How to run

Setting up the environments requires virtualbox, vagrant and ansible installed. Run `vagrant up` in the `training_environment` directory to bring up the training environment. Once started, the application can be accessed at:

- http://localhost:8080 (training environment - application)
- http://localhost:8080/manager/html/ (credentials are tomcat_admin:tomcat)

Run `vagrant up` in the directory `production_environment`, to bring up the following:

- http://localhost:8081 (production environment - balancer1)
- http://localhost:8082 (production environment - balancer2)
- http://localhost:8081/haproxy?stats (haproxy_stats_admin:iePha4se)

Architecture

All systems are based on the vagrant image `bento/debian-8.5`, for more information visit https://atlas.hashicorp.com/bento/boxes/debian-8.5. To serve the war file in each environment, I choose tomcat because it is an apache project. In the training environment tomcat also serves the static files, so it is literally the only service defined there. In production the static files are served via gatling (http://www.fefe.de/gatling/), because it does the trick with least complexity; it could be replaced when the system complexity grows.

Ram

The tomcat servers were configured to launch jvms with 64mb of ram (due to percepted memory usage and to make development easier). Also all vagrant boxes are configured to 256mb of ram in the production environment (128mb also seemed to work). In a real-world production setup one would surely increase the ram (prevayler requires to store the full application state in ram).

Production environment

The production setup looks roughly as follows:

```
Internet --> load balancers --> appservers / staticservers
                [haproxy]        [tomcat8] / [gatling]
                       |            |           |
                       v            v           v
                         logservers   [rsyslog]
```

The haproxy load balancers route an incoming request to the appropriate backend (appservers/static) based on the request url. The internet-facing port(s) of the environment are forwarded to the vagrant host's ports.

The hosts in the provided setup are (all ports `tcp`):

- balancer1 (ip: 10.0.0.10, port 80 forwarded to vagrant host port 8081)
- balancer2 (ip: 10.0.0.11, port 80 forwarded to vagrant host port 8082)
- appserver1 (ip: 10.0.0.20, serves the application at port 8080)
- appserver2 (ip: 10.0.0.21, serves the application at port 8080)
- static1 (ip: 10.0.0.30, serves static content at port 8090)
- static1 (ip: 10.0.0.31, serves static content at port 8090)
- logserver1 (ip: 10.0.0.100, receives logs from balancers, appservers and statics at port: 514)
- logserver2 (ip: 10.0.0.101, receives logs from balancers, appservers and statics at port: 514)

The configurations are generated to provide one-node fault-tolerance for the whole system.

All internal IP adresses used in the production setup are defined in `production-config.yml`. The variables defined in `production-config.yml` are used in ansible as well as vagrant and provide a single point of configuration for adding more machines to the environment.

Adding new hosts:

1. Add section to `production-config.yml`
2. Vagrant up the machine
3. If an application or static server was added, it is not yet included in the load balancer configs. Reprovision them with `vagrant provision balancerN` to deploy a new version of the haproxy config and restart haproxy.

Discussion of the application

The current implementation (Prevayler) does not provide a way to synchronize application data between network nodes. This makes one-node fault-tolerance and keeping tight SLAs very hard. I built a setup, that runs two appserver nodes, and their state diverges, once the first "news" is entered. Synchronizing the prevayler data directory nodes does not trigger updates in "the other appserver". Thus, I would recommend the dev-team to move away from prevayler, towards a distributed database, that fits the data flow of the application. Also, not having an authentication system for content creation in place, seems like a bad idea to me. The war file also seems to statically reference the directory `/Users/dcameron/persistence/files*` as the persistence directory. I would advise to make that configurable, for example through an environment variable.

Discussion of the environment

- service configurations are all very basic
- no encryption configured (https / deployment keys / ssl tunnels for logs)
- no backup of application data / disaster recovery plan
- vagrant adds support for other virtualization providers as well
- the ansible playbooks could even be used to deploy to platforms not supported by vagrant

Limited Release

A hosting provider must be chosen from the market, and the scripts be adapted to set up the stack there. An alerting system should be installed. I would probably suggest using a cloud provider like uptimerobot (http://uptimerobot.com). Ideally the most common administrative processes (restarting servers/applications) should be automated. The system should also have a resource monitoring solution in place to study bottlenecks and load patterns in the limited release phase.

Global Release

World-wide sub-second response-times require a distribution network around the world. For static content a CDN can be used, the application logic would be the part to focus on.