ephemeral
  - Sequential
- Delete existing ***znodes***
- Query ***znodes***

API Operations

What Can We Do

- Create new ***znodes***
    - Durable or ephemeral
    - Sequential
- Delete existing ***znodes***
- Query ***znodes***
    - Exist?

API Operations

What Can We Do

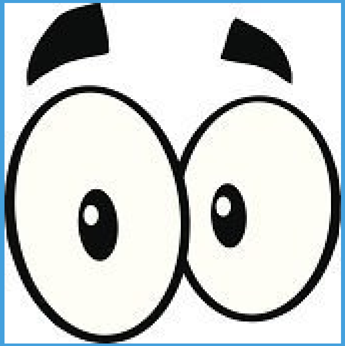
- Create new ***znodes***
  - Durable or ephemeral
  - Sequential
- Delete existing ***znodes***
- Query ***znodes***
  - Exist?
  - Children?

API Operations

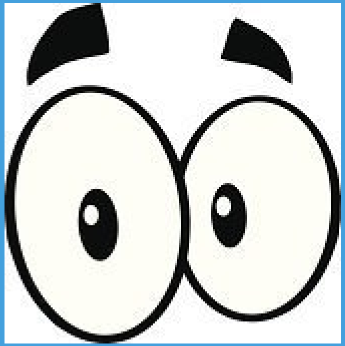
What Can We Do

- Create new ***znodes***
  - Durable or ephemeral
  - Sequential
- Delete existing ***znodes***
- Query ***znodes***
  - Exist?
  - Children?
- Get / modify ***znode*** {meta,}data

Watch Callbacks

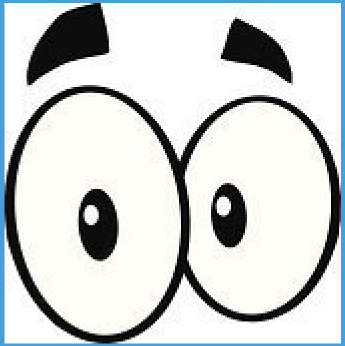


Watch Callbacks



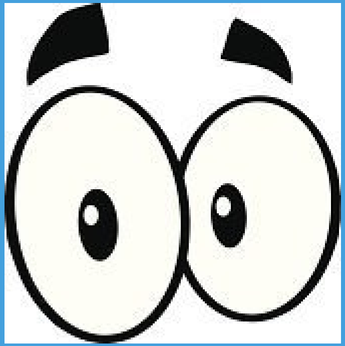
- Several operations support a ***watch*** callback
  - **One-time** callback invoked when the **znode** changes

Watch Callbacks



- Several operations support a ***watch*** callback
  - **One-time** callback invoked when the **znode** changes
- A ***get*** or ***exists*** watch
  - Called when the ***znode*** modified

Watch Callbacks



- Several operations support a ***watch*** callback
  - **One-time** callback invoked when the **znode** changes
- A ***get*** or ***exists*** watch
  - Called when the ***znode*** modified
- A ***children*** watch
  - Called when anything happens to the ***znode***'s children

