Evolution of a Microservice Infrastructure

Paul Puschmann
OSAD 2019, Munich
What do we actually run?
Recap
The state of 2018

We’re operating a custom Docker-Environment consisting of:
Recap
The state of 2018

We’re operating a custom Docker-Environment consisting of:

Everything was cool. Developers can bring Code live. All is well.
One repository for infrastructure-configuration

Ansible, Vagrant, Terraform, … executed via Jenkins.
... and works like this
for deployment of services

One central repository for service-deployments

- Used on every Team-Jenkins as external resource
- Teams provide a “service-descriptor.yaml” for each service
- “service-descriptor.yaml” gets updated with environment-specific variables
- containers get started with environment of “service-descriptor.yaml”
- standardised deployment is ensured
---

service_name: "example"
service_version: "1.2.1"
squad: "Example-Squad"
team: "Example-Team"
num_instances: 3
prometheus_enabled: "true"
prometheus_path: "/metrics/prometheus"

service_memory: 1536

service_configuration:
  JAVA_META_SIZE_TO_HEAP_QUOTA: 40

# Example DB

DATASOURCES_SHOP_JDBCURL: "jdbc:postgresql://{{ psql_cluster_master }}:5432/{{ db_name_example }}"
DATASOURCES_SHOP_USERNAME: "{{ db_user_example }}"

DATASOURCES_SHOP_PASSWORD: "{{ db_password_example }}"

...
Recap
The state of 2018

We’re operating a custom Docker-Environment consisting of:

Everything was cool. Developers can bring Code live. All is well.
All is fine
... and looks like

Ingress-Nodes

Docker-Hosts
a.k.a.
Worker-Nodes

Consul-Server
a.k.a.
Master-Nodes

Other “aaS”

Customer
... and looks like

Ingress-Nodes

Docker-Hosts
a.k.a.
Worker-Nodes

Consul-Server
a.k.a.
Master-Nodes

Ingress-Nodes
- Nginx-config written by consul-template on change of Consul-information
- Routes external Hostnames

Customer

Other "aaS"
... and looks like

Ingress-Nodes

Docker-Hosts
a.k.a. Worker-Nodes

Docker-Host
- Nginx-config written by consul-template on change of Consul-information
- Routes internal Hostnames to containers
- Runs containers

Consul-Server
a.k.a. Master-Nodes

Other "aaS"
... and looks like

Consul-Server & Swarm-Master
- Contain knowledge of all services
- Deployments are started from here
- Act as DNS-Servers for service-discovery

Ingress-Nodes

Docker-Hosts a.k.a. Worker-Nodes

Consul-Server a.k.a Master-Nodes

Customer

Other “aaS”
... and looks like

Ingress-Nodes

Docker-Hosts
a.k.a.
Worker-Nodes

Consul-Server
a.k.a.
Master-Nodes

Other centrally managed "platform-services"
- Kafka
- Databases
- ELK-Stack
- Prometheus & Grafana
- ...

Other “aaS”
Request routing
how can services be addressed

- Both colors have the same DNS record
  - Consul will return IPs for all hosts where the Service is running
- Nginx running on each Worker Node
  - routes to colour depending on used port
Routing Problems
Problems with Nginx
increased with the size of the environment

- There are requests which never reached their destination
- Always happened at the time of deployments
... and looks like

Ingress-Nodes

Docker-Hosts
a.k.a.
Worker-Nodes

Consul-Server
a.k.a.
Master-Nodes

Other “aaS”
Problems with Nginx
increased with the size of the environment

- There are requests which never reached their destination
- Always happened at the time of deployments
- Consul-template would **reload all** Nginx instances **at the same time**
- What happens at a reload?
Problems with Nginx
looking for solutions

Look for different reverse proxy

● No reload on config change (optional)
● Dynamic configuration (optional)
● Robust connections to the client
Problems with Nginx
possible replacements

- envoy
- Fabio
- traefik
Traefik

- Dynamically configurable
- Live reloading of configuration
- Lots of metrics
- Nice web ui
- Single Go binary

Since Traefik 2.x:
- independent configuration of frontend & backend
  - mix consul service-discovery with file-based configuration
Traefik
1. Install alongside Nginx on Worker and Ingress Nodes
   ○ listen on different ports
2. Check that configured routes are correct and work
3. Change port mapping host by host -> Traefik is active
4. Remove Nginx
Traefik
how to migrate

some service

:80
:10080

docker-1

basket
Traefik
how to migrate

- some service:80 :10080
- docker-1
- basket
Traefik

how to migrate

some service

:80

docker-1

basket
Traefik
Benefits

- Keepalive and connection problems immediately went away
- Almost real time data about service response time
- Web UI to check routes
- Rich access logs
Benefits
Problems with standalone Swarm
also increased with increasing workload

- Poor container spread
  - all service instances running on one host
- No self healing
- Manual node draining (e.g. for maintenance)
  - we’re still dependent on docker-compose files
- Only few metrics
Swarm...

