CouchbaseConnect

Go For It

Building Advanced Systems with Go and Couchbase Server

7 October 2014

Marty Schoch

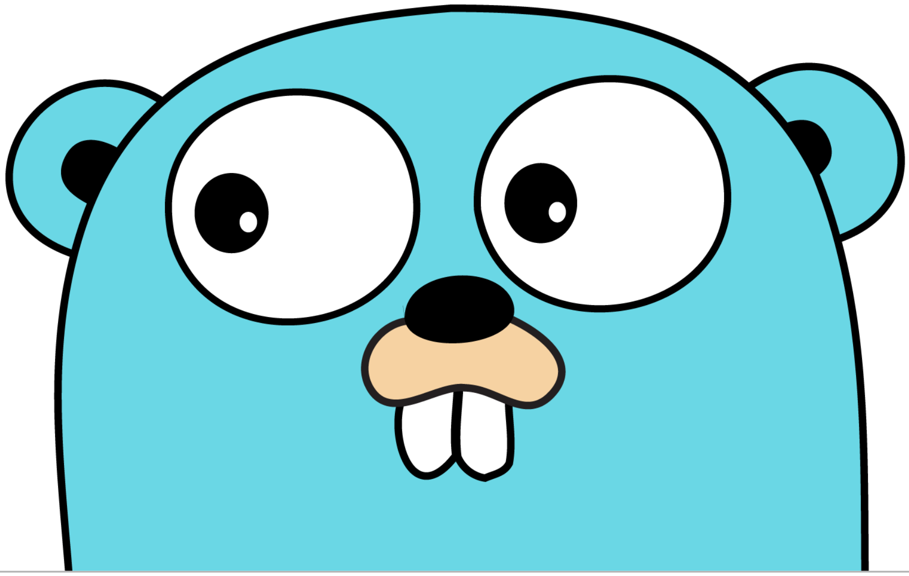
Go @ Couchbase

History

- Started small with Couchbase Labs
- Steady growth internally
- Now in production

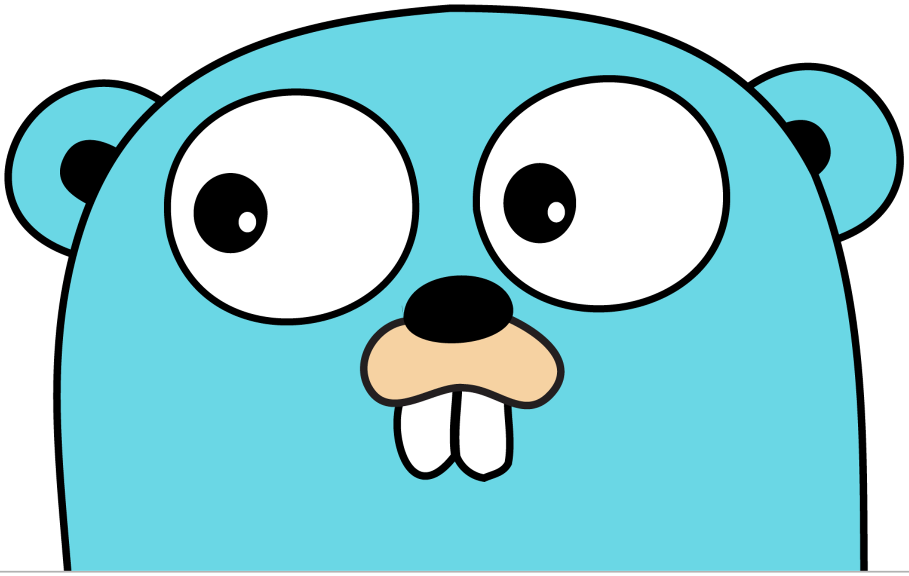
Things we like

- First class concurrency support with clean code
- Custom structs easily map to JSON
- Out of the box support for HTTP/HTTPS



Go + Couchbase

- Client SDK born in 2012
- Currently community supported
- Used for many projects internally
- Officially supported client soon



CouchbaseConnect

Intro to Couchbase SDK

Running Code in this Presentation

```go
func handleError(err error) {
    if err != nil {
        panic(err)
    }
}

func main() {
    err := doSomething()
    handleError(err)
    fmt.Printf("Success")
}
```

```
Success
Program exited.
```

Run Kill Close

Connect

```go
import (
    "fmt"

    "github.com/couchbaselabs/go-couchbase"
)

func main() {
    bucket, err := couchbase.GetBucket("http://localhost:8091/", "default", "demo")
    handleError(err)
    defer bucket.Close()
    fmt.Printf("Connected to Couchbase Bucket
}
```

```
Connected to Couchbase Bucket 'demo'

Program exited.
```

Run Kill Close

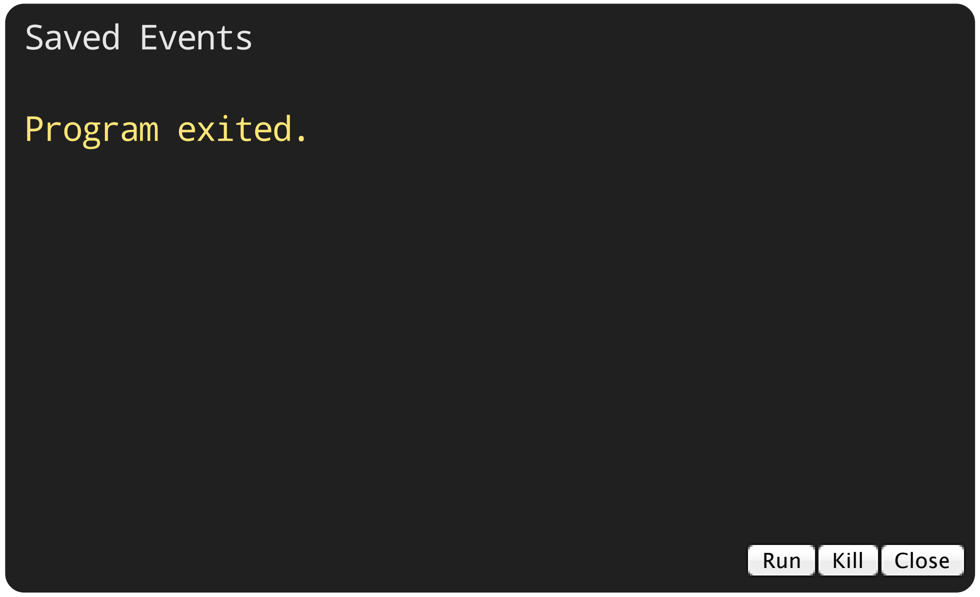
Working with Data

```go
type Event struct {
    Type  string `json:"type"`
    Name  string `json:"name"`
    Likes int    `json:"likes"`
}

func NewEvent(name string) *Event {
    return &Event{"event", name, 0}
}

func NewEventJSON(jsonbytes []byte) (event *Event) {
    err := json.Unmarshal(jsonbytes, &event)
    handleError(err)
    return
}

func (e *Event) String() string {
    return fmt.Sprintf("Event '%s', Likes: %d", e.Name, e.Likes)
}
```