zookeepertcl

zookeepertcl

- Open-source library
  - **github.com/flightaware/zookeepertcl**

zookeepertcl

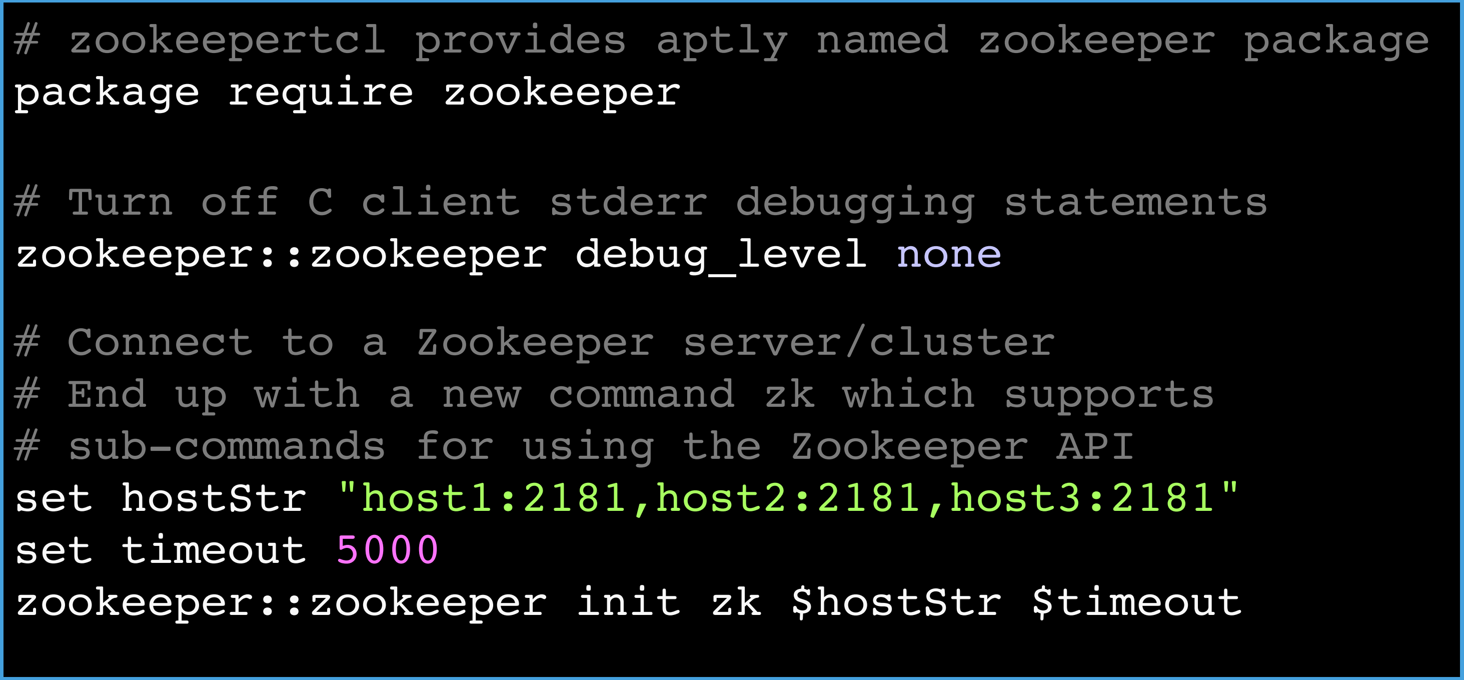
- Open-source library
  - **github.com/flightaware/zookeepertcl**
- Wraps the official C client
  - Supports the latest stable Zookeeper version
    - r3.4.13

zookeepertcl

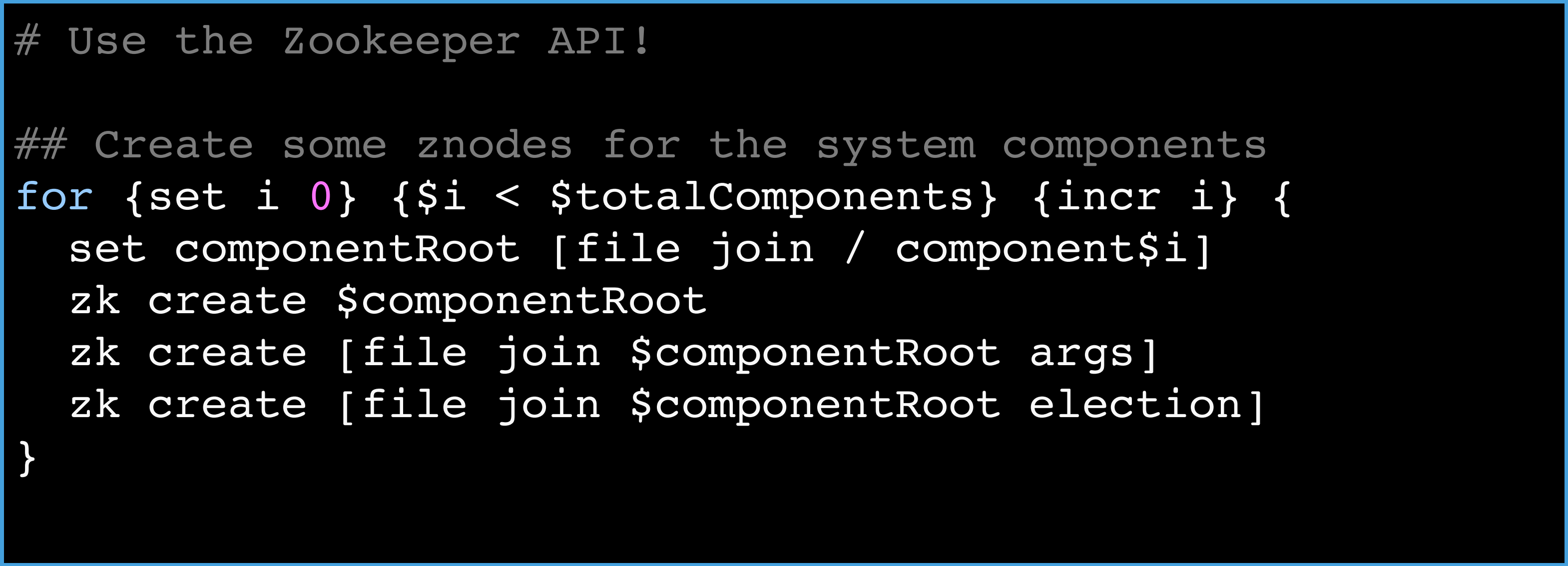
- Open-source library
  - **github.com/flightaware/zookeepertcl**
- Wraps the official C client
  - Supports the latest stable Zookeeper version
    - r3.4.13
- Each API operation supports two styles
  - Synchronous
  - Asynchronous

```
# zookeepertcl provides aptly named zookeeper package
package require zookeeper
```

