A Docker Swarm Hands-on

Ville-Veikko Valtonen

Contents

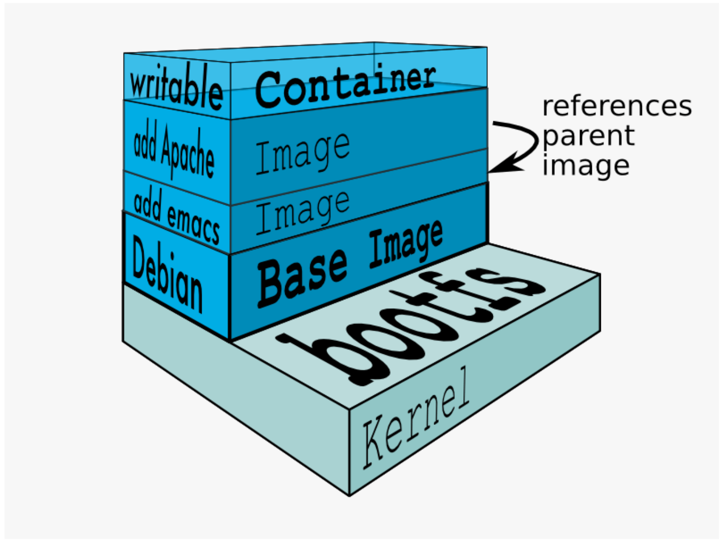
Docker: Container

- Operating system –level virtualization, LXC
- Limited resources
- Sandbox –like operation
- Kernel dependent
- Managed by Docker process
- Single process
- **Not a method of added security!**

digia

Docker: Image

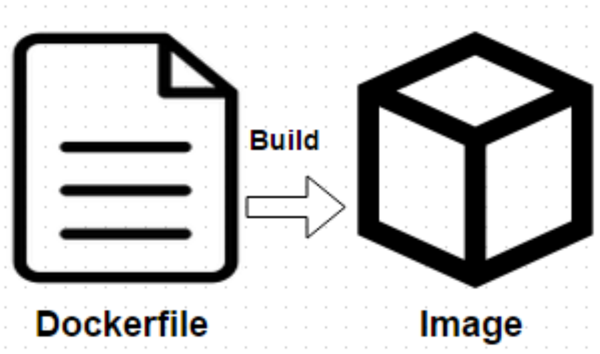
- Containers are built from images
- Images consist of layers, that are also images
  - Operating system
  - Installations 1 .. N
  - Customization and configuration
- Built with recipes called Dockerfiles
- Tagged to tell where, what and which version



digia

Docker: Dockerfile

- File named ‘Dockerfile’
- Contains commands to create an image
- Usually extends another image
- Common commands
  - FROM: Which image to base on
  - RUN: Run commands
  - COPY: Copy files, e.g. configuration
  - CMD: What command should start the **single process**



Docker: Registry

- Place to store Docker images
- Open to everyone
  - hub.docker.com
- Open Source, private
  - Docker registry container
  - Nexus 3

Nexus

artifactory

DOCKERHUB

digia

Docker: Volume

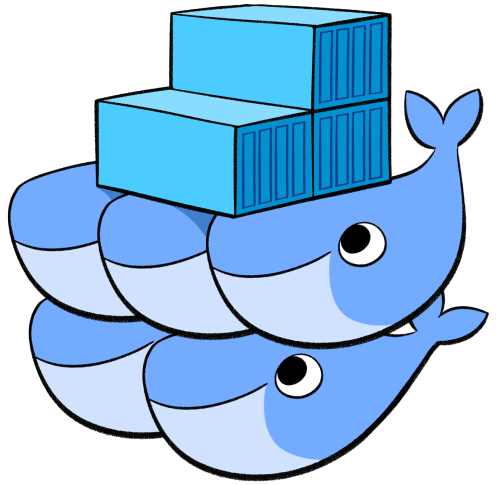
- Docker images are by philosophy, immutable
- Volumes provide current state
- Volumes can be used for data persistence
- Volumes are linked to Docker containers
  - -v
- Also consider data containers
  - File permission and structure advantages

Docker: Network

- Containers can access host network
- Containers access each other via Docker networking
  - Default: All containers belong to bridge
- Containers must publish ports to be available outside the Docker network
  - -p external_port:container_port
- New networks can be defined
  - Docker swarm automatic DNS requires an explicit overlay network!