- API supports working with any JSON serializable structure, or raw []byte
- Examples today will use the structure above

CRUD - Set

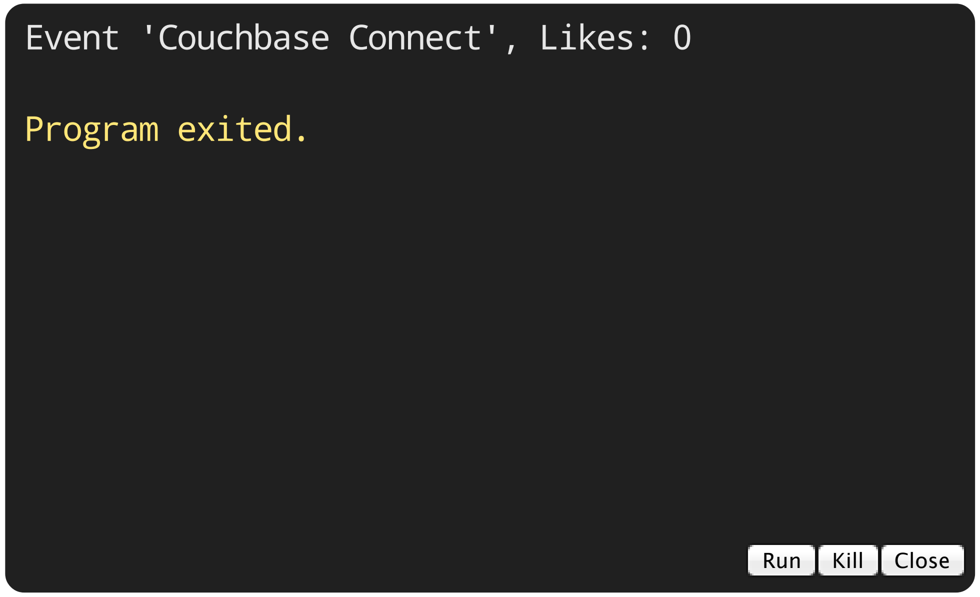
```
    event := NewEvent("Couchbase Connect")
    err = bucket.Set("cc2014", 0, event)
    handleError(err)

    event = NewEvent("GopherCon India")
    err = bucket.Set("gci2015", 0, event)
    handleError(err)

    fmt.Printf("Saved Events\n")
```



```
Saved Events

Program exited.
```

CRUD - Get

```
    var event Event
    err = bucket.Get("cc2014", &event)
    handleError(err)
    fmt.Println(&event)
```



```
Event 'Couchbase Connect', Likes: 0

Program exited.
```

Mutation Ops

- Add/AddRaw
- Append
- Cas/CasRaw
- Delete
- Incr
- Set/SetRaw

These all have very similar semantics to the other SDKs.

Handling Concurrency

- Upcoming Events
  - Couchbase Connect 2014
  - GopherCon India 2015

Liking Events - Wrong

```go
func likeEvent(bucket *couchbase.Bucket, id string) {
    var event Event
    err := bucket.Get(id, &event)
    handleError(err)
    event.Likes++
    bucket.Set(id, 0, event)
}
```

Concurrent Updates - Incorrect

```go
    var wg sync.WaitGroup
    for i := 0; i < 100; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            likeEvent(bucket, "cc2014")
        }()
    }
    wg.Wait()

    var event Event
    err = bucket.Get("cc2014", &event)
    handleError(err)
    fmt.Println(&event)
```

```
Event 'Couchbase Connect', Likes: 20

Program exited.
```

