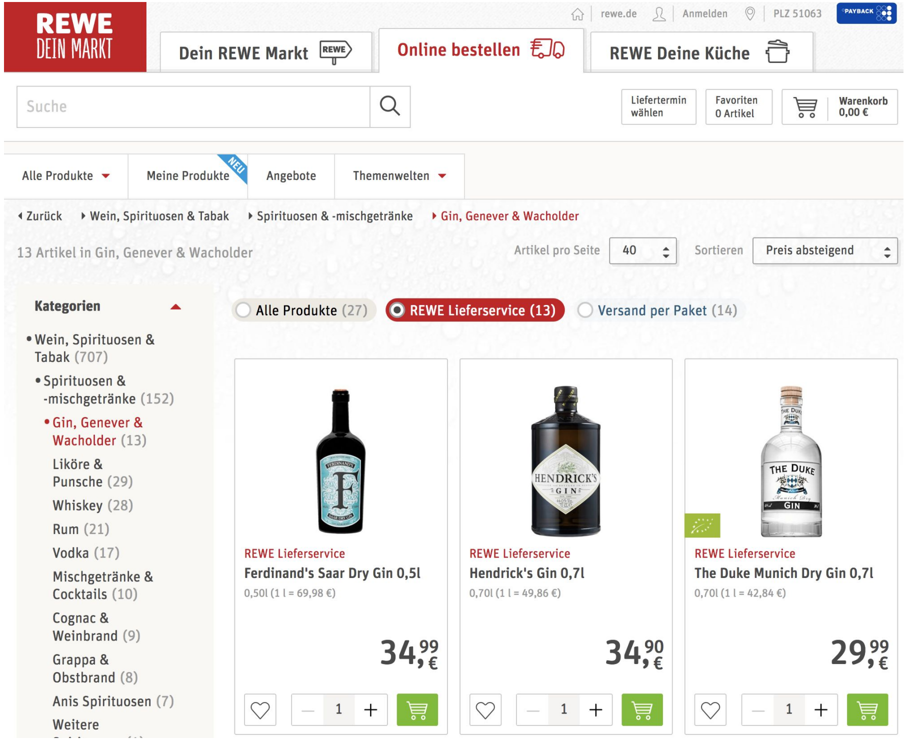
Evolution of a Microservice Infrastructure

Paul Puschmann
OSAD 2019, Munich

REWE digital

What do we actually run?



Zurück › Wein, Spirituosen & Tabak › Spirituosen & -mischgetränke › Gin, Genever & Wacholder

13 Artikel in Gin, Genever & Wacholder

Artikel pro Seite: 40

Sortieren: Preis absteigend

Kategorien

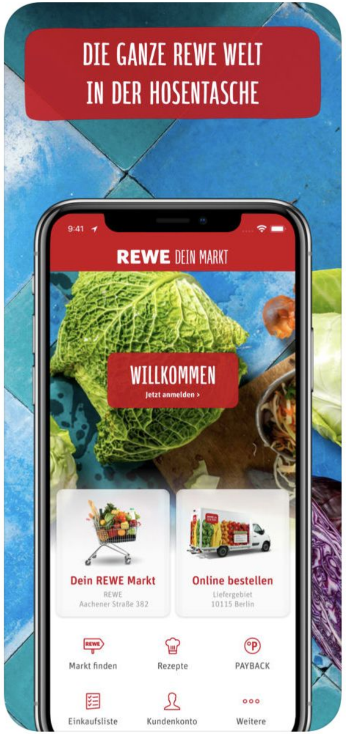
- Wein, Spirituosen & Tabak (707)
  - Spirituosen & -mischgetränke (152)
    - Gin, Genever & Wacholder (13)
    - Liköre & Punsche (29)
    - Whiskey (28)
    - Rum (21)
    - Vodka (17)
    - Mischgetränke & Cocktails (10)
    - Cognac & Weinbrand (9)
    - Grappa & Obstbrand (8)
    - Anis Spirituosen (7)
    - Weitere

○ Alle Produkte (27) ● REWE Lieferservice (13) ○ Versand per Paket (14)

