Além dos Containers na Nuvem

QConSP - Containers & Devops
26 April 2017

Guilherme Rezende
Globo.com

About me

- Software Engineer at Globo.com/Tsuru
- **@guilhermebr** (GitHub)
- **in/guilhermebr** (LinkedIn)
- **@gbrezende** (Twitter)

Agenda

- Cloud Native App
- Containers
- CaaS
- Swarm
- Kubernetes
- PaaS
- Tsuru ❤
- FaaS

Cloud Native App

- Born to be Containerized
- 12Factor Manifesto
- Ephemeral File System
- Endpoint for Healthcheck
- Endpoint for Metrics
- Resilient
- Stateless
- Horizontal Scalable
- Better use of Resources
- Smaller Bills
- Culture Change

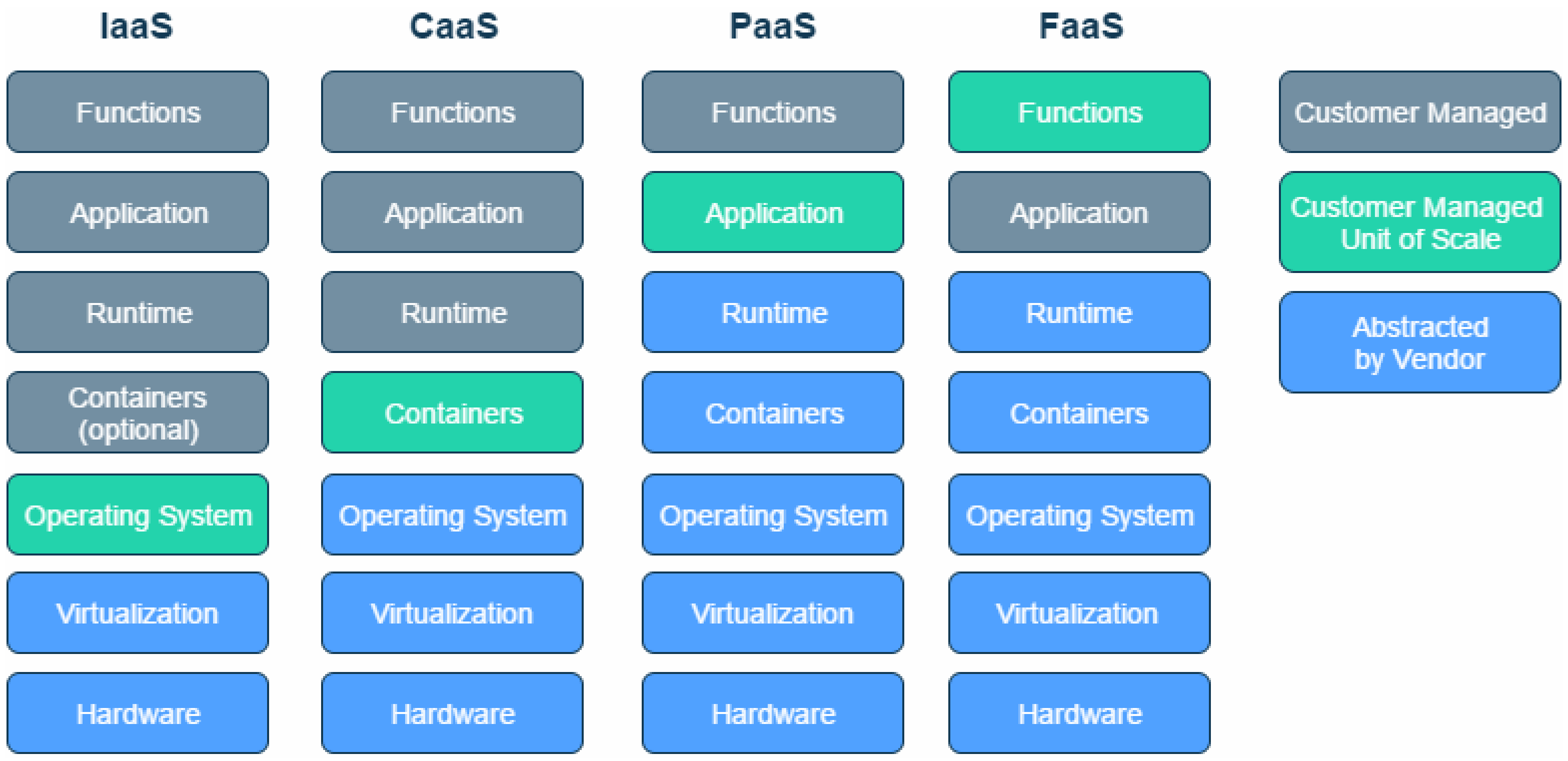
Our App

- **Thumbor ❤**
- github.com/thumbor/thumbor
- Smart Imaging Service
- Crop, Resizing, Filters on-demand
- Thumbnail Service
- by globo.com