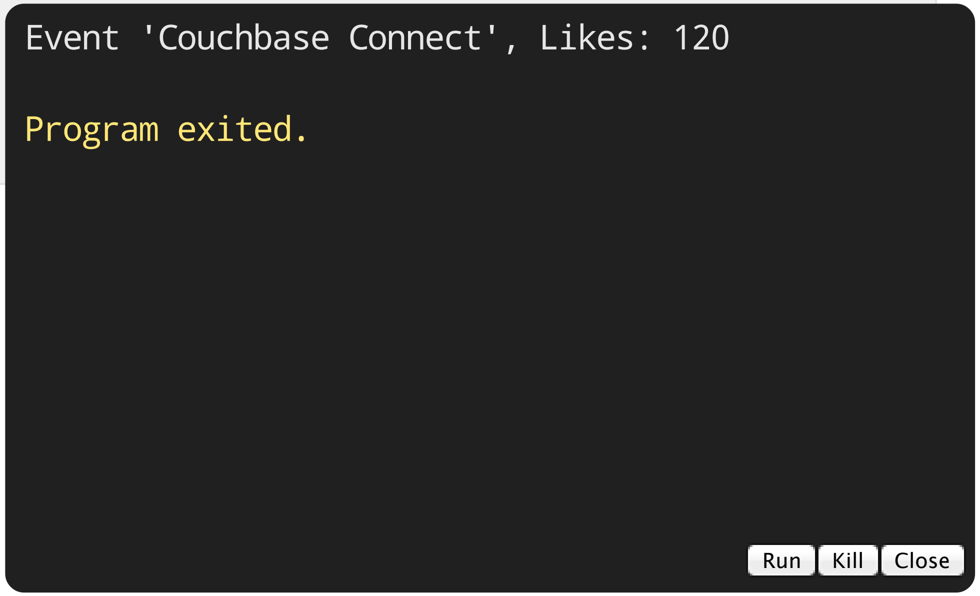
Run Kill Close

Liking Events - Right

```go
func likeEvent(bucket *couchbase.Bucket, id string) {
    bucket.Update(id, 0, func(current []byte) ([]byte, error) {
        event := NewEventJSON(current)
        event.Likes++
        return json.Marshal(event)
    })
}
```

Concurrent Updates - Safe using Update()

```go
var wg sync.WaitGroup
for i := 0; i < 100; i++ {
    wg.Add(1)
    go func() {
        defer wg.Done()
        likeEvent(bucket, "cc2014")
    }()
}
wg.Wait()

var event Event
err = bucket.Get("cc2014", &event)
handleError(err)
fmt.Println(&event)
```



```
Event 'Couchbase Connect', Likes: 120

Program exited.
```

Views - Top Events by Likes

```
function (doc, meta) {
    if(doc.type === 'event') {
        emit(doc.likes, null);
    }
}
```

- Emit 1 row for every event
- Key is the number of likes
- No value, we just use this view to find Event IDs

View Querying

```go
    args := map[string]interface{}{
        "stale":      false,
        "descending": true,
    }

    res, err := bucket.View("ddoc", "likes", args)
    handleError(err)

    for _, r := range res.Rows {
        fmt.Printf("Key: %v - DocID: '%s'\n", r.Key, r.ID)
    }
```

```
Key: 120 - DocID: 'cc2014'
Key: 0 - DocID: 'gci2015'

Program exited.
```

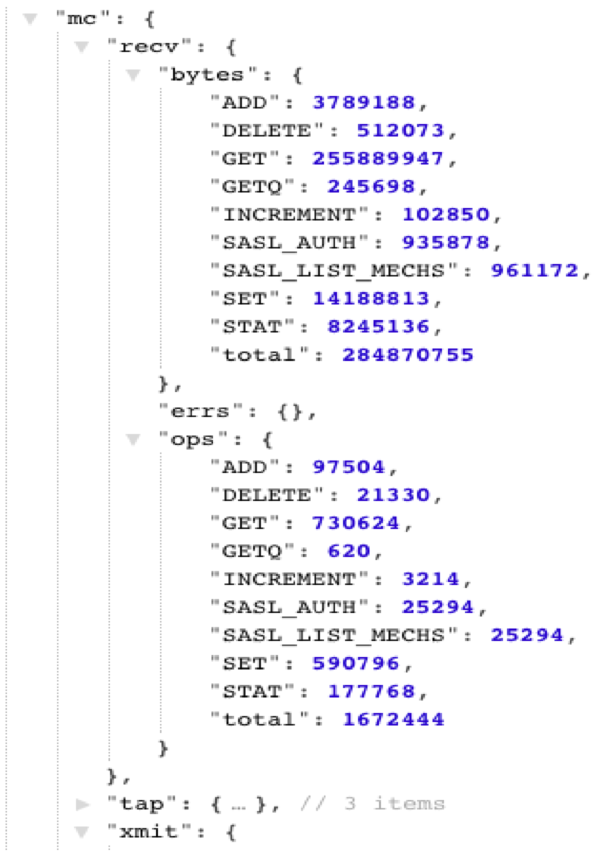
Run Kill Close

CouchbaseConnect

Behind the Curtains

expvar

```
import _ "github.com/couchbase/gomemcached/debug"
```



```
▼ "mc": {
  ▼ "recv": {
    ▼ "bytes": {
        "ADD": 3789188,
        "DELETE": 512073,
        "GET": 255889947,
        "GETQ": 245698,
        "INCREMENT": 102850,
        "SASL_AUTH": 935878,
        "SASL_LIST_MECHS": 961172,
        "SET": 14188813,
        "STAT": 8245136,
        "total": 284870755
      },
      "errs": {},
    ▼ "ops": {
        "ADD": 97504,
        "DELETE": 21330,
        "GET": 730624,
        "GETQ": 620,
        "INCREMENT": 3214,
        "SASL_AUTH": 25294,
        "SASL_LIST_MECHS": 25294,
        "SET": 590796,
        "STAT": 177768,
        "total": 1672444
      }
    },
  ► "tap": {…}, // 3 items
  ▼ "xmit": {
```

- Go stdlib hidden gem - http://golang.org/pkg/expvar/

Connection Pooling

- Operations on the Couchbase bucket ultimately need to talk to one of the Couchbase servers
- Bulk operations talk to multiple Couchbase servers
- Applications perform bucket operations on separate go routines, don't expect to be blocked by one another
- This is simulated by maintaining pools of connections to the underlying servers

Connection Pool Properties

- Return usable connection as fast as possible
- Creating connections is relatively expensive (as compared to reusing them)
- Don't create them unnecessarily
- Don't create too many of them
- The usual tuning operation here, too large a pool wastes resources, too small a pool means waiting for connections.

