Building Scalable Deployments with Multiple Goroutines

GopherCon UK 2022

Sadie Freeman

Github Repo:
https://bit.ly/3C33ICS

@Sadef
on
Twitter

Agenda

- Step by step example
- Scaling for speed with Goroutines
- Scaling for load with deployments
- Case study: NFL All Day

Coffee Anyone?

How do we serve:

- as much coffee as possible
- to as many people as possible
- as fast as possible?

Speed + Load

- Take payment
- Steam milk
- Make espresso

1 No Concurrency

```
func main() {
    start := time.Now()
    PayForCoffee()
    MakeEspresso()
    SteamMilk()
    log.Printf("Coffee made, 1 customer served")

    timeTaken := time.Since(start)
    log.Printf("Took %s to serve coffee", timeTaken)
}
```

1 No Concurrency

```go
func PayForCoffee() {
    time.Sleep(2 * time.Second)
    log.Printf("Coffee paid for 💰")
}

func MakeEspresso() {
    time.Sleep(2 * time.Second)
    log.Printf("Espresso made ☕")
}

func SteamMilk() {
    time.Sleep(2 * time.Second)
    log.Printf("Milk steamed 🥛")
}
```

1 No Concurrency

go run main.go

```
Coffee paid for 💰
Espresso made ☕
Milk steamed 🥛
Coffee made, 1 customer served
Took 6.004392209s to serve coffee
```

Serve more customers

```go
func ServeCustomer(w http.ResponseWriter, r *http.Request) {
    start := time.Now()

    numCustomers, err := strconv.Atoi(strings.TrimPrefix(r.URL.Path, "/serve-customer/"))
    if err != nil || numCustomers == 0 {
        numCustomers = 1
    }

    count := 0
    for i := 0; i < numCustomers; i++ {
        MakeCoffee()
        count++
    }

    timeTaken := time.Since(start)
    log.Printf("Took %s to serve coffee to %v customer(s)", timeTaken, count)
}
```

```go
func MakeCoffee() {
    PayForCoffee()
    MakeEspresso()
    SteamMilk()
}
```

Serve more customers

go run main.go

curl http://localhost:8080/serve-customer/3

```
Took 18.007806208s to serve coffee to 3 customer(s)
```


Speed

Use a Goroutine

```
go MakeCoffee()
```

Threads: Set of instructions that can be run independently

Concurrently: Happening at the same time

Use a Goroutine

```
Took 49.958µs to serve coffee to 3 customer(s)
Coffee paid for 💰
Coffee paid for 💰
Coffee paid for 💰
Espresso made ☕
Espresso made ☕
Espresso made ☕
Milk steamed 🥛
Milk steamed 🥛
Milk steamed 🥛
```

4 Add Wait Group

```
wg := sync.WaitGroup{}
count := 0
for i := 0; i < numCustomers; i++ {
    wg.Add(1)
    go MakeCoffee(&wg)
    count++
}
wg.Wait()
```

```
func MakeCoffee(wg *sync.WaitGroup) {
    defer wg.Done()
    PayForCoffee()
    MakeEspresso()
    SteamMilk()
}
```

4 Add Wait Group

go run main.go

curl http://localhost:8080/serve-customer/3

```
Coffee paid for 💰
Coffee paid for 💰
Coffee paid for 💰
Espresso made ☕
Espresso made ☕
Espresso made ☕
Milk steamed 🥛
Milk steamed 🥛
Milk steamed 🥛
Took 6.005458958s to serve coffee to 3 customer(s)
```

5 Add MORE Goroutines

```
func MakeCoffee(wg *sync.WaitGroup) {
    defer wg.Done()

    newWg := sync.WaitGroup{}
    newWg.Add(3)
    go PayForCoffee(&newWg)
    go MakeEspresso(&newWg)
    go SteamMilk(&newWg)
    newWg.Wait()
}
```