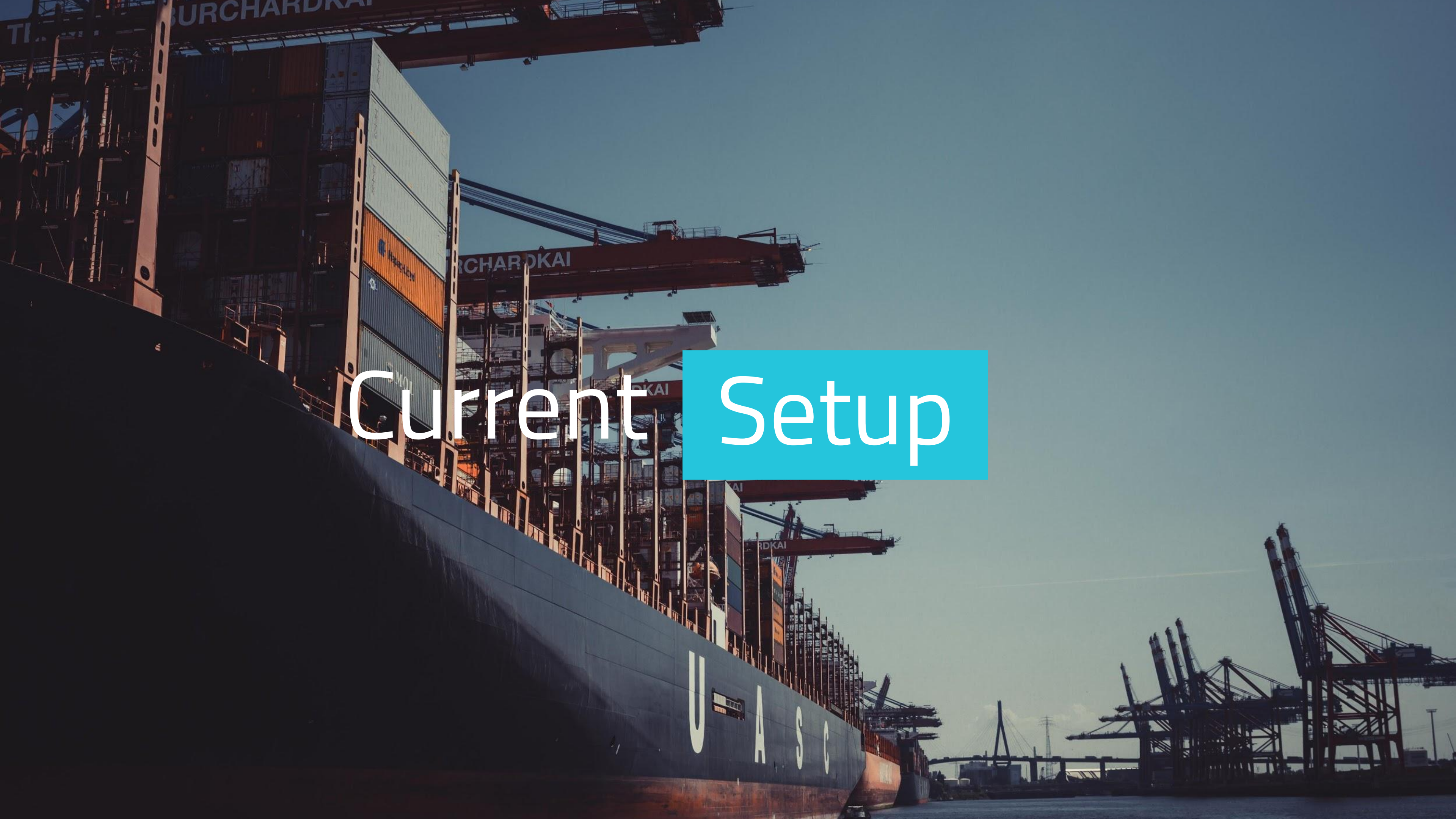
REWE Lieferservice
Ferdinand's Saar Dry Gin 0,5l
0,50l (1 l = 69,98 €)
34,99 €