Connection Pool

```
type connectionPool struct {
    host        string
    auth        AuthHandler
    connections chan *memcached.Client
    createsem   chan bool
}

func newConnectionPool(host string, ah AuthHandler, poolSize, poolOverflow int) *connectionP
ool {
    return &connectionPool{
        host:        host,
        connections: make(chan *memcached.Client, poolSize),
        createsem:   make(chan bool, poolSize+poolOverflow),
        auth:        ah,
    }
}
```

- Using a buffered channel of connections as a thread-safe pool
- Using a buffered channel of bools to track overflow connections

Quick Go Channel Review

```
    // write to channel
    channel <- val

    // read from channel
    val = <-channel
```

Connection Pool - Get Connection 1

```go
func (cp *connectionPool) GetWithTimeout(d time.Duration) (rv *memcached.Client, err error)
{

    // short-circuit available connetions
    select {
    case rv, isopen := <-cp.connections:
        if !isopen {
            return nil, errClosedPool
        }
        return rv, nil
    default:
    }
```

- Select on the pool channel, if reading won't block, read and return connection
- If this would have blocked (no available connections in pool), proceed to next step

Connection Pool - Get Connection 2

```
    // create a very short timer, 1ms
    t := time.NewTimer(ConnPoolAvailWaitTime)
    defer t.Stop()

    select {
    case rv, isopen := <-cp.connections:
        // connection became available
        if !isopen {
            return nil, errClosedPool
        }
        return rv, nil
    case <-t.C:
        // waited 1ms
    }
}
```

Connection Pool - Get Connection 3

```
        t.Reset(d) // reuse original timer for full timeout
        select {
        case rv, isopen := <-cp.connections:

            // keep trying to get connection from main pool
            if !isopen {
                return nil, errClosedPool
            }
            return rv, nil

        case cp.createsem <- true:

            // create a new connection
            rv, err := cp.mkConn(cp.host, cp.auth)
            if err != nil {
                <-cp.createsem // buffer only allows poolSize + poolOverflow
            }
            return rv, err

        case <-t.C:


            // exceeded caller provided timeout
            return nil, ErrTimeout
```

Connection Pool - Summary

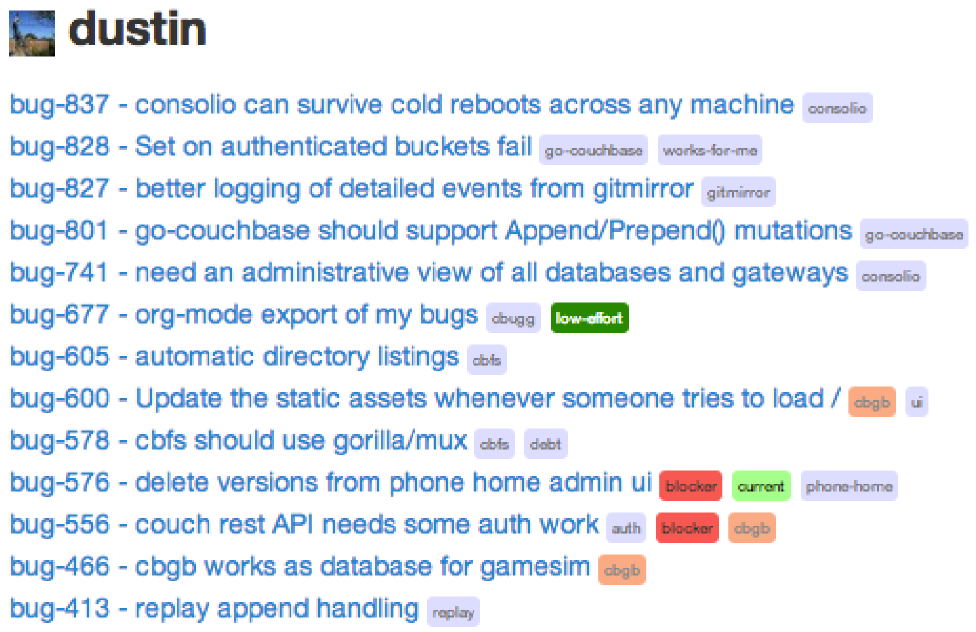
- Somewhat dense block of Go code
- Worth your time to try to understand it
- This current version of the code was refined during performance benchmarks of sync_gateway
- See Dustin's blog

http://dustin.sallings.org/2014/04/25/chan-pool.html

CouchbaseConnect

Applications

cbugg - bug tracker on top of Couchbase



dustin

bug-837 - consolio can survive cold reboots across any machine consolio
bug-828 - Set on authenticated buckets fail go-couchbase works-for-me
bug-827 - better logging of detailed events from gitmirror gitmirror
bug-801 - go-couchbase should support Append/Prepend() mutations go-couchbase
bug-741 - need an administrative view of all databases and gateways consolio
bug-677 - org-mode export of my bugs cbugg low-effort
bug-605 - automatic directory listings cbfs
bug-600 - Update the static assets whenever someone tries to load / cbgb ui
bug-578 - cbfs should use gorilla/mux cbfs debt
bug-576 - delete versions from phone home admin ui blocker current phone-home
bug-556 - couch rest API needs some auth work auth blocker cbgb
bug-466 - cbgb works as database for gamesim cbgb
bug-413 - replay append handling replay

- Typical CRUD operations, bugs, comments attachments
- Uses a large number of features in the SDK, but not a complex application

cbugg - How it Works

- Go HTTP server exposing REST API
- Also serves static resources HTML/CSS/JS/images
- End-user functionality through HTML5/AngularJS interface
- Bugs, Comments stored in Couchbase
- Searchable through Couchbase-Elasticsearch integration
- Attachments stored in cbfs

cbugg - why?

Ensure that the engineers building Couchbase rely on it being a high quality product.

cbugg - Deploying Views?