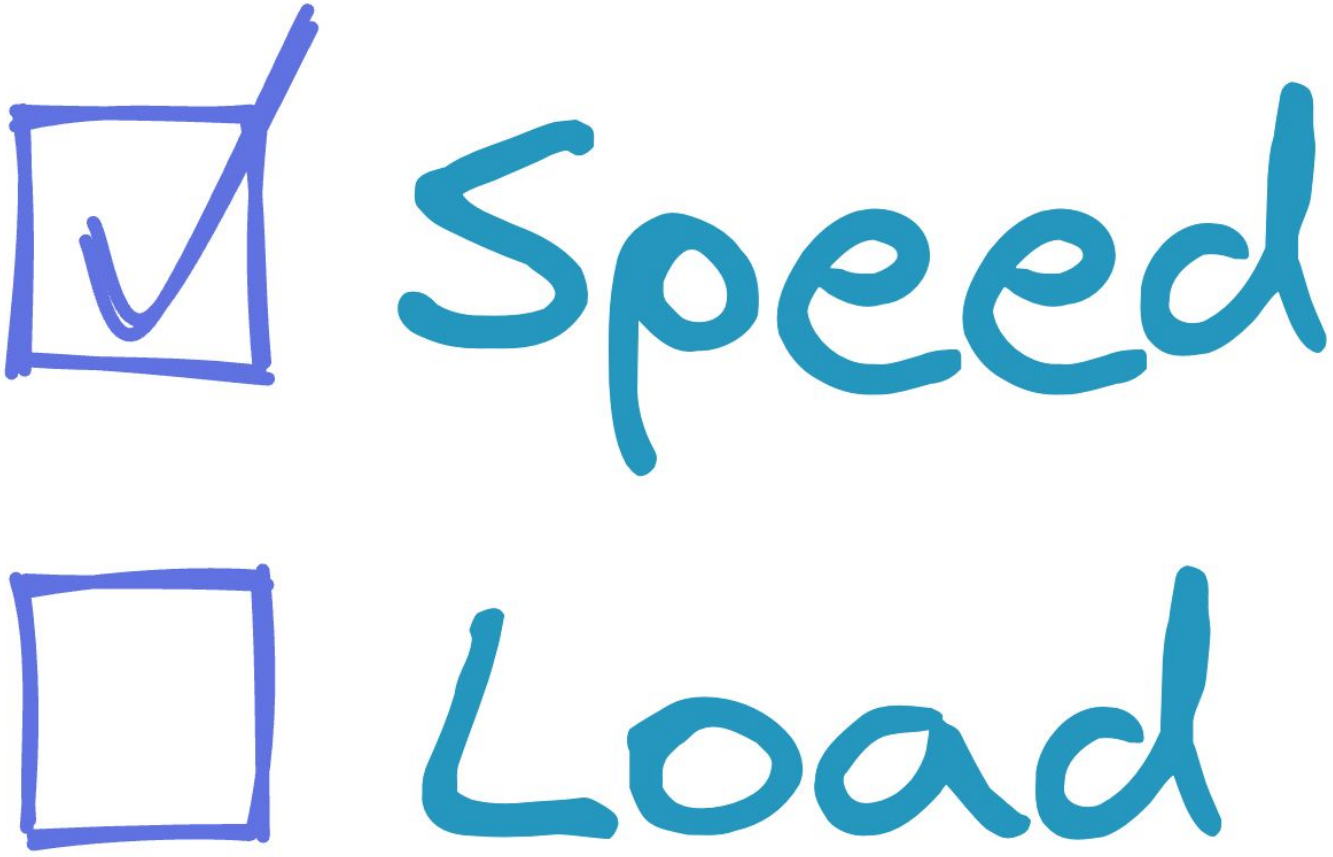
5 Add MORE Goroutines

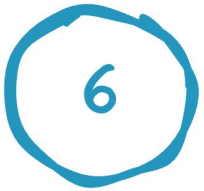
go run main.go

curl http://localhost:8080/serve-customer/3

```
Coffee paid for 💰
Milk steamed 🥛
Milk steamed 🥛
Coffee paid for 💰
Espresso made ☕
Milk steamed 🥛
Coffee paid for 💰
Espresso made ☕
Espresso made ☕
Took 2.001751333s to serve coffee to 3 customer(s)
```

MORE CUSTOMERS!!!!

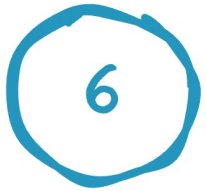

Speed
Load



Containerize

Dockerfile

```
FROM golang:1.18.4-alpine

WORKDIR /app

COPY go.mod ./
RUN go mod download

COPY *.go ./

RUN go build -o /coffee-shop

EXPOSE 8080

CMD [ "/coffee-shop" ]
```



Containerize

Makefile

```
build:
    docker build --tag coffee-shop .

run:    build
    docker run -d --name coffee-shop -p 8080:8080 coffee-shop

stop:
    docker stop coffee-shop
    docker container rm coffee-shop
```