digia

Docker Swarm: Swarm



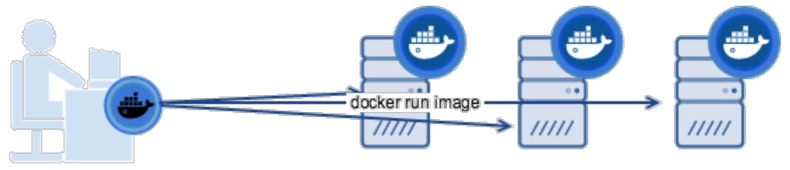
- A container orchestration tool
- Consists of servers, nodes
- Manages desired state of services
- Abstracts the concept of server to a pool of resources
- Networks span the whole swarm
- Provides common tools for updates, maintenance, etc

digia

Docker Swarm: Node

- A single server joined into a Swarm
- Manager, Worker or both
- Managers check and manipulate Swarm state
  - Swarm commands are only ran on Manager nodes
- Worker nodes host containers
- High availability poses constraints on number of nodes
  - https://docs.docker.com/swarm/multi-manager-setup/

Docker Swarm: Service



- Definition of content managed by Swarm
- Defined similarly to a single container
- Discoverable by it’s name
  - Hides actual location of containers
- Scalable
  - 2 logstashes today, 20 tomorrow
- Stateful or **Stateless**

digia

Docker Swarm: Tasks

docker run image

- Instances of work to be done by a service
- Docker containers on Worker nodes
- Inherit parameters from the service
- Produce actual capability from the Swarm

Docker in Action: Running a container

- Docker-toolbox for Windows and OS X, native for Linux
  - https://www.docker.com/products/docker-toolbox
  - https://docs.docker.com/engine/installation/linux/
- docker run -d --name mongo -p 91:27017 -p 92:28017 mongo:3 mongod --rest
  - run is the base command, start a container
  - -d, detach instead of just running until closed by user
  - --name, name of the container, otherwise random
  - -p external_port:container_port, exposing ports so external services can use it
  - mongo, the name of the image, :3 the tag or version
  - mongod –rest, command we wish to run within the container

Docker in Action: Building a container

- Dockerfile and needed files in a folder
- docker build . --tag customimage:2
  - build, build a container
  - --tag, give our image a name customimage and version 2
- Often built automatically by CI
  - In our hands-on, by docker hub
- Can be copied into a repository with docker push
  - https://docs.docker.com/engine/reference/commandline/push/

Docker in Action: Github + Docker hub

- https://github.com/
  - Awesome repository
- https://hub.docker.com/
  - Largest source of docker containers
  - Open to everyone
- Demo

Docker Swarm in action: Setting up

- Docker 1.12 or newer needed
- First manager node
  - docker swarm init --advertise-addr <MANAGER-IP>
  - Outputs the command to run on worker nodes
- Ports required
  - 2377, 7946, 4789

Docker Swarm in action: Service

- Run on any manager node
- docker service **create** *--replicas 1 --name stash logstash*
  - service create, deploy a service to the Swarm
  - --replicas 1, run only one task on the collection of workers
  - --name, name our service stash
  - logstash, the name of our container to base the service on
    - Omitting the tag will default to :latest

Docker Swarm in action: Examining

- List services
  - docker service ls
- Examine a service
  - docker service ps <servicename>
- Examine swarm
  - docker info
  - docker node ls