- 3-4 developers
- Important functionality built on top of views
- Each with local Couchbase, and shared production instance
- How do we propagate changes to design documents/views?
- Need to promote changes up to production, and back down to other developers

Version Controlled View Definitions

```
type viewMarker struct {
	Version   int       `json:"version"`
	Timestamp time.Time `json:"timestamp"`
	Type      string    `json:"type"`
}

const ddocKey = "/@ddocVersion"
const ddocVersion = 1
const designDoc = `
{
  "views": {
    "likes": {
      "map": "function (doc, meta) { if(doc.type === 'event') { emit(doc.likes, null);} }"

    }
  }
}`
```

- viewMarker tracks the latest deployed version
- we store viewMarker in *ddocKey*
- when we update *designDoc*, we bump the *ddocVersion*

Automatic View Definition Updating

```go
    marker := viewMarker{}
    err := bucket.Get(ddocKey, &marker)
    if err != nil && !gomemcached.IsNotFound(err) {
        handleError(err)
    }
    if marker.Version < ddocVersion {
        fmt.Printf("Installing new version of views (old version=%v)\n",
            marker.Version)
        doc := json.RawMessage([]byte(designDoc))
        err = bucket.PutDDoc("ddoc", &doc)
        handleError(err)
        marker.Version = ddocVersion

        marker.Timestamp = time.Now().UTC()
        marker.Type = "ddocmarker"

        bucket.Set(ddocKey, 0, &marker)
    } else {
        fmt.Printf("Version %v already installed\n", marker.Version)
    }
```

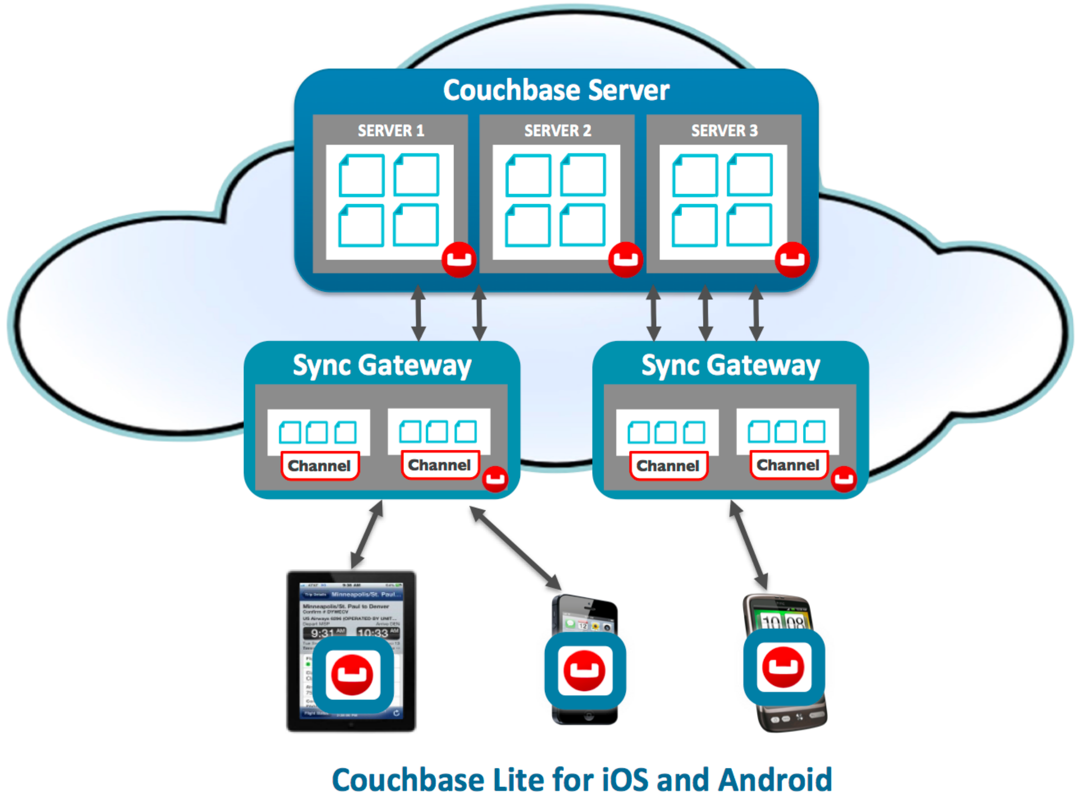
Automatic View Definition Updating

```
const ddocKey = "/@ddocVersion"
const ddocVersion = 1
const designDoc = `
{
  "views": {
    "likes": {
      "map": "function (doc, meta) { if(doc.type === 'event') { emit(doc.likes, null);} }"
    }
  }
}`


func main() {
    bucket, err := couchbase.GetBucket("http
    handleError(err)

    updateDesignDocs(bucket)
}
```

```
Installing new version of views (old version=0)

Program exited.
```

Run Kill Close

Sync Gateway



- Server-side component integrating Couchbase Server and Couchbase Lite

Sync Gateway - How it Works

- Shared nothing architecture, need to scale Sync Gateway nodes just like Couchbase Server
- Sync Gateway maintains caches of data structures used for replication
- Relies on the Couchbase TAP protocol to be notified of changes
- These notifications invalidate/update cache

TAP

```go
    args := memcached.DefaultTapArguments()
    feed, err := bucket.StartTapFeed(&args)
    handleError(err)

    go func() {
        time.Sleep(1 * time.Second)
        for i := 0; i < 5; i++ {
            bucket.SetRaw(fmt.Sprintf("tap-%d", i), 0, []byte("x"))
        }
    }()

    fmt.Printf("Listening to TAP:\n")
    for op := range feed.C {
        fmt.Printf("Received %s\n", op.Strin
        if len(op.Value) > 0 && len(op.Value
            fmt.Printf("\tValue: %s\n", op.V
        }
    }
```