6 Containerize

```
docker build --tag coffee-shop .
[+] Building 1.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 37B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/golang:1.18.4-alpine
=> [1/6] FROM docker.io/library/golang:1.18.
4-alpine@sha256:9b2a2799435823dbf6fef077a8ff7ce83ad26b382ddabf22febfc333926400e4
=> [internal] load build context
=> => transferring context: 1.33kB
=> CACHED [2/6] WORKDIR /app
=> CACHED [3/6] COPY go.mod ./
=> CACHED [4/6] RUN go mod download
=> CACHED [5/6] COPY *.go ./
=> CACHED [6/6] RUN go build -o /coffee-shop
=> exporting to image
=> => exporting layers
=> => writing image sha256:edb9c4e028c289b5a86f914ff8d43826313a7e37dfac0512894b4f7f251429f6
=> => naming to docker.io/library/coffee-shop
```

coffee-shop	IN USE	latest	8adda91d151f	20 minutes ago	331.3 MB

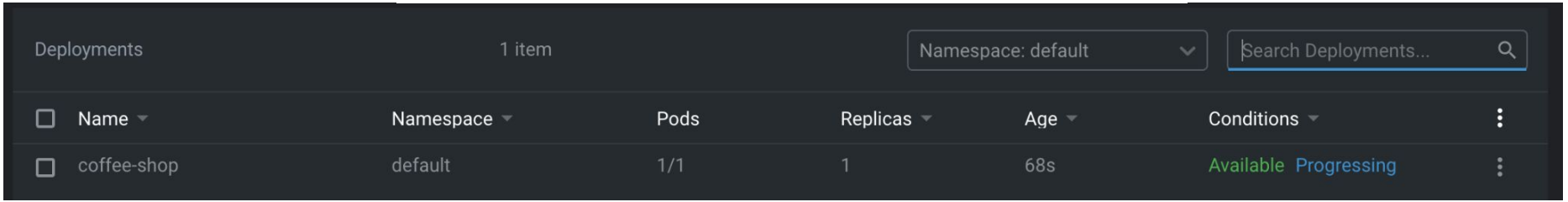
coffee-shop 43bd5d6d7c8f	coffee-shop	Running	8080	23 minutes ag	