Docker Swarm in action: Updating

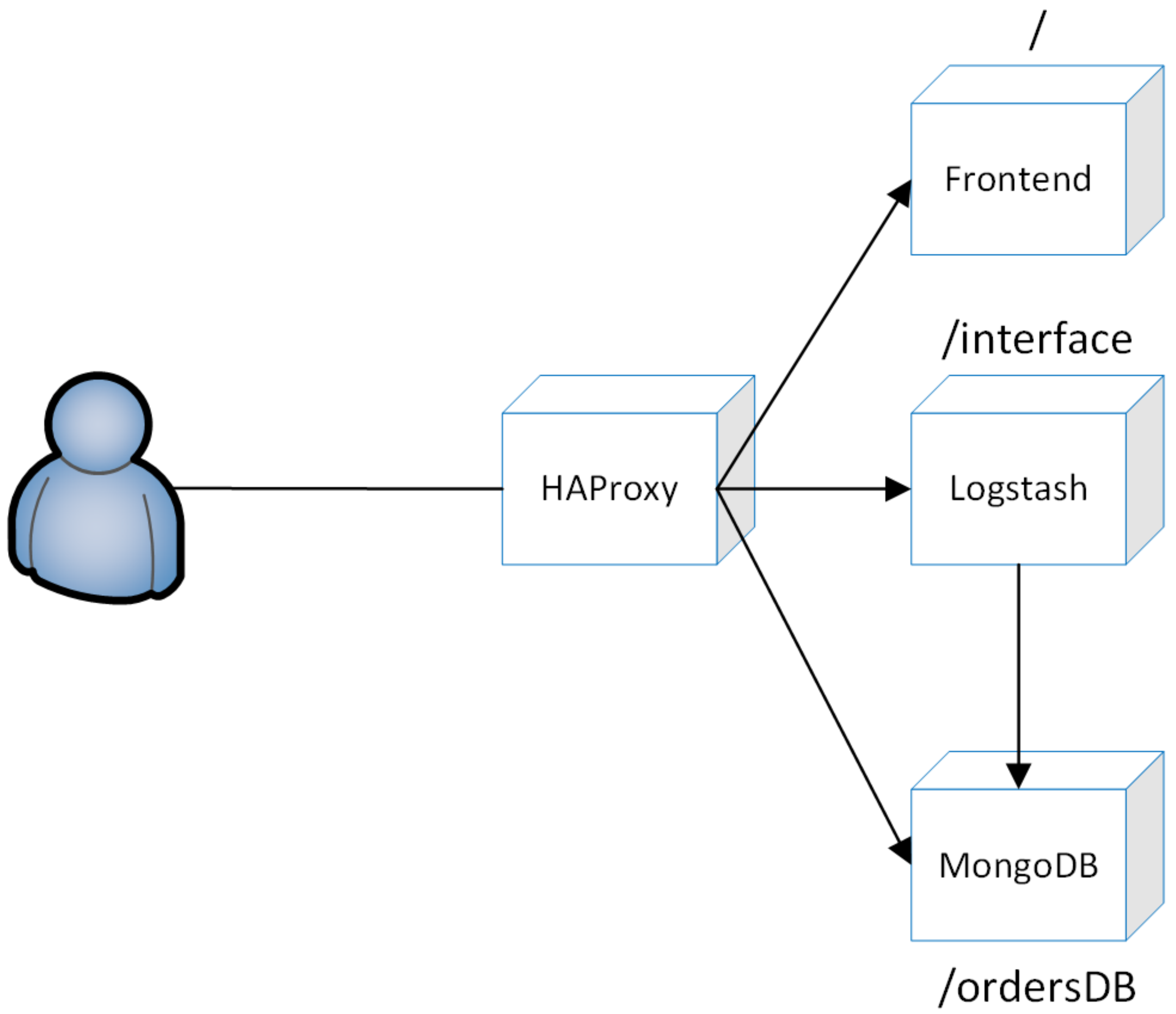
- Run on any manager node
- docker service **update** *--image redis:3.0.7 redis*
  - service update, update some parameter of a service
  - --image, change the image to redis, tag 3.0.7
  - redis, the name of the service to update

Exercise: Your first swarm application

- A microservices oriented application running on docker swarm
- Showcases what it is like to build a microservices application
  - Focus on planning, understanding and configuring components
  - Modularized design, isolated development
  - Step by step building