Containers

- 10+y/o (2000 Jails - FreeBSD)
- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Linux control Groups (cgroups) - memory, cpu, i/o
- Chroot
- LXC by IBM 2008
- Docker 2013 - focus on developers
- OCI (image-spec, runtime-spec, runC, containerd)
- Market: 2016: $762m | 2020: $20b in revenue

Overview of *aaS



CaaS

- Between IaaS and PaaS
- Manage Containers using API's and Web Interface
- Pay only for Container Resources (compute instances, LB, Schedulling)
- Basically the Orchestration Platform

CaaS Providers

- Azure Container Service (DC/OS, Swarm, Kubernetes)
- Google Container Engine (Kubernetes)
- IBM Bluemix Container Service (Kubernetes)
- Amazon ECS

Orchestrators

- Swarm
- Kubernetes
- Apache Mesos
- Docker Cluster by Tsuru

Swarm

- Docker, Inc
- SwarmKit
- Docker Engine embedded >/
- Swarm Mode
- Services
- Tasks
- Compose File
- Written in Go :)

Demo

Demo: Compose File

- docker-compose.yml

```yaml
version: '3.1'
services:
  thumbor:
    image: apsl/thumbor:latest
    environment:
      - ALLOW_UNSAFE_URL=True
      - DETECTORS=['thumbor.detectors.face_detector','thumbor.detectors.feature_detector']
      - LOG_LEVEL=debug
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure
    restart: always
    ports:
      - "80:8000"
```

Demo: Google Cloud Compute and Docker Machine

```
$ gcloud config set project qconsp-demo
$ gcloud auth application-default login

$ docker-machine create --driver google \
                        --google-project qconsp-demo \
                        --google-open-port 80/tcp \
                        swarm-master

$ docker-machine create --driver google \
                        --google-project qconsp-demo \
                        --google-open-port 80/tcp \
                        swarm-node1

$ gcloud compute instances list
```

Demo: Swarm Cluster

```
$ eval $(docker-machine env swarm-master)

$ docker swarm init --advertise-addr {SWARM_MASTER_PRIVATE_IP}
  * Copy the join command

$ eval $(docker-machine env swarm-node1)
  * Paste the join command

$ eval $(docker-machine env swarm-master)

$ docker node ls
```

Demo: Deploy

```
$ docker stack deploy --compose-file=docker-compose.yml qconsp

$ docker stack ps qconsp

$ open http://{SWARM_EXTERNAL_IP}/unsafe/200x300/smart/https://goo.gl/yudmrQ
```

Kubernetes

- Google, RedHat, Microsoft, CoreOS, ...
- Google Borg - Cluster Manager
- 10+y/o
- Stable API
- Deployments
- Pods
- Services
- Written in Go :)

Demo

Demo: deployment.yml

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: thumbor
spec:
  replicas: 2
  selector:
  matchLabels:
      app: thumbor
      version: "latest"
  template:
    metadata:
      name: thumbor
      labels:
        app: thumbor
        version: "latest"
      spec:
```

(cont...)

Demo: deployment.yml

```
(...cont)

      containers:
      - name: thumbor
        image: apsl/thumbor:latest
        env:
        - name: LOG_LEVEL
          value: "DEBUG"
        - name: ALLOW_UNSAFE_URL
          value: "True"
        - name: DETECTORS
          value: "['thumbor.detectors.face_detector', 'thumbor.detectors.feature_detector']"
        - name: THUMBOR_PORT
          value: "8000"
        imagePullPolicy: Always
```

Demo: service.yml

```
kind: Service
apiVersion: v1
metadata:
  name: thumbor
  labels:
    app: thumbor
spec:
  ports:
  - port: 8000
    nodePort: 30000
  selector:
    app: thumbor
  type: NodePort
```

Demo: Google Container Engine

```
$ gcloud container clusters create qconsp --num-nodes=2

$ gcloud compute instances list |grep gke
```

Demo: Deploy

```
$ kubectl create -f .