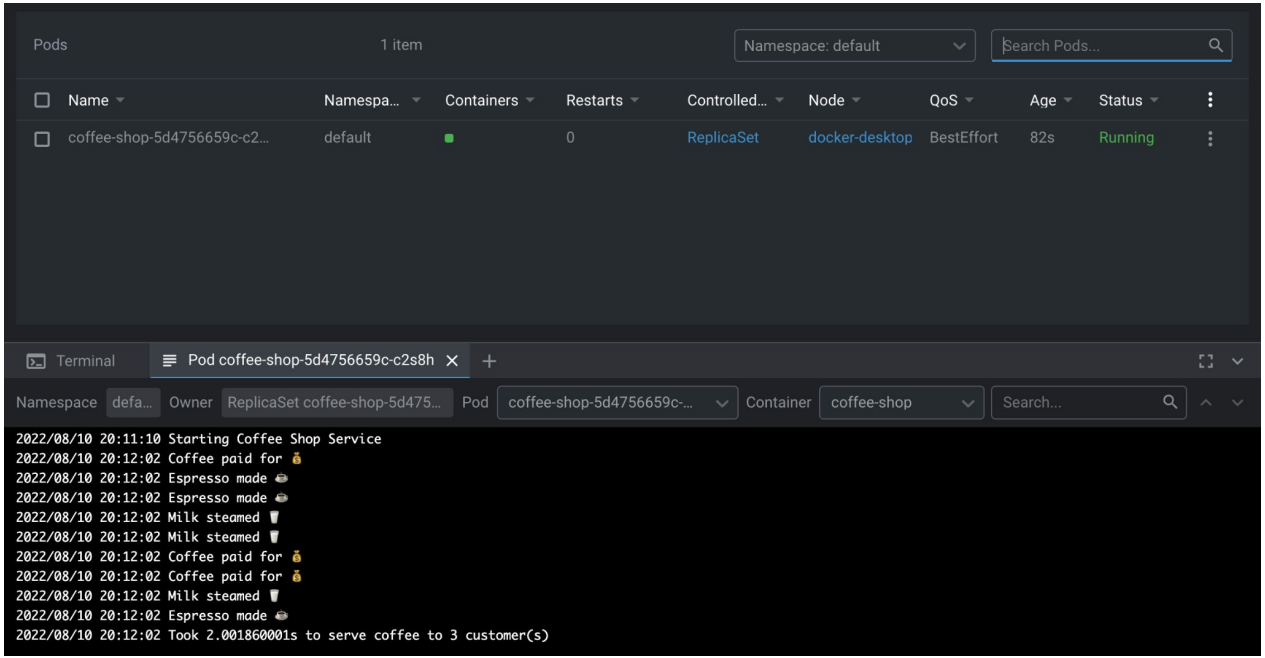
6 Containerize

7 Deploy on Kubernetes

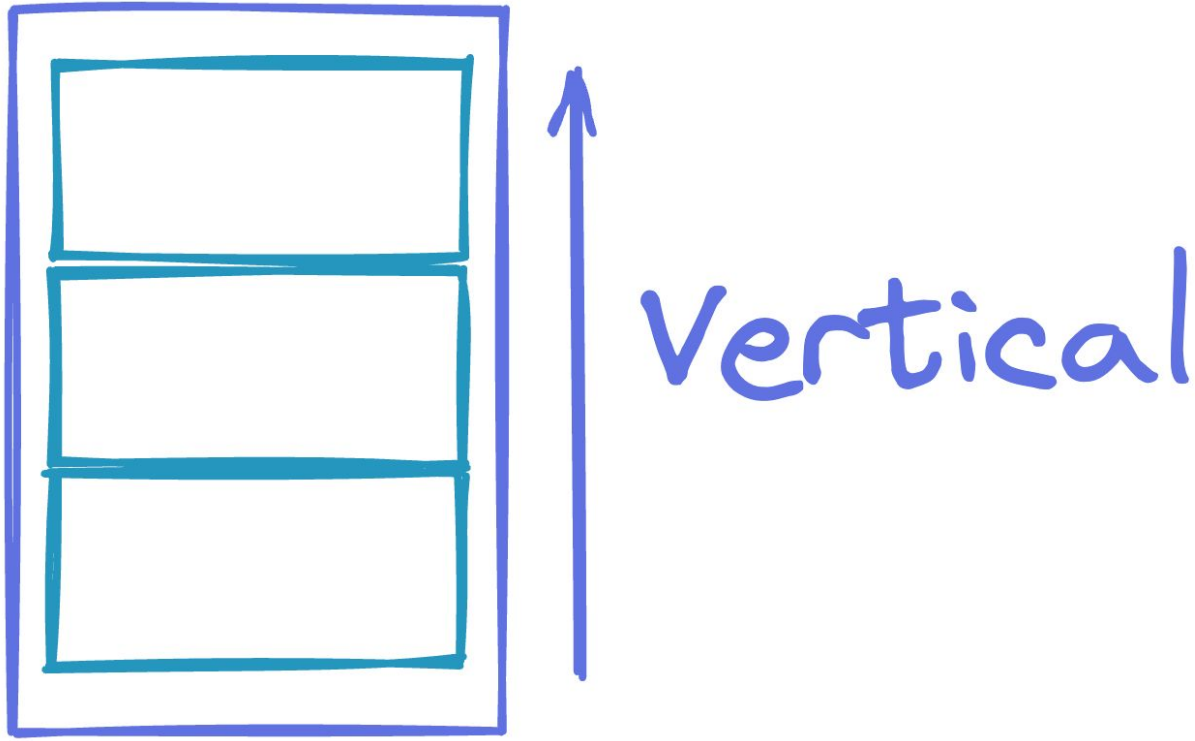
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: coffee-shop
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      coffee-shop: web
  template:
    metadata:
      labels:
        coffee-shop: web
    spec:
      containers:
      - name: coffee-shop
        image: coffee-shop
        imagePullPolicy: Never
---
```

7 Deploy on Kubernetes





Scaling Option 1



8 Set Resources