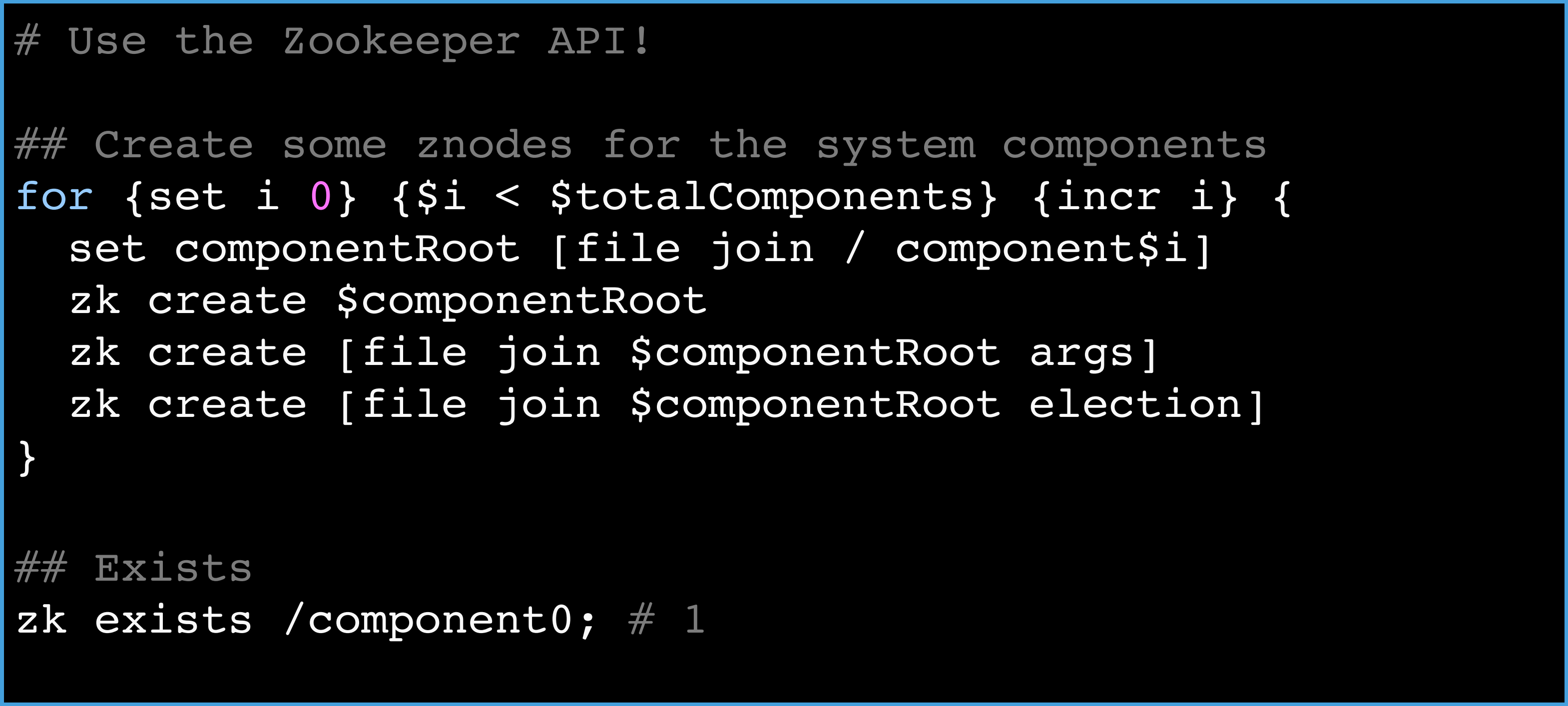
```
# zookeepertcl provides aptly named zookeeper package
package require zookeeper

# Turn off C client stderr debugging statements
zookeeper::zookeeper debug_level none
```



```
# zookeepertcl provides aptly named zookeeper package
package require zookeeper

# Turn off C client stderr debugging statements
zookeeper::zookeeper debug_level none

# Connect to a Zookeeper server/cluster
# End up with a new command zk which supports
# sub-commands for using the Zookeeper API
set hostStr "host1:2181,host2:2181,host3:2181"
set timeout 5000
zookeeper::zookeeper init zk $hostStr $timeout
```



```
# Use the Zookeeper API!

## Create some znodes for the system components
for {set i 0} {$i < $totalComponents} {incr i} {
  set componentRoot [file join / component$i]
  zk create $componentRoot
  zk create [file join $componentRoot args]
  zk create [file join $componentRoot election]
}
```