Ingress-Nodes

Docker-Hosts
a.k.a.
Worker-Nodes

Consul-Server
&
Docker-Swarm master

Other "aaS"
We want this

- self healing
- proper container spread
- metrics
- resource limits (optional)
- stateless docker-host
Possible replacements

- Rancher
- Nomad
- Docker Swarm
- Kubernetes
Nomad

- Seamless Consul integration
  - almost no setup needed
- Self healing
- Bin packing
- Single Go binary
- Nice Web UI
- (Memory) Limits enforced by default
- Token-based ACL

HashiCorp

Nomad
Nomad
Benefits

● Not limited to Docker
  ○ Rkt and LXC

● Not limited to Containers
  ○ Jar files
  ○ Binaries
  ○ VMs
Swarm...

Ingress-Nodes

Docker-Hosts
a.k.a.
Nomad-Clients

Consul-Server
&
Nomad-Server

Other “aaS”

Customer
Nomad

Benefits

Ram-bandwidth-6

Type: service | Priority: 154

Allocation Status: 4

Task Groups

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Alloc</th>
<th>Running</th>
<th>Complete</th>
<th>Queued</th>
<th>Starting</th>
<th>Reserved CPU</th>
<th>Reserved Memory</th>
<th>Reserved Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>array-g-23</td>
<td>3</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6500 MHz</td>
<td>14336 MB</td>
<td>0 MB</td>
</tr>
<tr>
<td>bun-g-21</td>
<td>2</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1200 MHz</td>
<td>3072 MB</td>
<td>0 MB</td>
</tr>
<tr>
<td>feed-g-24</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2500 MHz</td>
<td>2048 MB</td>
<td>0 MB</td>
</tr>
<tr>
<td>port-g-22</td>
<td>4</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6750 MHz</td>
<td>16896 MB</td>
<td>0 MB</td>
</tr>
</tbody>
</table>
Nomad
Benefits - Cluster Level
Nomad

Benefits - Service Level
### Nomad Benefits

#### Containers - Blue

<table>
<thead>
<tr>
<th>ID</th>
<th>Started</th>
<th>Server</th>
<th>Image</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>72edc7ed7b</td>
<td>2019-04-29 13:48:45</td>
<td>mc-siteline:02a</td>
<td>00:00:00:blue_12</td>
<td>Running</td>
</tr>
<tr>
<td>171:31732</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Task Allocation Logs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Logs by Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Metrics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Memory Allowance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Disk Allowance</td>
<td></td>
</tr>
<tr>
<td>d24dc586</td>
<td>2019-04-29 13:48:45</td>
<td>master_siteline:02a</td>
<td>00:00:00:blue_12</td>
<td>Running</td>
</tr>
</tbody>
</table>

#### Containers - Green

<table>
<thead>
<tr>
<th>ID</th>
<th>Started</th>
<th>Server</th>
<th>Image</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>3a3fc3d3d</td>
<td>2019-04-29 16:58:18</td>
<td>mc-siteline:02a</td>
<td>00:00:00:green_13</td>
<td>Running</td>
</tr>
<tr>
<td>65032a5f3f</td>
<td>2019-04-29 16:58:13</td>
<td>master_siteline:02a</td>
<td>00:00:00:green_13</td>
<td>Running</td>
</tr>
</tbody>
</table>
State of 2019

We’re operating a custom Docker-Environment consisting of:
State of 2019
And we’re also using

kafka
HashiCorp
Terraform
What we Learned
What helped us most?

- Having a centralised deployment-toolset
  - perform all changes for all teams / developers at the same time
- Do Canary-like changes on our infrastructure
  - fully interoperable changes
  - Nginx <-> Traefik
What did we learn?

- Distributed systems can be hard
- Keeping your architecture pluggable helps a lot
- Computing resources can be finite
  - Enforcing resource limits can be interesting
- You might not need Kubernetes...
Evolution of a Microservice Infrastructure
OSAD 2019, Munich

Thank You!

Paul Puschmann  @ppuschmann

www.rewe-digital.com  @rewedigitaltech

All background photos are licensed under CCO. Source: pexels.com