```
resources:
  limits:
    cpu: 1m
    memory: 10Mi
  requests:
    cpu: 1m
    memory: 10Mi
```

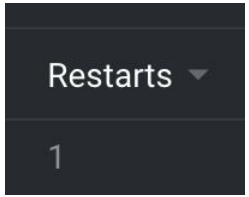
8 Set Resources

3 → 2.002422959s

30 → 2.001635459s

300 → 3.782164496s

3000 → OOM Killed



Last Status terminated

Reason:Reason: OOMKilled - exit code: 137

9 Bump resources

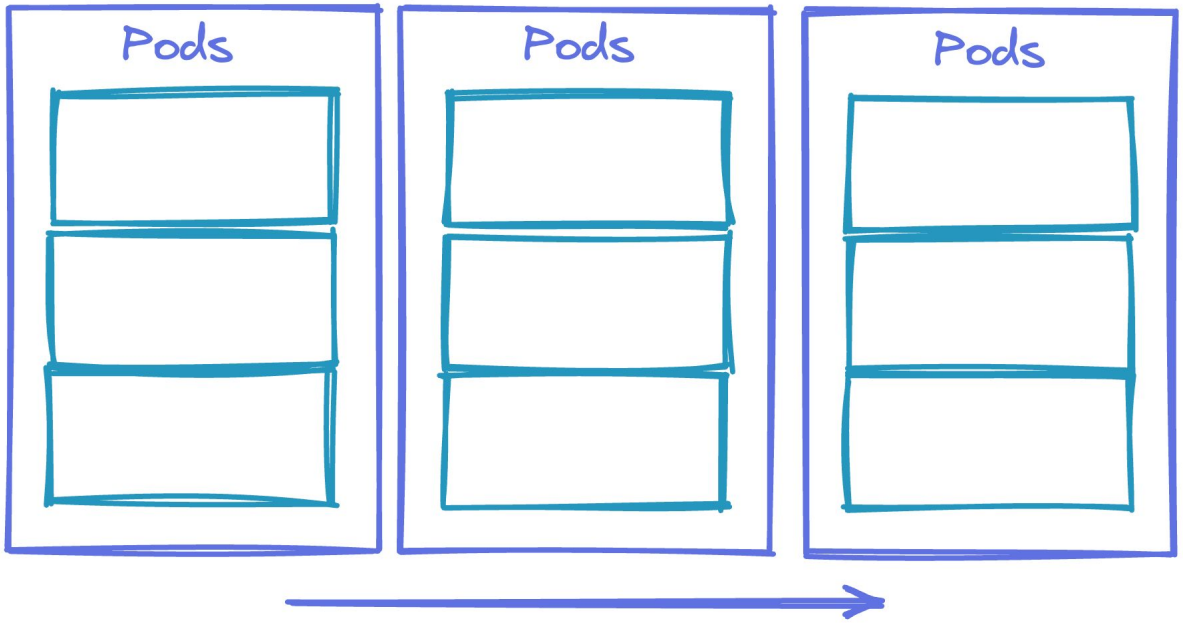
```
resources:
  limits:
    cpu: 4
    memory: 3092M
  requests:
    cpu: 2
    memory: 1024M
```

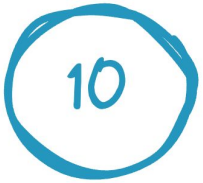
9 Bump resources

30,000 → 3.797142293s

3,000,000 → OOM Killed

Scaling Option 2





Scale up pods

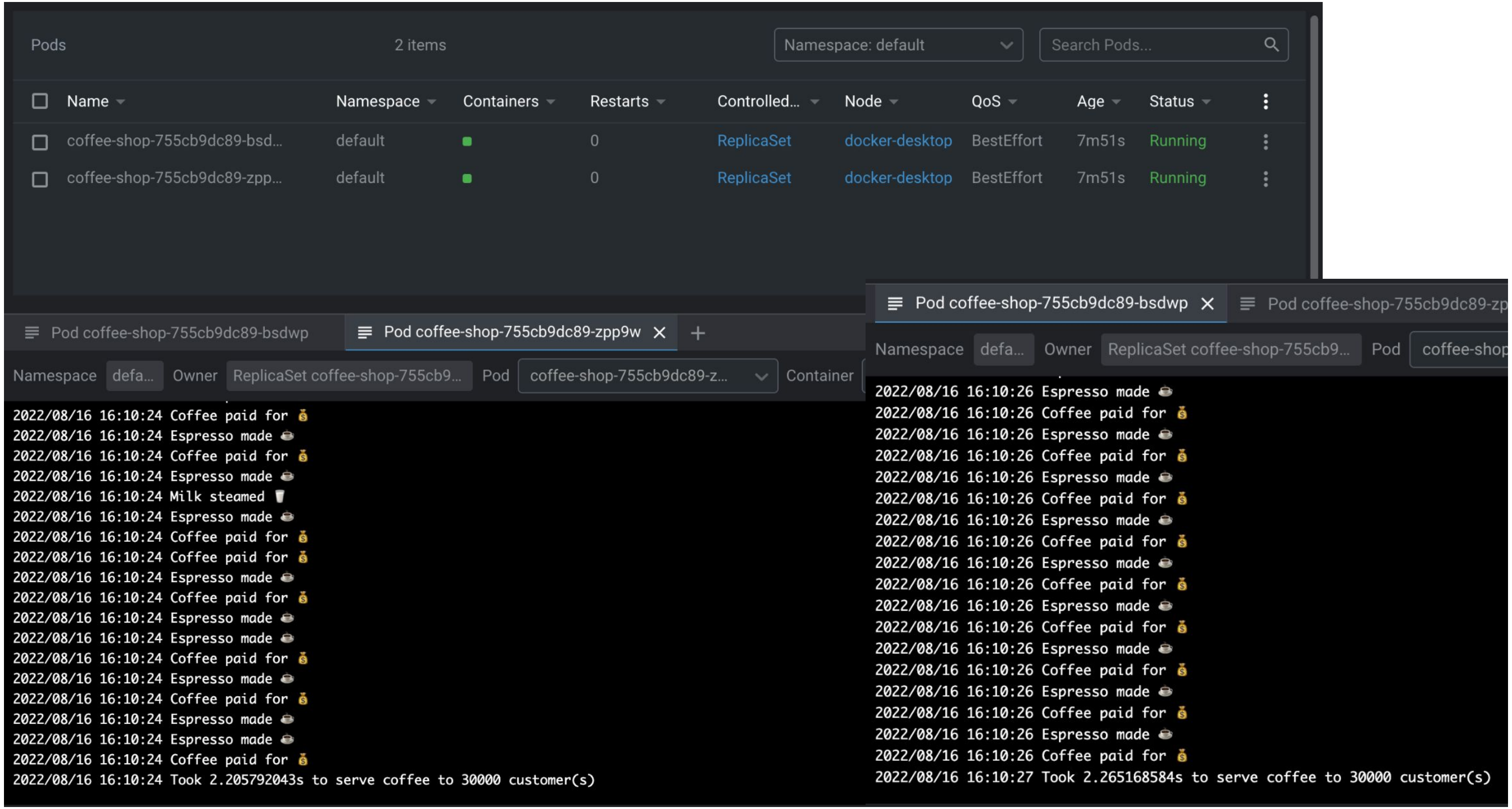
```
replicas: 2
```

10 Scale up pods

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress

metadata:
  namespace: default
  name: coffee-shop
  labels:
    ingress-controller: nginx
  annotations:
    kubernetes.io/ingress.class: "nginx"

spec:
  rules:
  - host: "localhost"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: coffee-shop
            port:
              number: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: coffee-shop
  labels:
    run: coffee-shop
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    run: coffee-shop
  type: NodePort
  sessionAffinity: None
```