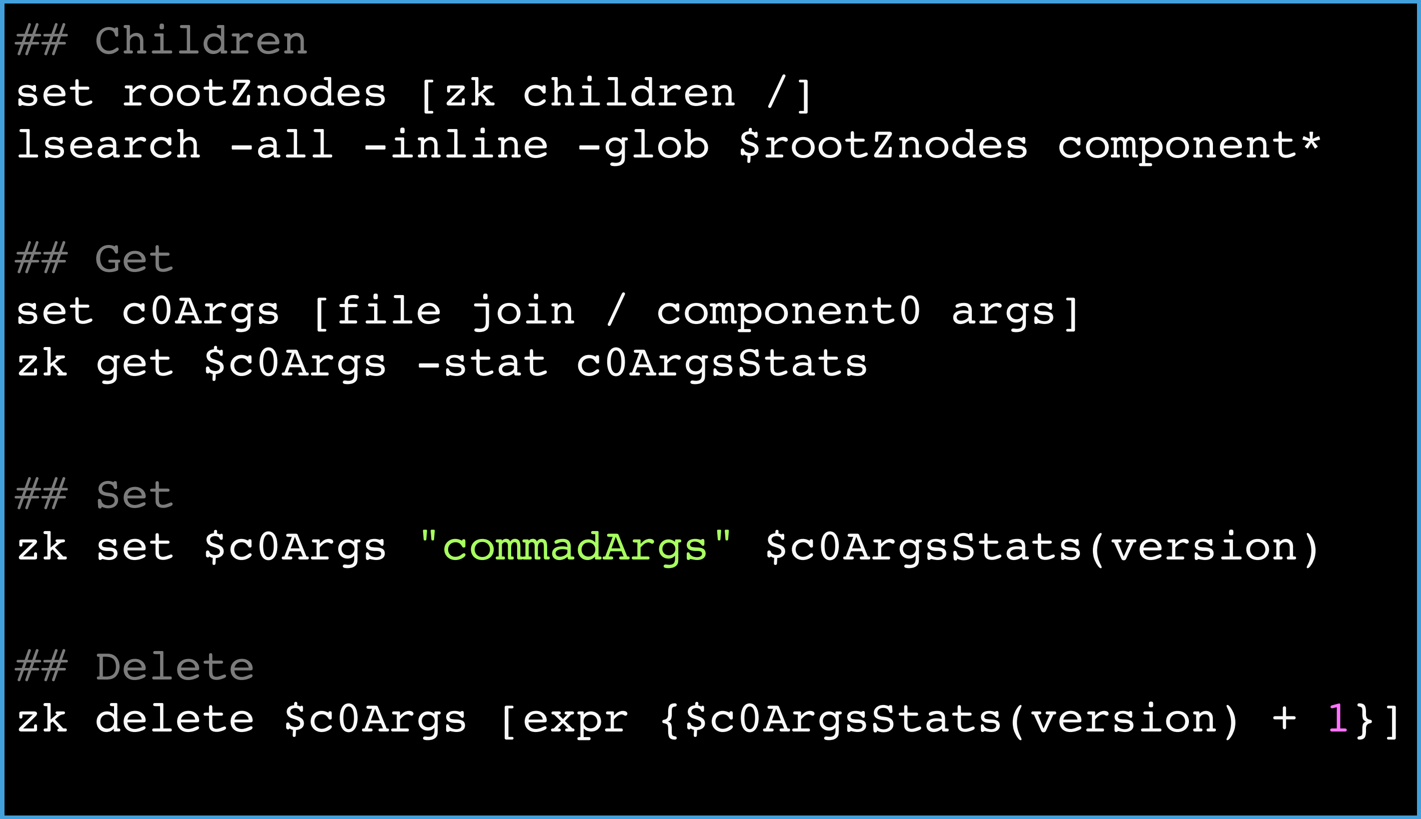


```
# Use the Zookeeper API!

## Create some znodes for the system components
for {set i 0} {$i < $totalComponents} {incr i} {
  set componentRoot [file join / component$i]
  zk create $componentRoot
  zk create [file join $componentRoot args]
  zk create [file join $componentRoot election]
}

## Exists
zk exists /component0; # 1
```

```
## Children
set rootZnodes [zk children /]
lsearch -all -inline -glob $rootZnodes component*
```

```
## Children
set rootZnodes [zk children /]
lsearch -all -inline -glob $rootZnodes component*

## Get
set c0Args [file join / component0 args]
zk get $c0Args -stat c0ArgsStats
```

```
## Children
set rootZnodes [zk children /]
lsearch -all -inline -glob $rootZnodes component*

## Get
set c0Args [file join / component0 args]
zk get $c0Args -stat c0ArgsStats


## Set
zk set $c0Args "commadArgs" $c0ArgsStats(version)
```



```
## Children
set rootZnodes [zk children /]
lsearch -all -inline -glob $rootZnodes component*

## Get
set c0Args [file join / component0 args]
zk get $c0Args -stat c0ArgsStats


## Set
zk set $c0Args "commadArgs" $c0ArgsStats(version)

## Delete
zk delete $c0Args [expr {$c0ArgsStats(version) + 1}]
```