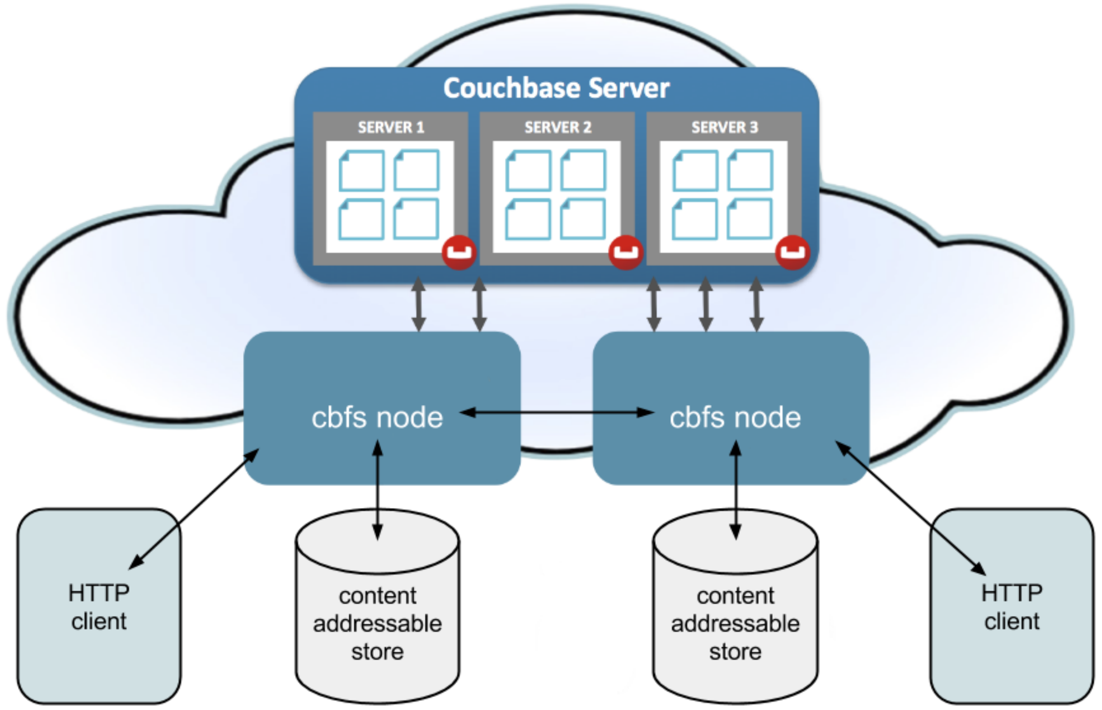
```
Listening to TAP:
Received <TapEvent Mutation, key="tap-4" (1 bytes
        Value: x
Received <TapEvent Mutation, key="tap-3" (1 bytes
        Value: x
Received <TapEvent Mutation, key="tap-2" (1 bytes
        Value: x
Received <TapEvent Mutation, key="tap-0" (1 bytes
        Value: x
Received <TapEvent Mutation, key="tap-1" (1 bytes
        Value: x
```

Run Kill Close

From TAP to DCP

- TAP nearing end of life
- With 3.0 comes DCP (Database Change Protocol)
- Go SDK will have one of the first DCP implementations
- DCP only supported for internal replication at this time

