DOCKER ATHENS USER GROUP

Thursday 11 April 2019

About Docker Athens

- Started in May 2014
- ~2’000 members!
- 29 meetups already!

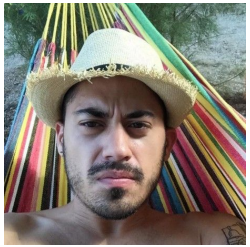
Organisers

Antonis Kalipetis

Co-founder at SourceLair
Docker Captain
Docker Certified Associate

Paris Kasidiaris

Co-founder at SourceLair



Sponsor: SourceLair Private Company

https://lair.io

Online IDE for web developers.

https://2hog.codes

Workshops on Docker, JavaScript and more.

Come and speak!

Attendees would love to hear you speaking.

Please reach out.

https://docker.gr

Let’s get it started.

Stateful applications and principles

Docker Athens — Thursday 11 April 2019

State in Computer Science

- We call **state** all information, that need to be retrieved and processed by an application, in order to function appropriately.

- There are many options for storing state: RAM, block storage or cold storage.

Stateful software

- We call stateful software applications that rely on stored state in order to function appropriately
- In most cases stateful software is called the one relying on state stored on a block storage device (e.g. SSD disk)
- Typical examples of stateful software are the databases we use (e.g. Postgres, MongoDB etc.)

Maintaining stateful software in production

- High Availability
- Data redundancy
- Performance

Scaling stateful software

- Multiple factors (e.g. CPU, RAM, network and disk I/O)
- Implementation-specific topology (e.g. Postgres cluster and MongoDB Replica Set)
- Client-side implementation (e.g. read-only replicas and read/write masters)

Containerizing stateful software

- Run the stateful application in process isolation (**not** virtualization!)
- Use an overlay or user-space network connection
- Access selective host devices (e.g. mount a dedicated block storage device)

Mythbusting time!

Myth #1

I can just use *“docker run postgres”*

Myth #2

I can just use

“docker service scale pg=3”

Myth #3

Containers will help my database auto scale.

Myth #3

Containers will help my database auto scale. (lol)

Distributing containerized stateful software is hard

- **Hard to move between hosts**: Stateful software needs access to particular hardware (block devices for storage), pretty much pinning each instance to a single node.
- **Cannot use “native” scaling parameters**: Stateful clusters need to access each instance via a unique hostname (e.g. *pg-replica-03)*, so *docker service scale postgres=4* won’t work at all.
- **Cannot always use native container networking**: Container clusters use NATred networks, which can prevent some software to work at all (e.g. Redis Cluster).

Let’s get some demographics!

Who of you uses a managed database (e.g. Amazon RDS)?

Who of you manage their own database servers?

Who of you run their databases in containers?

Who of you manage database clusters (master-replicas topology)?

Which are your most unpleasant database management challenges?

Database challenges get worse in containers

- Container network issues can affect your production database
- Misconfigured resource limits can deeply affect database performance
- Debugging gets way more challenging in containers

So, why would anyone containerize their database?

- Use a single management plane for all your services.
- Cost reduction compared to completely managed solutions.
- Fast, straightforward, predictable, “undoable” upgrades.

Use a managed database (e.g. Amazon RDS), if you can afford it.

Let’s take a (coffee) break!

Deploying a stateful application to Docker Swarm

Docker Athens — Thursday 11 April 2019

We will playback our story

SourceLair 2019 cloud provider migration

- We migrated literally **everything** from one provider to another
- We migrated our deployments from Upstart and Supervisord to Docker Swarm services
- We migrated our stateful MongoDB Replica Set, Postgres and Redis servers

Let’s get our hands dirty.

Step #1: MongoDB in a container

1. Find the appropriate Docker Image
2. Determine configuration
3. Write Docker Compose file

Let’s go!

Step #2: Persisting storage

1. Pick a Docker Volume driver

2. Add a Docker Volume in the Compose file

Let’s go!

Step #3: Deploy MongoDB in Docker Swarm

1. Deployment configuration
2. Secrets
3. Common network

Let’s go!

Step #4: MongoDB Replica Set on Docker Swarm

1. Introduce multiple services for data nodes and arbiters
2. Avoid code and configuration duplication
3. Deployment options

Let’s go!

Step #5: Remove constraints (?)

It **would** be amazing, if there was a straightforward way to:

- Attach a block storage device from our cloud provider to a Docker Swarm service
- Consequently remove placement constraints
  (Each service would get the appropriate disk, regardless of the Docker Swarm Node scheduled)

Step #5: Remove constraints (?)

The tools we **could** use at some point are

- Docker Volume Plugins: https://hub.docker.com/search/?type=plugin&category=volume
- Container Storage Interface (CSI): https://github.com/container-storage-interface/spec
- Using CSI in Docker (and Volume Plugins): https://github.com/moby/moby/issues/31923

Before closing...

Kelsey Hightower
@kelseyhightower

Some people believe that rubbing Kubernetes on a stateful workload turns it into a fully managed database offering rivaling RDS. This is false. Maybe with enough effort, and additional components, and an SRE team, you can build RDS on top of Kubernetes.

12:15 AM - 24 Mar 2019

Thank you!

Questions?