REWE Lieferservice
Hendrick's Gin 0,7l
0,70l (1 l = 49,86 €)
34,90 €

REWE Lieferservice
The Duke Munich Dry Gin 0,7l
0,70l (1 l = 42,84 €)
29,99 €



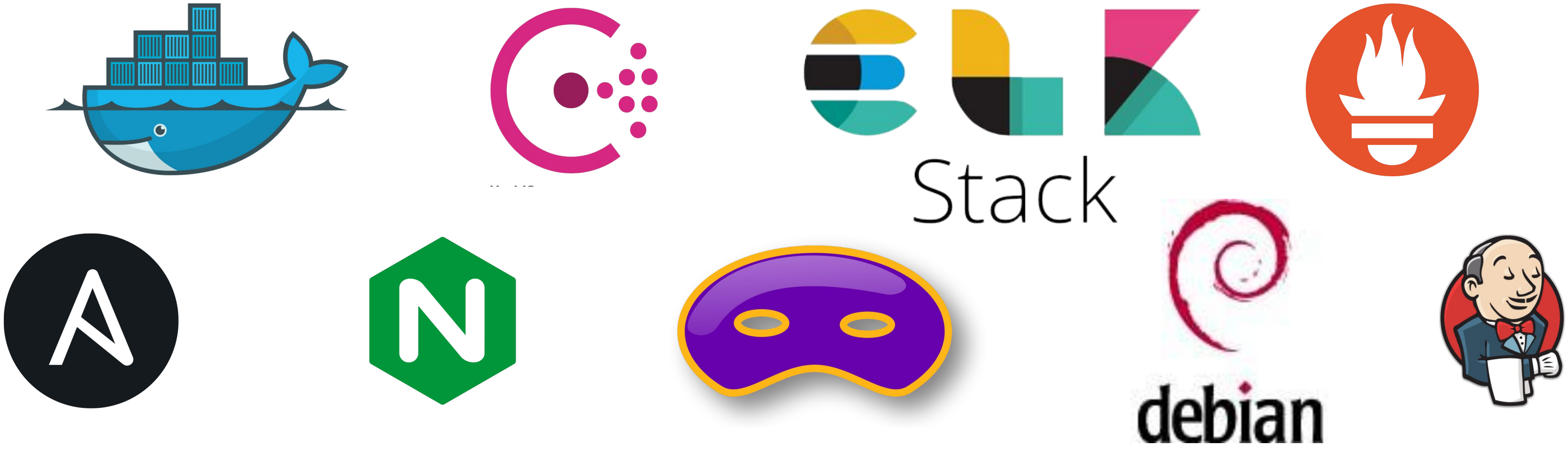
Current Setup



Recap

The state of 2018

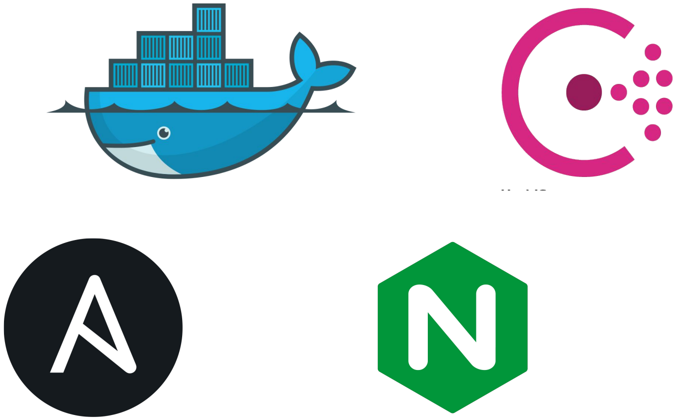
We’re operating a custom Docker-Environment consisting of:



Recap

The state of 2018

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



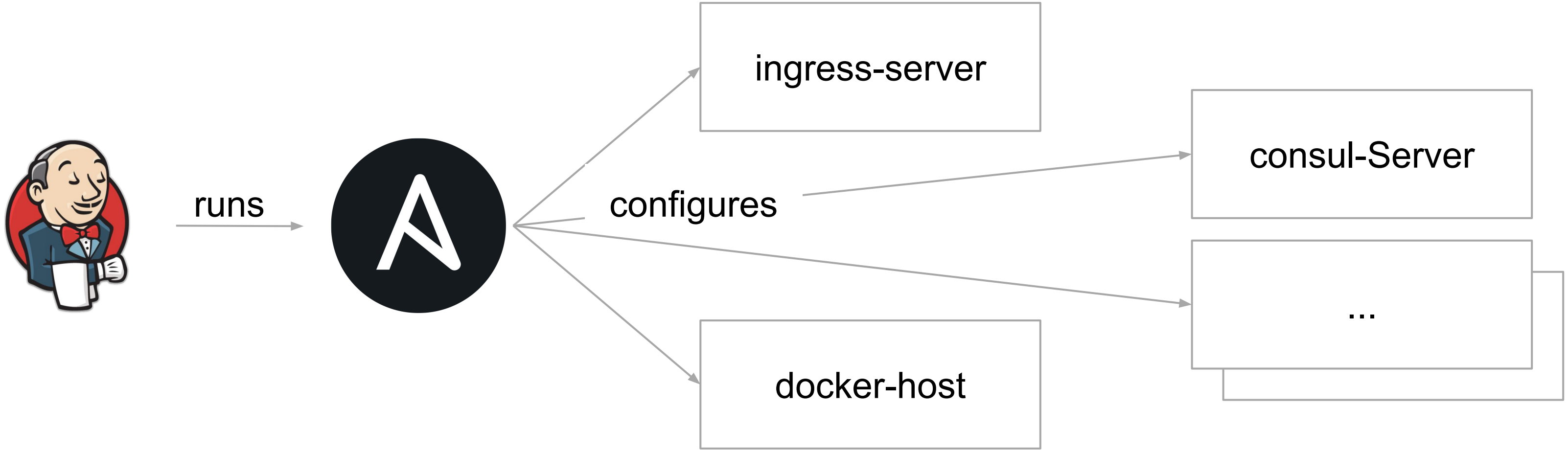
Everything was cool. Developers can bring Code live. All is well.