cbfs

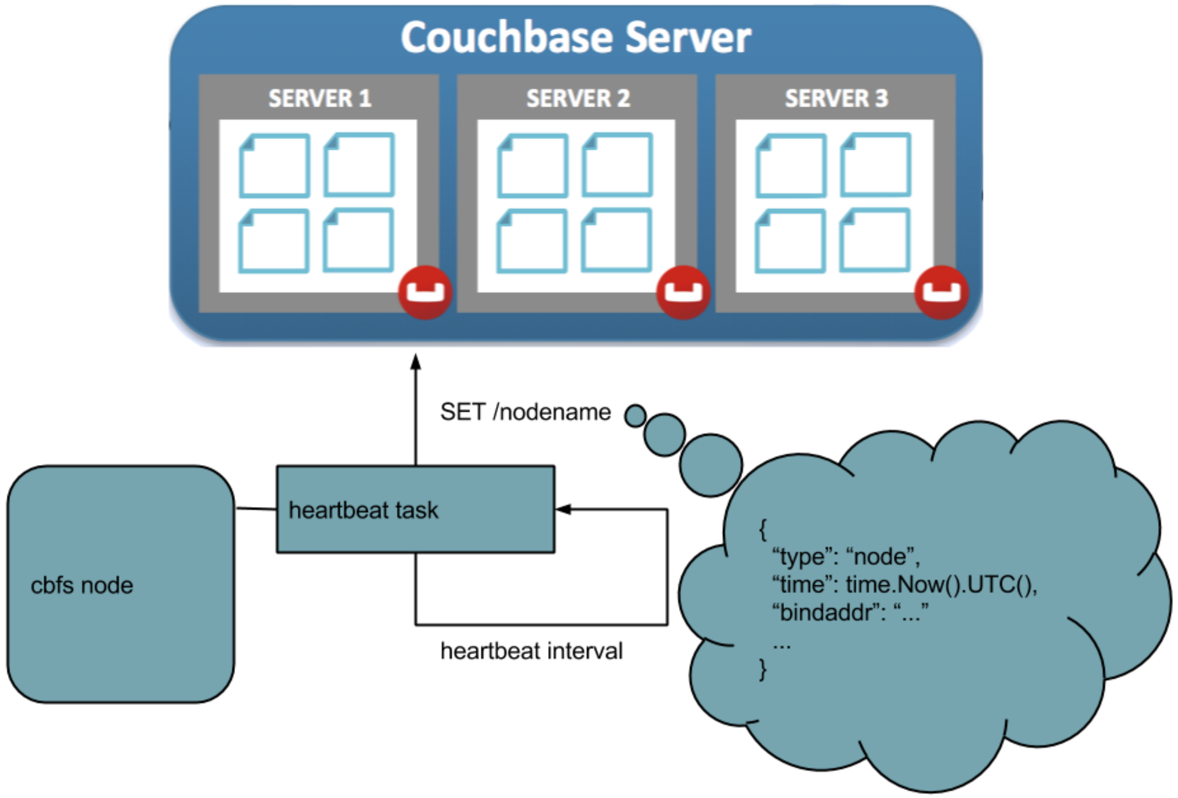


- Distributed file storage on top of Couchbase

cbfs - How it Works

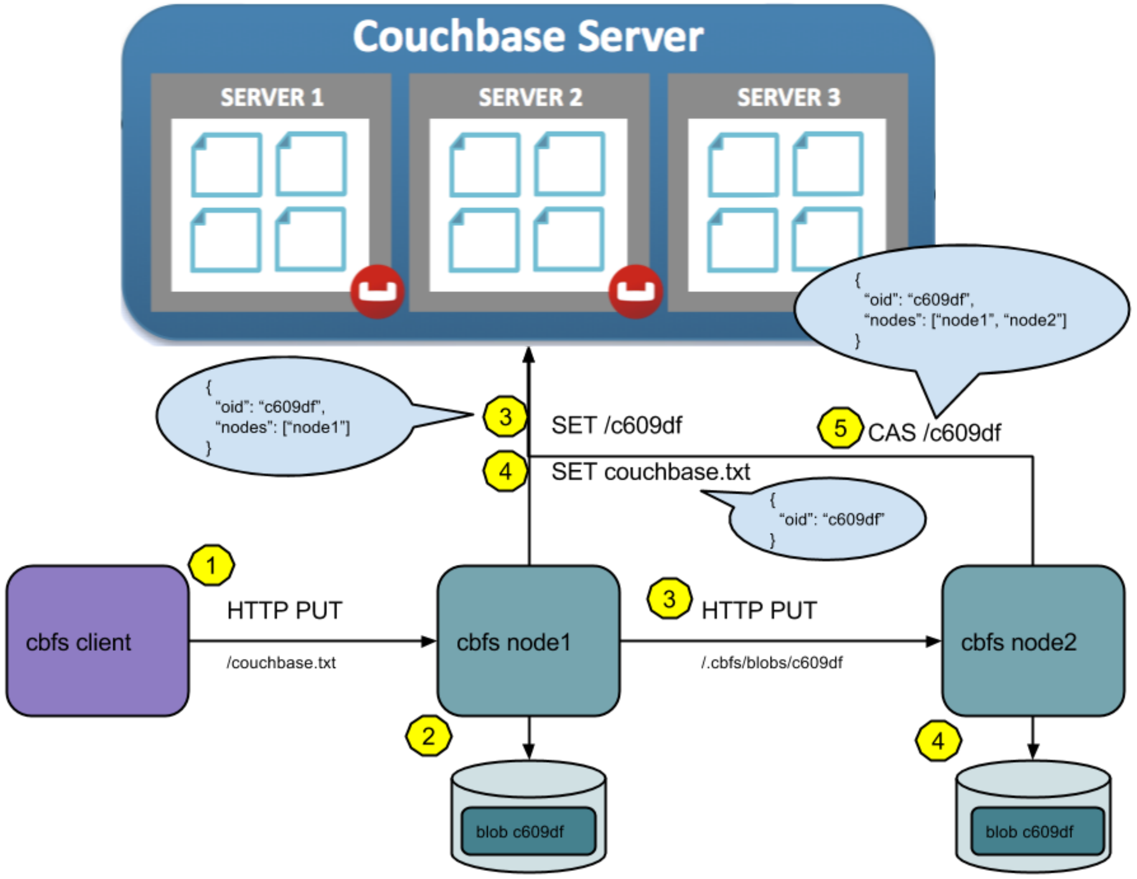
- Clients upload/download files via HTTP
- Nodes store file content locally in a content-addressable store (filename = content hash)
- File metadata is stored in Couchbase
- Nodes announce themselves/discover one another through Couchbase
- Nodes ensure a minimum replica count is maintained to safely store data

cbfs - Node Heartbeat

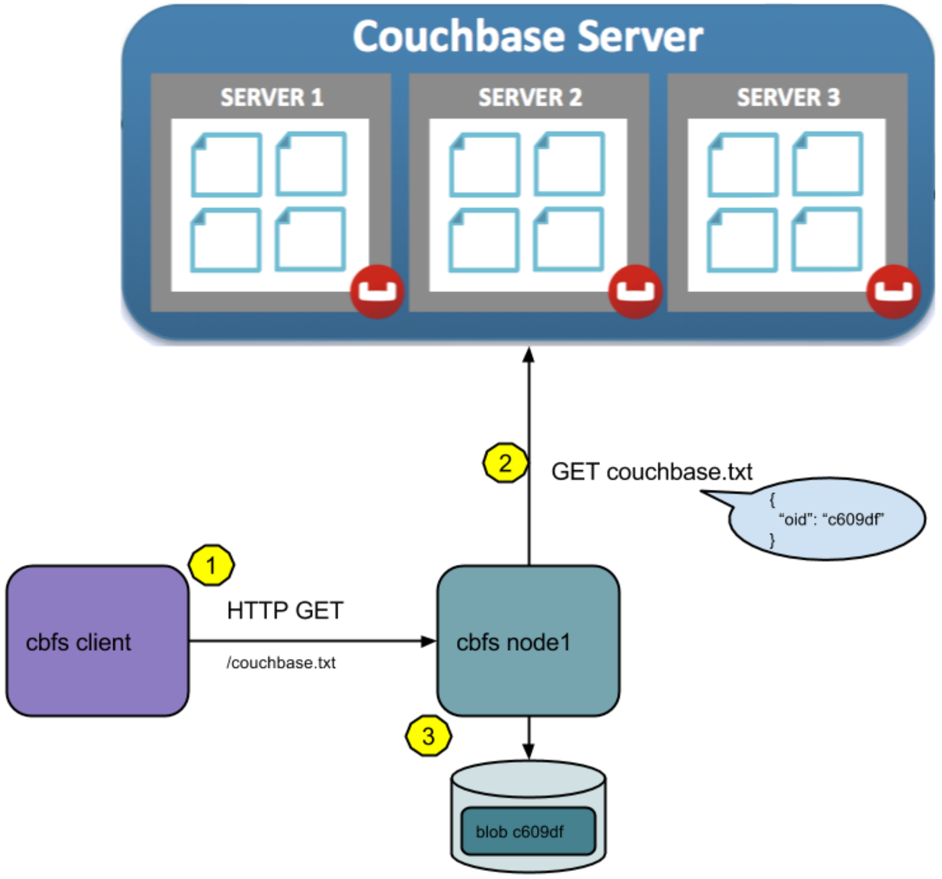


```
{
  "type": "node",
  "time": time.Now().UTC(),
  "bindaddr": "..."
  ...
}
```