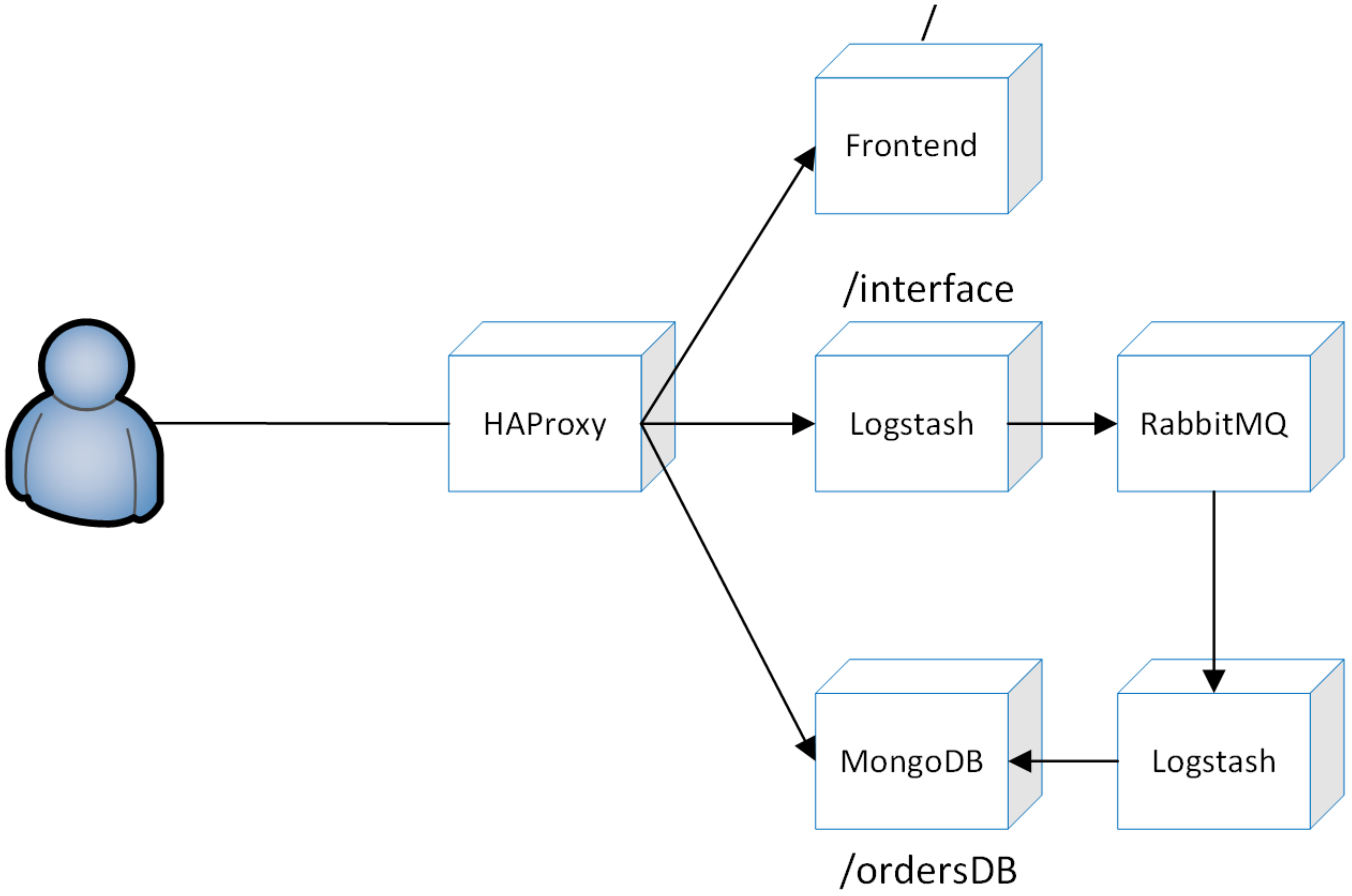
Exercise: Your first swarm application

- Version 1: Minimum viable product
- Javascript frontend
  - Connects to Mongo rest to fetch orders
  - Sends orders to Logstash
- HAProxy router
  - Routes calls based on url to frontend, logstash or mongo
- Logstash adapter
  - Transforms http post from frontend to mongodb insert
- Mongo database
  - Receives inserts
  - Provides REST api for frontend to fetch from