... and looks like this

for infrastructure provisioning

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

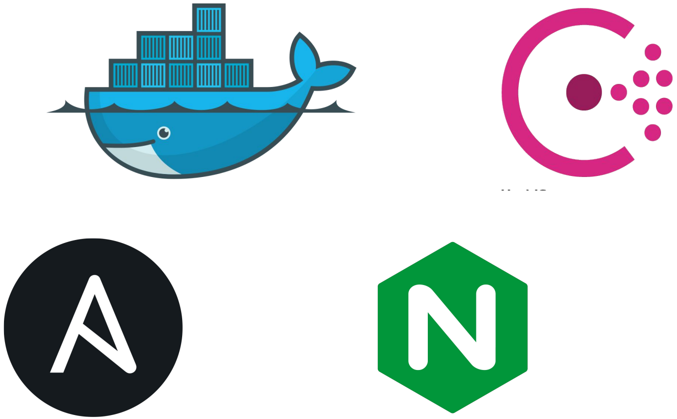
Example

```yaml
---
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

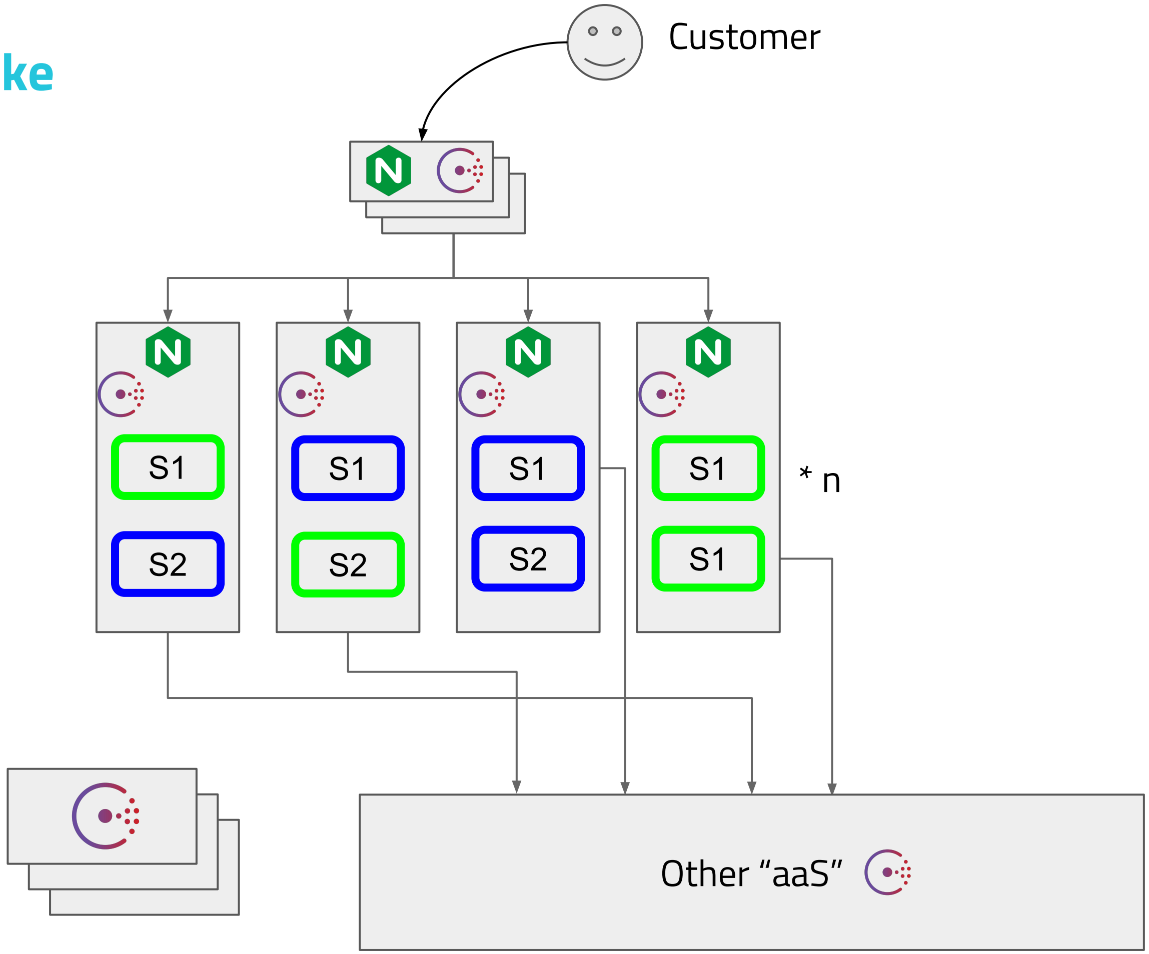