$ kubectl get all

$ open http://{GKE_NODE_IP}:30000/unsafe/200x300/smart/https://goo.gl/yudmrQ
```

PaaS

- Zero Downtime Deploy
- Application Healing
- Auto-scaling
- LB
- Focus on Developer Experience
- Deploy from Source Code

Some Paas

- Heroku
- Google App Engine
- OpenShift
- CloudFoundry
- Tsuru

Tsuru ❤

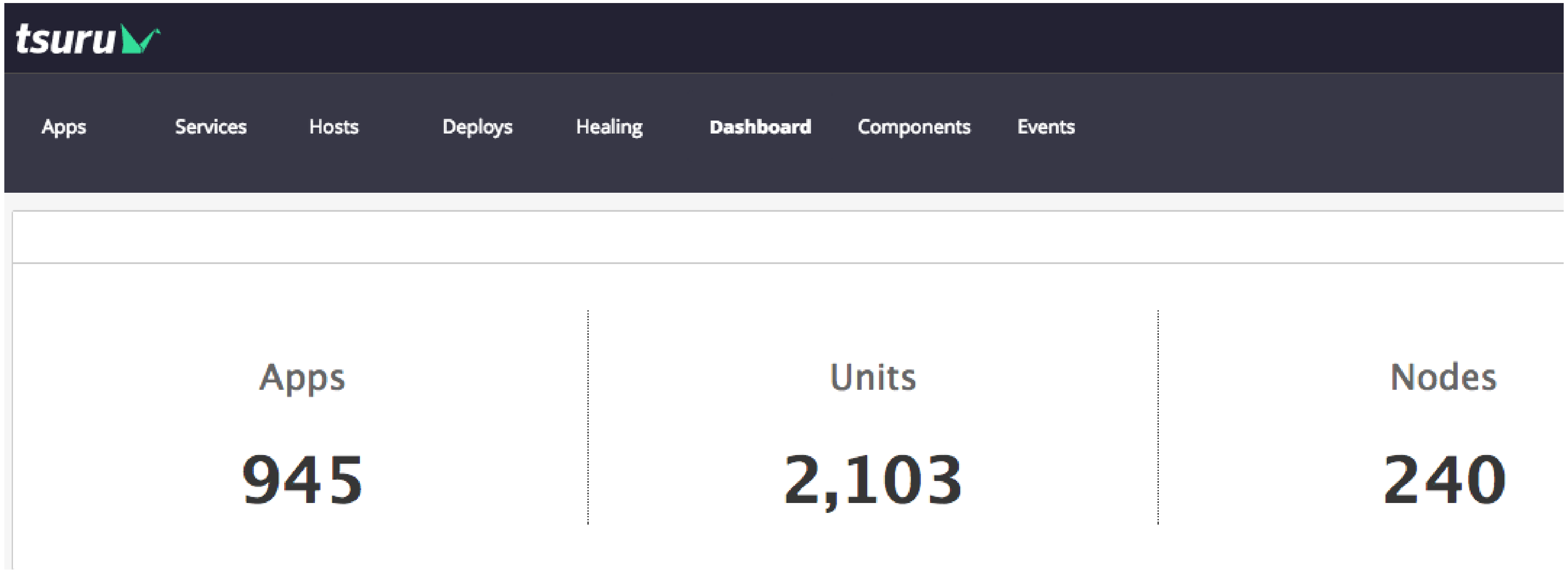
Premisses

- Reduce Time to Market
- Open Source
- Simplicity
- No Vendor Lock-in
- Deploy Safe
- Multi-Cluster
- Written in Go :)

At Globo.com

- Cartola
- GloboPlay
- APIs
- Feed
- WebMedia

At Globo.com



Apps	Units	Nodes
945	2,103	240

Out of Globo.com

- JusBrasil
- eduK
- Nuveo
- Filmow
- Rivendel
- Stone
- Hotel Urbano
- EF.com (London)
- École nationale supérieure d'informatique pour l'industrie et l'entreprise

Demo

- docs.tsuru.io
- github.com/tsuru/tsuru

Demo: tsuru-gce.yml