Leader Election Recipe

Step 1

Create ***znode*** z with path "ELECTION/n_" with both SEQUENCE and EPHEMERAL flags;

```
# assume that $electionRoot already exists
set electionRoot [file join / component0 election]
```

```
# assume that $selectionRoot already exists
set electionRoot [file join / component0 election]


set myVote [file join $selectionRoot "n_"]
```

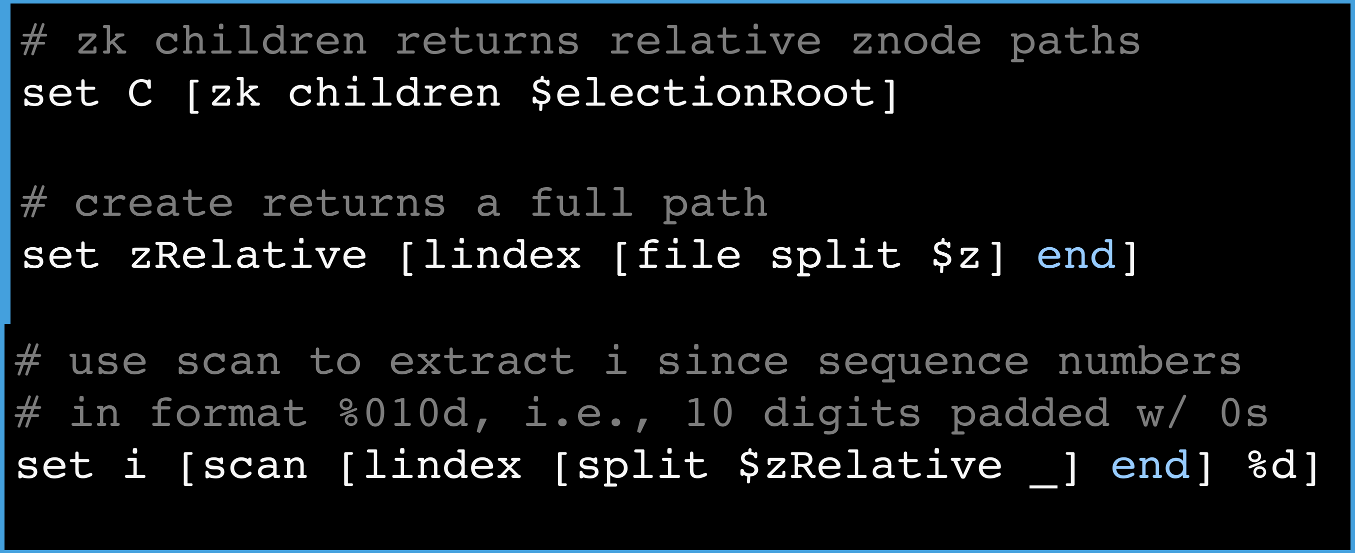
```
# assume that $selectionRoot already exists
set electionRoot [file join / component0 election]

set myVote [file join $selectionRoot "n_"]

set z [zk create $myVote -ephemeral -sequence]
```