Photo by Perfecto Capucine from Pexels

... and looks like

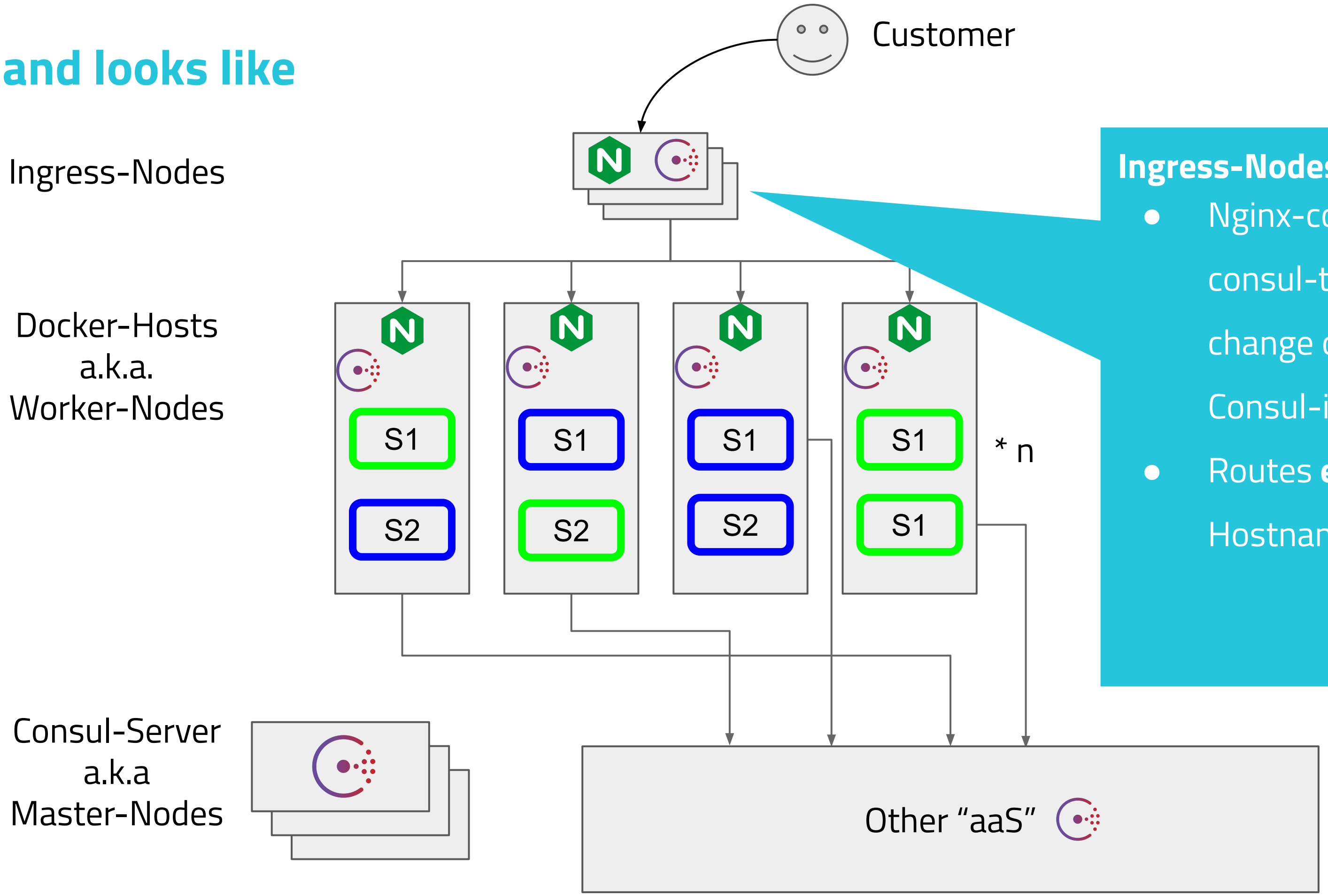
Ingress-Nodes

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

Consul-Server
a.k.a
Master-Nodes



… and looks like

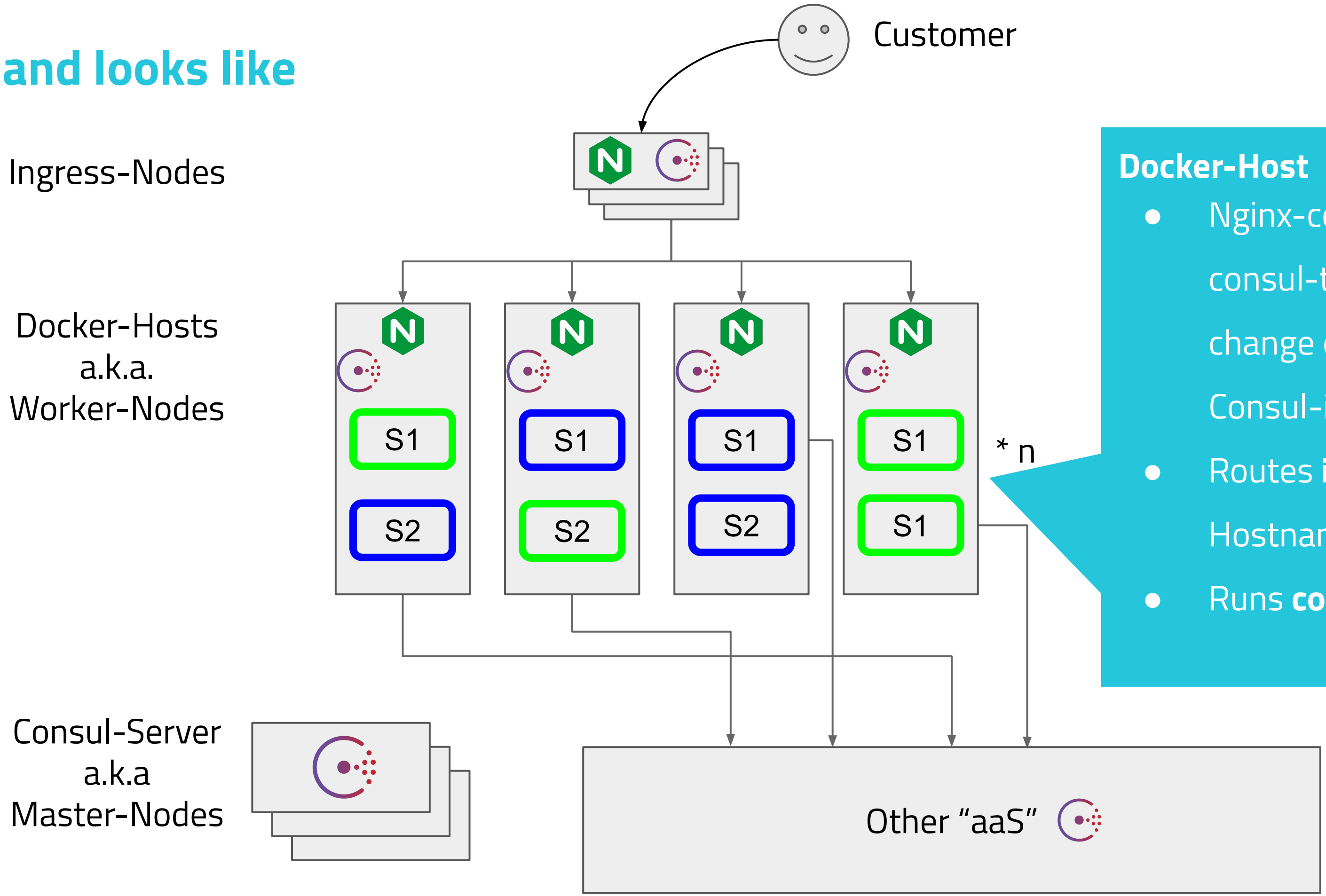