/
Frontend
/interface
HAProxy
Logstash
MongoDB
/ordersDB

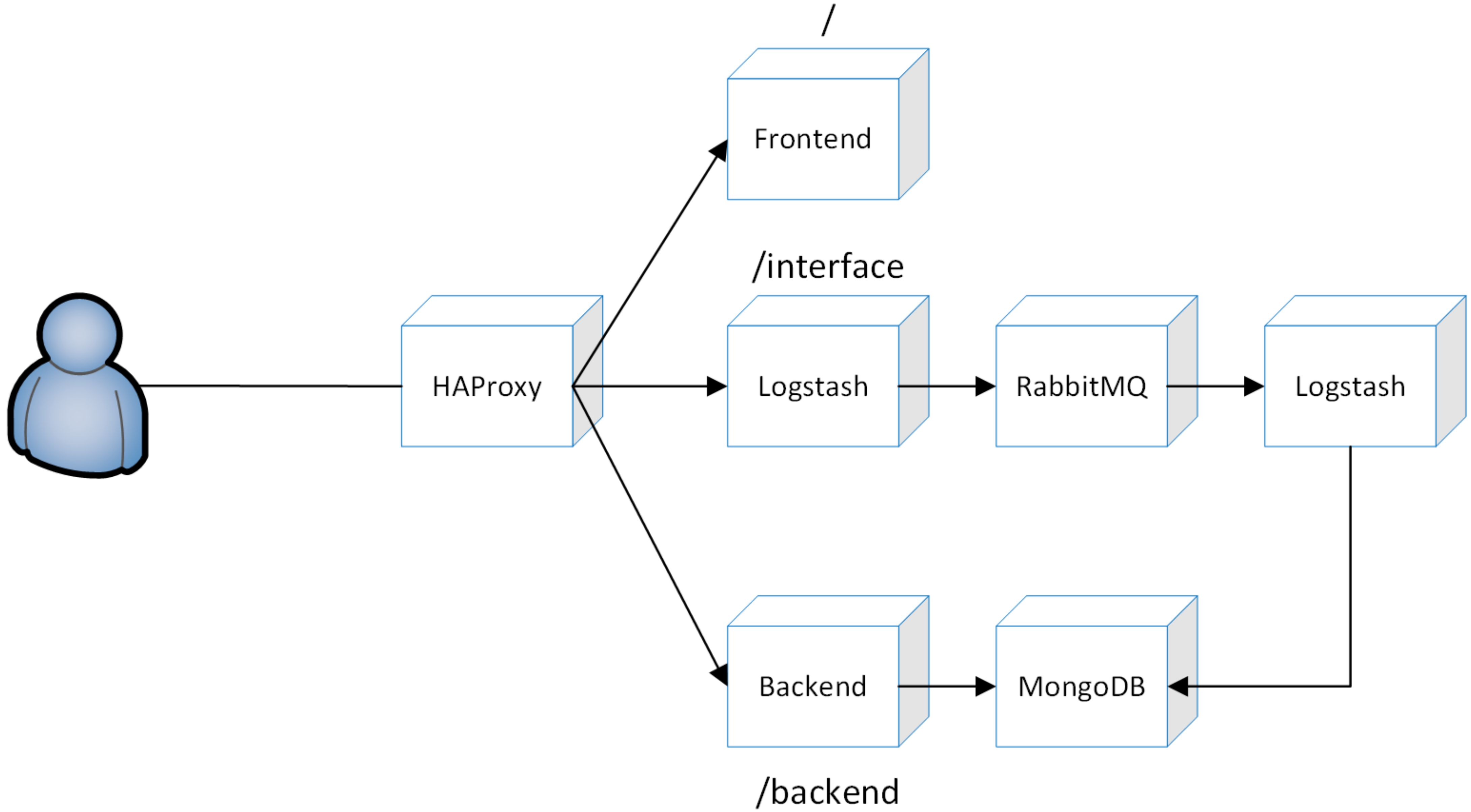
Exercise: Your first swarm application

- Version 2: Preparing for load
- Javascript frontend
- HAProxy router
- 2 Logstash adapters
  - One receives HTTP post and inputs into RabbitMQ
  - One reads from RabbitMQ and inputs into MongoDB
- RabbitMQ queue
  - Persists messages in case of high load
- Mongo database


/
Frontend
/interface
HAProxy
Logstash
RabbitMQ
MongoDB
Logstash
/ordersDB

Exercise: Your first swarm application

- Version 3: Advanced features
- Javascript frontend
- HAProxy router
- 2 Logstash adapters
- RabbitMQ queue
- Mongo database
- Node backend
  - Performs custom logic instead of direct call from frontend to MongoDB
  - Use MongoJS or other Mongo driver instead of REST


/
Frontend
/interface
HAProxy
Logstash
RabbitMQ
Logstash
Backend
MongoDB
/backend

Optional challenges

- Node-red API enrichment before MongoDB save
- Persistent storage for MongoDB
- RabbitMQ clustering
- Persistent storage

Documentation links

- Container examples
  - https://hub.docker.com/r/villevaltonen/
- Container pages
  - https://hub.docker.com/_/httpd/
  - https://hub.docker.com/_/haproxy/
  - https://hub.docker.com/_/logstash/
  - https://hub.docker.com/_/mongo/
  - https://hub.docker.com/_/rabbitmq/
  - https://hub.docker.com/_/node/

Documentation links

- Configuration documentation
- HAProxy
  - https://cbonte.github.io/haproxy-dconv/
- Logstash
  - https://www.elastic.co/guide/en/logstash/2.4/index.html
- Usage documentation
- MongoDB
  - https://docs.mongodb.com/manual/
  - http://www.rabbitmq.com/documentation.html
- Searching google for **examples** is very helpful!
  - E.g. ‘logstash http input example’

Laitamme itsemme likoon.

Ryhdytäänkö töihin?

digia