Step 2

Let C be the children of "ELECTION", and i be the sequence number of z;

```
# zk children returns relative znode paths
set C [zk children $electionRoot]
```

```
# zk children returns relative znode paths
set C [zk children $selectionRoot]

# create returns a full path
set zRelative [lindex [file split $z] end]
```

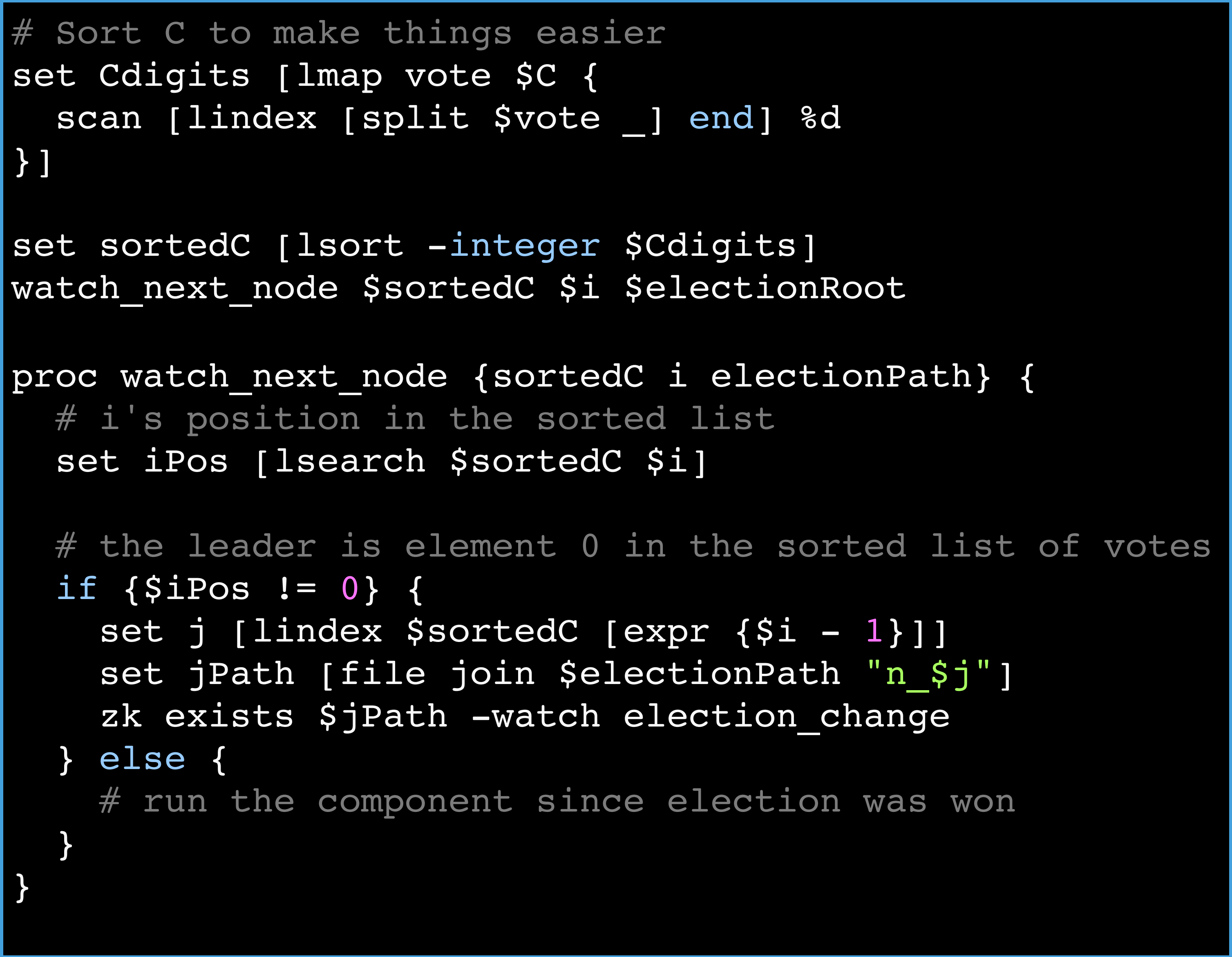


```
# zk children returns relative znode paths
set C [zk children $selectionRoot]

# create returns a full path
set zRelative [lindex [file split $z] end]

# use scan to extract i since sequence numbers
# in format %010d, i.e., 10 digits padded w/ 0s
set i [scan [lindex [split $zRelative _] end] %d]
```