10 Scale up pods



10 Scale up pods

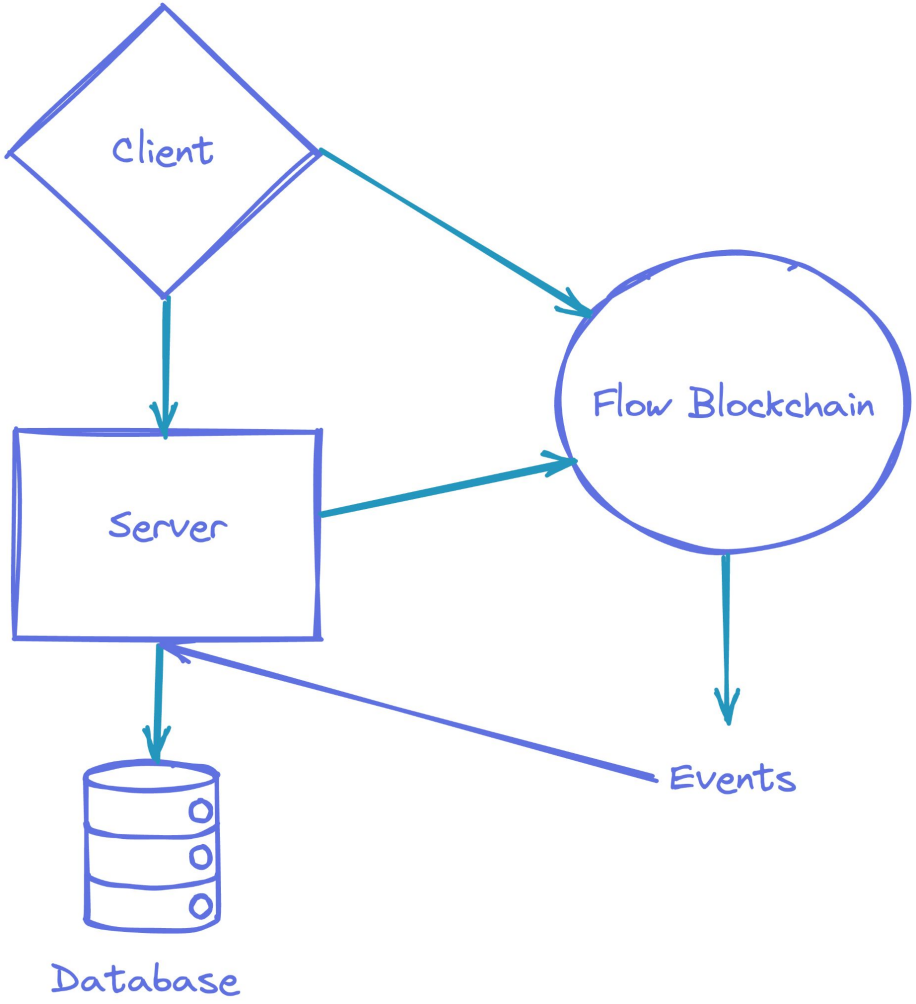
30,000 → 2.265168584s

30,000 → 2.210111751s

Take Payment	2 Pods
Steam Milk	2 Pods
Make Espresso	4 Pods

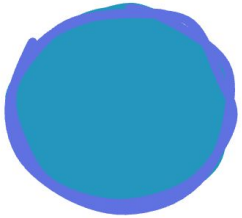
NFL All Day

Architecture overview


Client
Flow Blockchain
Server
Events
Database

PDS -> Pack Delivery Service

- Check Pack events
- Minting
- Sending Transactions
- Checking Transaction

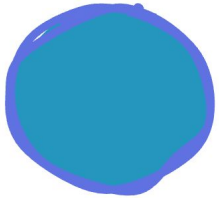


No Concurrency

One at a time

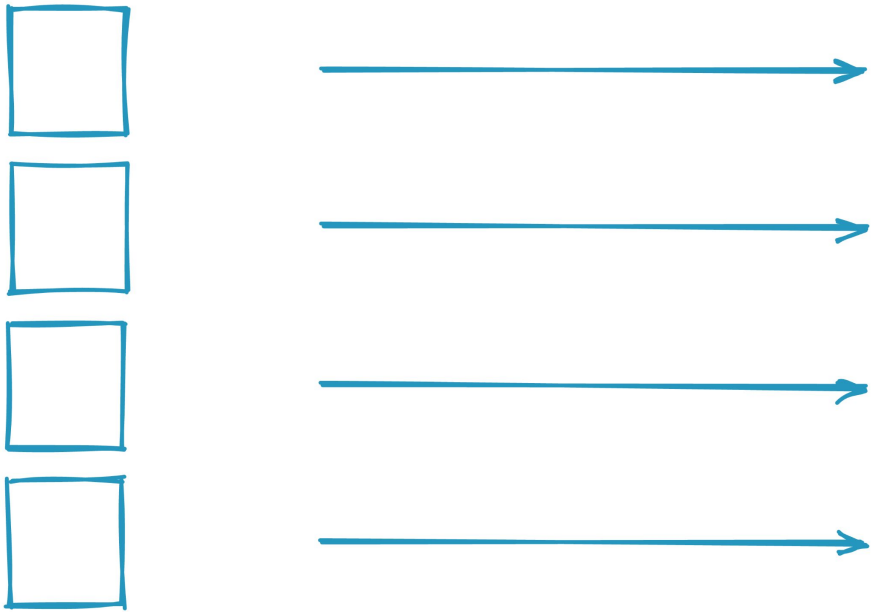
No Concurrency

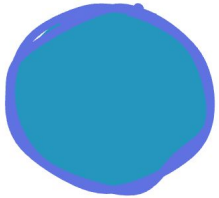
```
for {
        select {
        case <-ticker.C:
            handleMinting()
            pollCirculatingPackContractEvents()
            handleSentTransactions()
            handleSendableTransactions()
        case <-app.quit:
            cancel()
            ticker.Stop()
            return
        }
    }
```



Use Goroutines

Four at a time





Use Goroutines

```
go mintingPoller(app)
go packContractEventsPoller(app)
go sentTransactionsPoller(app)
go sendableTransactionPoller(app)
```

