KubeCon | CloudNativeCon

China 2018

Stop hitting yourself!

Michael Russell

> **Computers do what you say,**
>
> **Software should do what you mean.**
>
> - Me

> **Software is easy to use**
>
> **but hard to run.**
>
> - Also me

About me

Australian living in the Netherlands

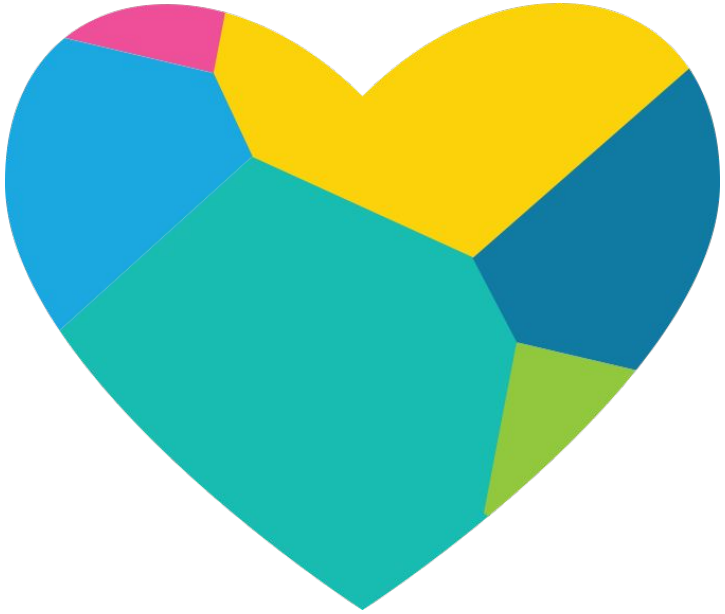
Works at Elastic

Responds to: Michael, Mick, Micky, Mike, Mikey, Crazybus, Rusty

Likes: Food, travelling, gaming

Stopping hitting yourself!

Disclaimer



A lot of this talk is going to come off pretty negative

I actually do like Kubernetes

I didn’t get paid to say that

I didn’t get paid at all actually

Restore

Dev

“Are you busy?” (yes)

Dev

“We need to restore the production database.” (Oh noes)

Me

“Do we have backups?”

Dev

“Yeah there is a bash script and a cronjob somewhere.”

Bash bashing


```
#!/bin/bash
# Author: Some dude
#
# Changelog
# 1995/05/23 - Some dude - Added mysql backups for important company database

mysqldump --all-databases > /mnt/backup/$(date +%s).sql
```

20 years later...


```
#!/bin/bash
# Author: Some dude
#
# Changelog
# 1995/05/23 - Some dude - Added mysql backups for important company database
# 1995/05/27 - Some dude - Added debug logging and log backup duration
# 1998/02/13 - New dude  - Create seperate dumps per backups
# 2010/09/03 - Ops dude  - Add set -e to make sure backup actually fails
# 2010/09/15 - Dev dude  - Compress backups to save space
```

Looks good to me!


```
# important don't remove!!!
set -e

echo “Started backup”
start=`date +%s`
timestamp=$(date+%s)

for table in $(echo "use jobsite_db; show tables;" | mysql -uroot -psupersecretpassword); do
    echo "Backup up table: ${table}"
    mysqldump jobsite_db ${table} | gzip > /mnt/backup/${timestamp}/${table}.sql.gz
    echo "Finished backing up table: ${table}"
done

end=`date +%s`
runtime=$((end-start))
echo "Backup successful!!! Finished in ${runtime} seconds!"
```

This is fine

No table locking

Need to restore the tables in the reverse order

Need to write a restore script

Only alerts when something went wrong

Things that email are good at

1

Humans sending a message to another human

who already knows the original human

2

Saving articles for yourself that you plan on reading later

Things that email are not so good at

- Monitoring alerts
- Computers contacting humans
- Alerting the fire brigade

Wow, it actually worked

Restore worked perfectly first go

Site started up

Site actually worked

‘Worked’ is a loaded term

The exit code of my restore script was 0

Stuff was on the website

Stuff was in the database

There were actually only 8 jobs online

OK, let's go back in time

git blame: "initial commit from svn"

"Maybe it was just a **bad backup**? Let's look at yesterday's backup."