Step 3

Watch for changes on "ELECTION/n_j", where j is the largest sequence number such that j < i and n_j is a znode in C;

```
# Sort C to make things easier
set Cdigits [lmap vote $C {
  scan [lindex [split $vote _] end] %d
}]

set sortedC [lsort -integer $Cdigits]
watch_next_node $sortedC $i $electionRoot
```



```
# Sort C to make things easier
set Cdigits [lmap vote $C {
  scan [lindex [split $vote _] end] %d
}]

set sortedC [lsort -integer $Cdigits]
watch_next_node $sortedC $i $electionRoot

proc watch_next_node {sortedC i electionPath} {
  # i's position in the sorted list
  set iPos [lsearch $sortedC $i]

  # the leader is element 0 in the sorted list of votes
  if {$iPos != 0} {
    set j [lindex $sortedC [expr {$i - 1}]]
    set jPath [file join $electionPath "n_$j"]
    zk exists $jPath -watch election_change
  } else {
    # run the component since election was won
  }
}
```

Implementation Decisions

"All my decisions are well thought out."

Abdication

Giving up Leadership

Abdication

Giving up Leadership

- Timing of elections can result in massive asymmetries
  - Do not want one node to crowd out others

Abdication
Giving up Leadership

- Timing of elections can result in massive asymmetries
  - Do not want one node to crowd out others
- Implement a policy of abdication
  - Based on, e.g., ***fair distribution***
  - Delay after win election
  - If leader, set *children* watch

Restart Loops

Limiting Abdication

Restart Loops

Limiting Abdication

- Intermittent failures and abdication
    - Single component could get passed around

Restart Loops

Limiting Abdication

- Intermittent failures and abdication
    - Single component could get passed around
- Need to avoid this potential instability
    - Matter of retaining sufficient state
        - Can do locally
        - Or in ***znodes***

Intentional Stops

Retaining Leadership

Intentional Stops

Retaining Leadership

- Often desirable to restart or stop component
    - Without giving up current leadership

Intentional Stops

Retaining Leadership

- Often desirable to restart or stop component
    - Without giving up current leadership
- Main justification for using a supervisor

Intentional Stops
Retaining Leadership

- Often desirable to restart or stop component
  - Without giving up current leadership
- Main justification for using a supervisor
- Many potential methods of addressing this
  - One is to use special ***znodes*** to pass commands

Config Changes

Targeted Restarts

Config Changes

Targeted Restarts

- Watch callbacks on ***/config*** portion of component's ***znode*** hierarchy

Config Changes

Targeted Restarts

- Watch callbacks on ***/config*** portion of component's ***znode*** hierarchy
- Callbacks can pile up
    - E.g., delete one argument and add another

Config Changes

Targeted Restarts

- Watch callbacks on ***/config*** portion of component's ***znode*** hierarchy
- Callbacks can pile up
    - E.g., delete one argument and add another
- Need a way of performing targeted restarts

Connection Loss

Zookeeper Session States

Connection Loss

Zookeeper Session States

- Need a policy about what to do when connection to Zookeeper is lost
  - Watch callbacks do not persist

Connection Loss
Zookeeper Session States

- Need a policy about what to do when connection to Zookeeper is lost
  - Watch callbacks do not persist
- Zookeeper connections
  - Called a session
  - Represented as a state machine
  - Distinguishes connection lost or interrupted