```yaml
name: tsuru-qconsp
components:
    tsuru:
        version: latest
driver:
    name: google
    private-ip-interface: ens4
    options:
      google-project: qconsp-demo
      google-tags: tsuru
      google-open-port:
        - 8080/tcp
        - 80/tcp
```

Demo: Tsuru Installer

```
Install client from: https://github.com/tsuru/tsuru-client/releases

$ tsuru install -c tsuru-gce.yml
```

Demo: Deploy

```
$ tsuru app-create qconsp python

$ tsuru env-set LOG_LEVEL="DEBUG" ALLOW_UNSAFE_URL="True" DETECTORS="['thumbor.detectors.face_detector',

$ tsuru app-deploy -i apsl/thumbor:latest -a qconsp

$ tsuru app-list # get app url

$ open http://{APP_URL}/unsafe/200x300/smart/https://goo.gl/yudmrQ
```

FaaS

- Serverless for the Dev
- Events, Workers, Custom Code
- CGI (request in, start process to handle it, return something)
- HotFunctions
- Nano-services...?
- Why the Hype?

Case Globo.com

- **Backstage/Functions ❤**
- github.com/backstage/functions
- Writen in ... NodeJS =\
- Sandbox
- Support: JS, Ruby
- Publication Platform
- Custom Code

Enterprise

- Lambda
- Google Cloud Functions
- IBM OpenWhisky

Open Source

- **IronFunctions** - github.com/iron-io/functions
- **GoFn** - github.com/nuveo/gofn
- **Fission** - github.com/fission/fission (Kubernetes Only)
- **Funktion** - github.com/funktionio/funktion (Kubernetes Only)
- **faas** - github.com/fission/fission (Swarm Only)

Demo

Demo: Deploy on Tsuru

```
# redis

$ tsuru app-create functions-redis static

$ tsuru app-deploy -i redis:latest -a functions-redis
```

Demo: Deploy on Tsuru

- functions

```
$ tsuru platform-add nodejs
$ tsuru app-create functions nodejs
$ tsuru app-info -a functions-redis

...
Units [web]: 1
+--------------+---------+------------+-------+
| Unit         | State   | Host       | Port  |
+--------------+---------+------------+-------+
| 10843bf6fba3 | started | 10.128.0.6 | 32789 |
+--------------+---------+------------+-------+
...

$ tsuru env-set REDIS_ENDPOINT=redis://10.128.0.6:32789/0 -a functions
$ git clone https://github.com/backstage/functions
$ cd functions/
$ tsuru app-deploy -a functions .
```

Demo: Using Functions

- hello.js

```
function main(req, res) {
  const name = (req.body && req.body.name) || "World"
  res.send({ say: `Hello ${name}!` })
}
```

- Create function route

```
$ curl -i -X POST http://{FUNCTIONS_APP_URL}/functions/example/hello \
    -H 'content-type: application/json' \
    -d '{"code":"function main(req, res) \
                {\n  const name = (req.body && req.body.name) || \"World\"\n \
                res.send({ say: `Hello ${name}!` })\n}\n"}'
```

- Testing

```
$ curl -i -X PUT http://{FUNCTIONS_APP_URL}/functions/example/hello/run \
    -H 'content-type: application/json'

$ curl -i -X PUT http://{FUNCTIONS_APP_URL}/functions/example/hello/run \
    -H 'content-type: application/json' \
    -d '{"name": "QconSP"}'
```

Demo: Using Functions

- qconsp.js

```
function main(req, res) {
  const say = (req.body && req.body.say)
  res.send({ say: `${say} I hope you enjoyed this talk at QconSP! Thank you ;)` })
}
```

- Create function route

```
$ curl -i -X POST http://{FUNCTIONS_APP_URL}/functions/example/qconsp \
    -H 'content-type: application/json' \
    -d '{"code":"function main(req, res) \
                {\n  const say = (req.body && req.body.say)\n \
                res.send({ say: `${say}! I hope you enjoyed this talk! Thank you ;)` })\n}\n"}'
```

- Running Pipeline

```
$ curl -g -i \
    -X PUT 'http://{FUNCTIONS_APP_URL}/functions/pipeline?steps[0]=example/hello&steps[1]=example/qconsp
    -H 'content-type: application/json' \
    -d '{"name": "Galera"}'
```

Thank you

Guilherme Rezende
Globo.com
@gbrezende (http://twitter.com/gbrezende)