Hmm...


```
$ du -hs /mnt/backups/*/jobs.tar.gz
1.0M    1539761034/jobs.tar.gz
500.3M  1539674634/jobs.tar.gz
183.5M  1539588234/jobs.tar.gz
302.3M  1539501834/jobs.tar.gz
10.7M   1539415434/jobs.tar.gz
485.9M  1539329034/jobs.tar.gz
56.3M   1539242634/jobs.tar.gz
152.3M  1539156234/jobs.tar.gz
501.3M  1539069834/jobs.tar.gz
```

Let’s take a closer look


```
# important don't remove!!!
set -e

echo “Started backup”
start=`date +%s`
timestamp=$(date+%s)

for table in $(echo "use jobsite_db; show tables;" | mysql -uroot -psupersecretpassword); do
    echo "Backup up table: ${table}"
    mysqldump jobsite_db ${table} | gzip > /mnt/backup/${timestamp}/${table}.sql.gz
    echo "Finished backing up table: ${table}"
done

end=`date +%s`
runtime=$((end-start))
echo "Backup successful!!! Finished in ${runtime} seconds!"
```

Changelog


```
#!/bin/bash
# Author: Some dude
#
# Changelog
# 1995/05/23 - Some dude - Added mysql backups for important company database
# 1995/05/27 - Some dude - Added debug logging and log backup duration
# 1998/02/13 - New dude  - Create seperate dumps per backups
# 2010/09/03 - Ops dude  - Add set -e to make sure backup actually fails
# 2010/09/15 - Dev dude  - Compress backups to save space
```

But, you were supposed to help me?

It's OK if every part of the script failed except for the last line `default`

Do you care if any of them failed or just the last one?

```
set -e
```

OK, but do you care if parts of each line failed or just the last one?

```
set -o pipefail
```

> **Software is easy to use**
>
> **but hard to run.**
>
> - Me again

> **Data is easy to backup,**
>
> **but hard to restore.**
>
> - Always me

Times have changed

OK, this was 1995...

And it was bash

So things have improved right?

Hello Docker


```
$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

> **The Docker daemon pulled the "hello-world" image from the Docker Hub.**
>
> - Docker daemon (in haiku)

What actually happened

The docker daemon checked if the image `hello world`

with the tag `latest` was already downloaded

It already existed, so it actually did **nothing**

and didn't connect to Docker hub

What didn’t happen

The Docker Daemon didn’t check that this was the right image from the Docker Hub

It didn’t download `docker.io/library/hello-world`

It didn’t download the latest `latest`

Nit picking?

`latest` is the default tag for the client -

it doesn't have to be **latest**, or **updated** or even **exist** in the registry

If you already have some version of this tag it won't pull a new one

Your `latest` may not be my `latest` or even exist on Docker Hub anymore

Down the rabbit hole

- 3 servers, 3 rabbitmq nodes running in Docker with image “rabbitmq”
- All running 2.8.7 at the time
- Updates happened to the rabbitmq docker image
- No updates happened to our setup
- New server was added to the pool
- Yay, we just upgraded to version 3.0.0!

Why haven't you patched prod?


```
FROM centos:6.6
COPY . /app
```

Ahh we forgot to pull!


```
# Need to pull first to get “real” latest
# https://github.com/moby/moby/issues/13331
docker pull centos:6.6
docker build -t business-app .
```

Nope, still no

Minor tags

Additionally, images with minor version tags that correspond to install media are also offered. **These images DO NOT recieve updates** as they are intended to match installation iso contents. If you choose to use these images it is highly recommended that you include `RUN yum -y update && yum clean all` in your Dockerfile, or otherwise address any potential security concerns. To use these images, please specify the minor version tag:

For example: `docker pull centos:5.11` or `docker pull centos:6.6`

We did it!


```
FROM centos:6.6
RUN yum -y update && yum clean all
COPY . /app
```

1 month later

"Why haven't you patched prod?"

Hello Docker build cache!

The cache for `RUN` instructions isn't invalidated automatically during the next build. The cache for an instruction like `RUN apt-get dist-upgrade -y` will be reused during the next build. The cache for `RUN` instructions can be invalidated by using the `--no-cache` flag, for example `docker build --no-cache` .

Ahh we forgot to disable cache!


```
# Need to pull first to get “real” latest
# https://github.com/moby/moby/issues/13331
docker pull centos:6.6

# --no-cache is needed to make sure we actually patch
# https://docs.docker.com/engine/reference/builder/#run
docker build --no-cache -t business-app .
```

Docs check

The docs actually show the output from an old `hello-world` image

I like to think that this is because they haven't done a `docker pull hello-world` in a while (this is not true though)

1. Test that your installation works by running the simple Docker image, hello-world:

```
docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:ca0eeb6fb05351dfc8759c20733c91def84cb8007aa89a5bf606bc8b315b9fc7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Updating Images

The default pull policy is `IfNotPresent` which causes the Kubelet to skip pulling an image if it already exists. If you would like to always force a pull, you can do one of the following:

- set the `imagePullPolicy` of the container to `Always`.
- omit the `imagePullPolicy` and use `:latest` as the tag for the image to use.
- omit the `imagePullPolicy` and the tag for the image to use.
- enable the AlwaysPullImages admission controller.