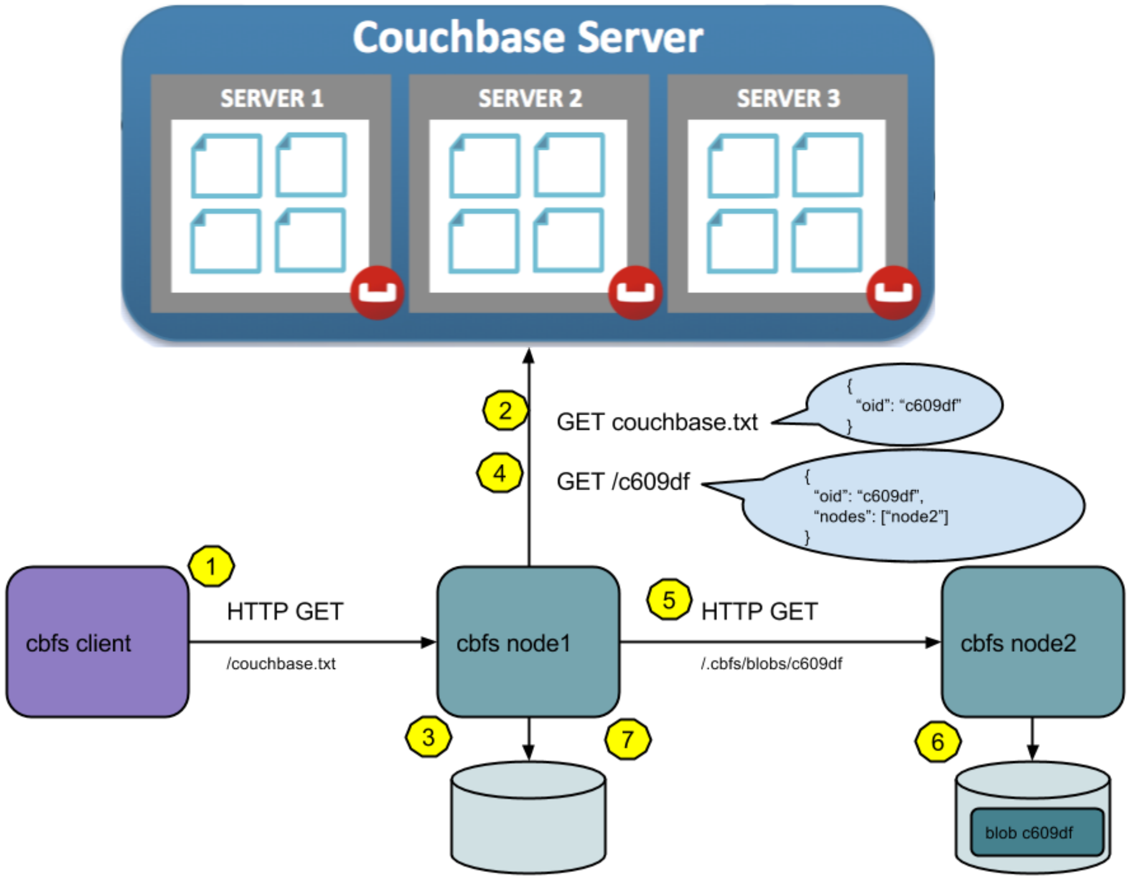
cbfs - Add Document



cbfs - Get Document (blob exist on node)

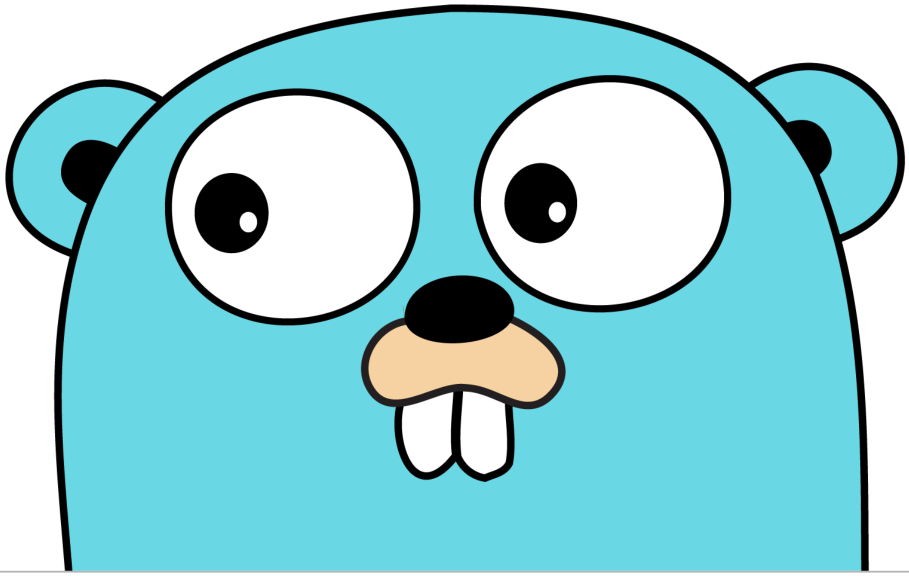


cbfs - Get Document (blob does NOT exist on node)



Go + Couchbase

- Go - First class concurrency support with clean code
- Go - JSON mapping to custom structs
- Go - Out of the box support for HTTP/HTTPS
- Couchbase - Fast and scalable JSON storage
- Go + Couchbase = Powerful starting point for your app



Thank you

Marty Schoch

marty@couchbase.com (mailto:marty@couchbase.com)

http://github.com/couchbaselabs/go-couchbase (http://github.com/couchbaselabs/go-couchbase)

https://github.com/couchbaselabs/cbugg (https://github.com/couchbaselabs/cbugg)

https://github.com/couchbase/sync_gateway (https://github.com/couchbase/sync_gateway)

https://github.com/couchbaselabs/cbfs (https://github.com/couchbaselabs/cbfs)

@mschoch (http://twitter.com/mschoch)