Ingress-Nodes

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

REWE digital

… and looks like



Customer

Ingress-Nodes

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

S1

S2

S1

S2

S1

S2

S1

S1

* n

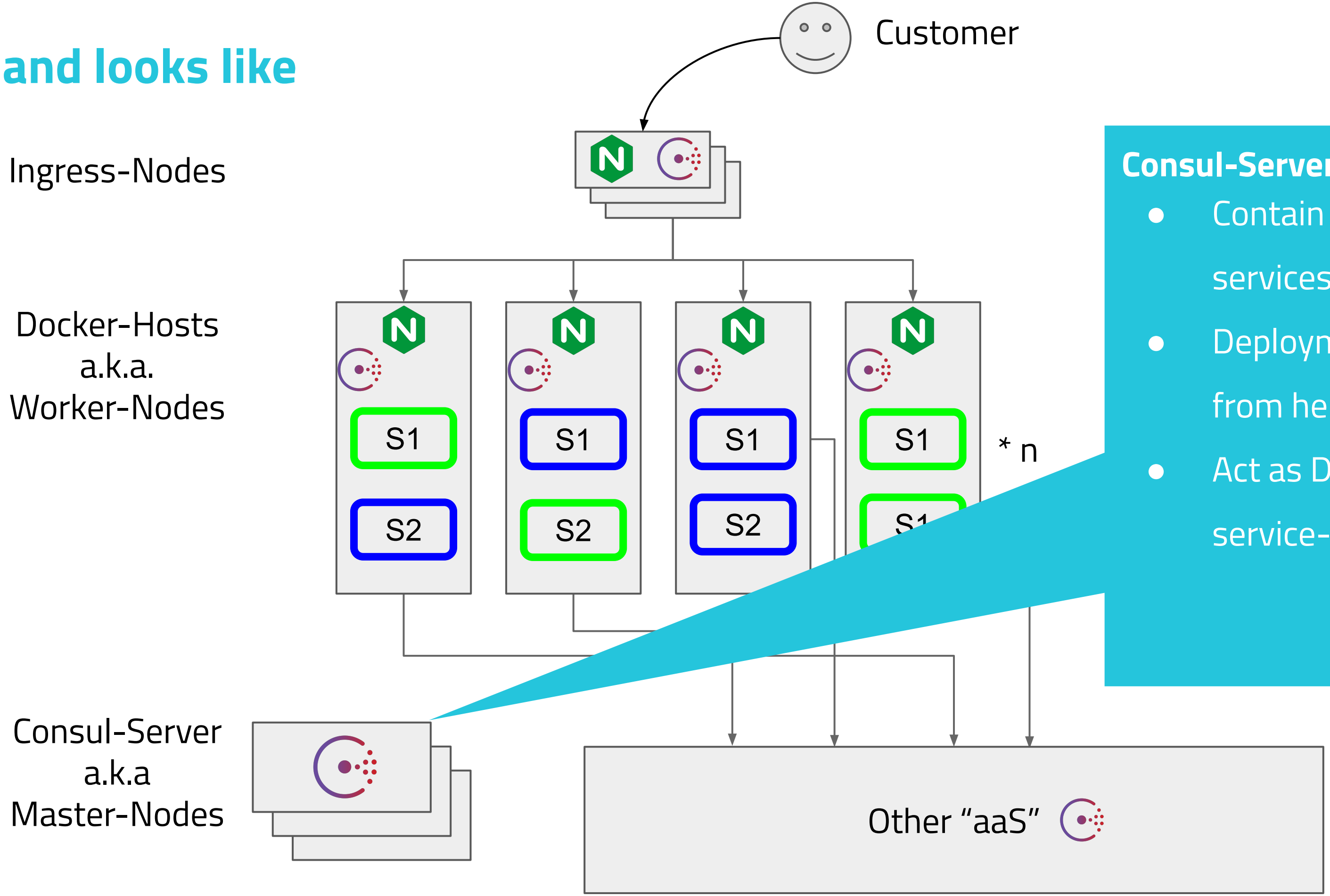