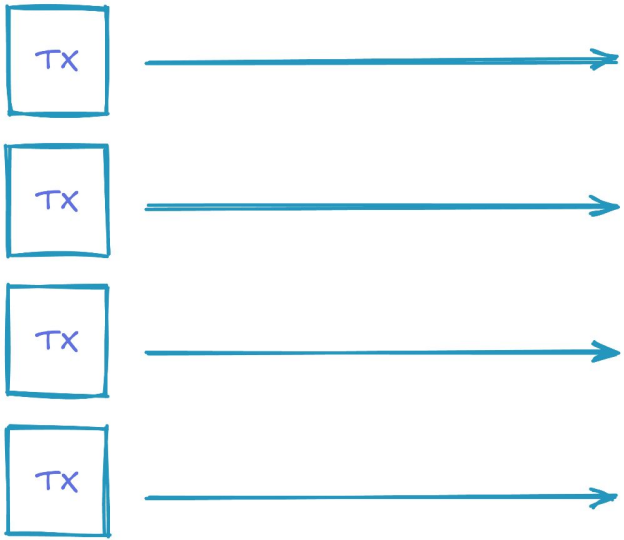
Sendable Transactions

One at a time

Add More Goroutines

Sendable Transactions

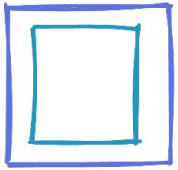
MORE at a time



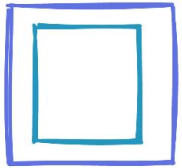
Add More Goroutines

```
for _, transaction := range ts {
    wg.Add(1)
    go func(ctx context.Context, dbtx *gorm.DB, wg *sync.WaitGroup, tx *transactions.StorableTransaction) {
        defer wg.Done()
        if err := processSendableTransaction(ctx, app, logger, dbtx, tx); err != nil {
            logger.Warn("error processing storable transaction", err)
        }
    }(ctx, dbtx, &wg, &transaction)
}
wg.Wait()
```

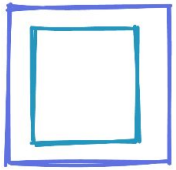
Split Deployments



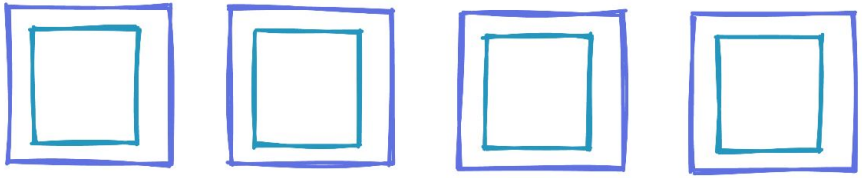
Minting



Pack Events

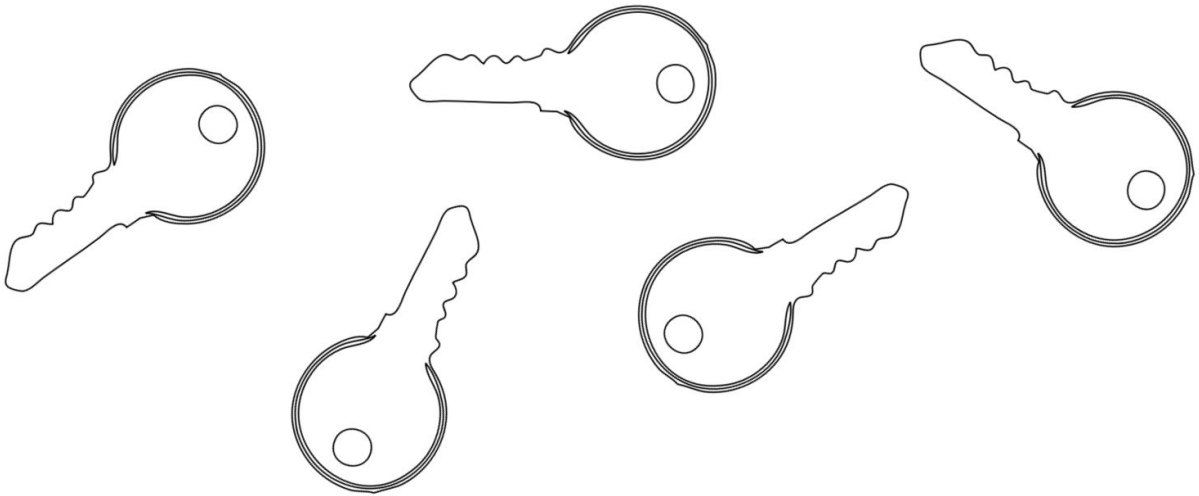


Sent Transactions



Sendable Transactions

Keys?



How did we scale?

- Use Goroutines (they’re pretty cool)
- Scale horizontally + vertically
- Split different responsibilities in different deployments

- Start with no concurrency
- Use Goroutines
- Use Goroutines inside of Goroutines
- Scale vertically - add resource
- Scale horizontally - add pods
- Split deployments according to Goroutine usage

Thank You!

Eric L

Fabio K

Quinn H

Jude Z

Ryan W

Nima B

Ana M

Deewai A

James H

Ksenia O

Brant H

Sean E

James H

Thilina R

Mike G

Kate N

Dan B

@Sadef on Twitter