Note that you should avoid using `:latest` tag, see Best Practices for Configuration for more information.

Hello Kubernetes

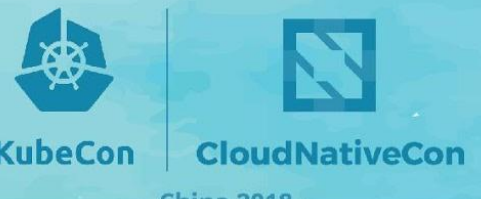

```
kubectl run nginx --image=nginx
kubectl expose deployment nginx --port=80 --type=LoadBalancer
```

Show me what happened!


```
$ kubectl get deploy nginx -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: kubecon
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

```
template:
  spec:
    containers:
    - image: nginx
      imagePullPolicy: Always
      name: nginx
      resources: {}
    restartPolicy: Always
```

Some important questions



Is it OK if all containers are running on the same host?

Is it OK if all containers go down and fail to start again during upgrades?

Should these containers always run? Or just sometimes?

Should I check if the container is healthy?

You asked for 3 replicas. Do you really want that many or do you just like that number?

Update Strategy


```yaml
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

What would you expect?

I deploy a new version

The new version is deployed

Once it is ready and healthy, the old one is removed

What actually happens

Deploy new version

New version is deployed

At the same time old version is removed

New version fails

Application goes down

What you probably want


```yaml
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
```

What you really really want


```yaml
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
```

My application is invincible!

Kubernetes will restart it if it crashes

“When deploying a new version I can’t break anything!”

Kubernetes upgrade time!

Kubernetes cluster is upgraded

Rolling restart of nodes

Application fails to start

Kubernetes doesn't care

and continues to take down all 3 replicas

You forgot pod disruption budgets!


```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: nginx
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nginx
```

Kubernetes node crashes

Kubernetes node crashes

All 3 replicas were running on that node

Kubernetes waits 5 minutes

Kubernetes starts the 3 replicas again

You forgot pod anti-affinity!


```yaml
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - nginx
        topologyKey: kubernetes.io/hostname
```

Kubernetes is happy now

Application is running great

Other teams are jealous

They too want the Kubernetes

They deploy application with this `resources` thing

Your application goes down

What happened

No resource requests or limit were being set

All containers were “best effort”

Every container got an equal share of the available resources

New application was deployed with resource requests and limits

It decided to use them

No resources leftover for your application

You forgot resources!


```
resources:
   limits:
     cpu: 1000m
     memory: 500Mi
   requests:
     cpu: 1000m
     memory: 500Mi
```

Oh, and maybe quotas and defaults

Resource Quotas

https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/

Default resource limits

https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/cpu-default-namespace/

https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/memory-default-namespace/

Something is wrong with my app

My code has a problem

One container isn't feeling so good

Requests keep getting sent

even though it is clearly broken

You forgot probes!


```yaml
readinessProbe:
  httpGet:
    path: /health
    port: 8080
```

```yaml
livenessProbe:
  httpGet:
    path: /health
    port: 8080
```

Production ready checklist (save this)

- Resource Requests/Limits and Quotas
- Liveness and Readiness Probes
- Pod disruption budgets
- Anti-affinity rules
- Upgrade strategy
- Logging, Monitoring and Alerting
- A way to restore your application if Kubernetes gets deleted
  - Version controlled deployments
  - Secret persistence

Why?

First impressions matter

Convenience is king

Not everyone cares about HA and performance

Backwards compatibility

Can we make things better?

- Simpler documentation with ‘production ready’ examples?
- Stricter defaults that prioritise safety over compatibility?
- ‘Production mode’ where you are forced to always set these values?
- Include these features in the default helm charts templates?

> **Deploying to production is easy.**
>
> **Running tests locally is hard.**
>
> - Everyone (including me)

Lope this way

https://github.com/Crazybus/lope

- Automatically mount current working directory
- Forward environment variables
- Forward your SSH agent into the container
- Forward docker socket and automatically add a docker client

KubeCon | CloudNativeCon

China 2018