Consul-Server
a.k.a
Master-Nodes

Other “aaS”

Docker-Host

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

... and looks like

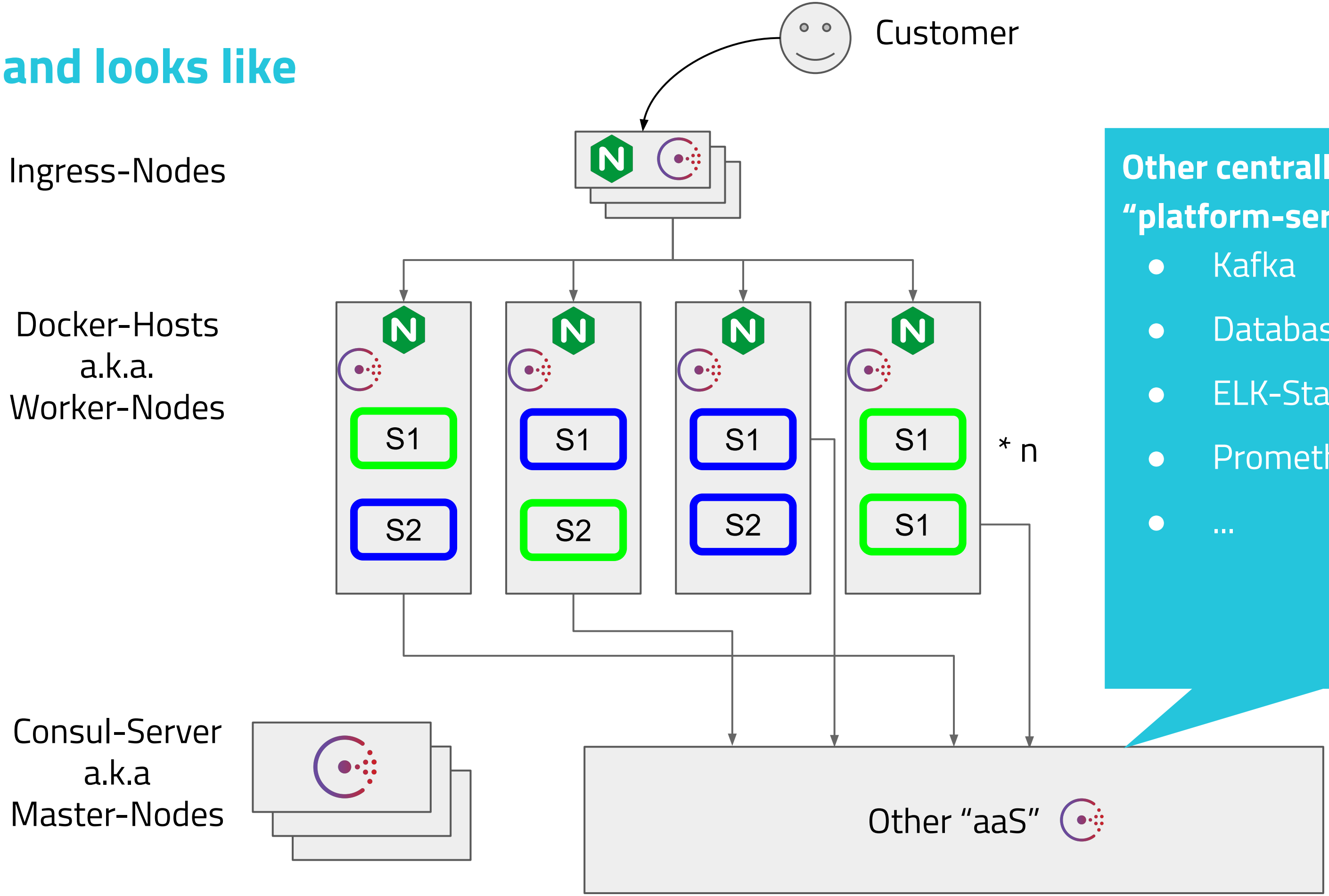


Consul-Server & Swarm-Master

- Contain knowledge of all services
- Deployments are started from here
- Act as DNS-Servers for service-discovery

REWE digital

… and looks like



Other centrally managed “platform-services”

- Kafka
- Databases
- ELK-Stack
- Prometheus & Grafana
- …

REWE digital

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

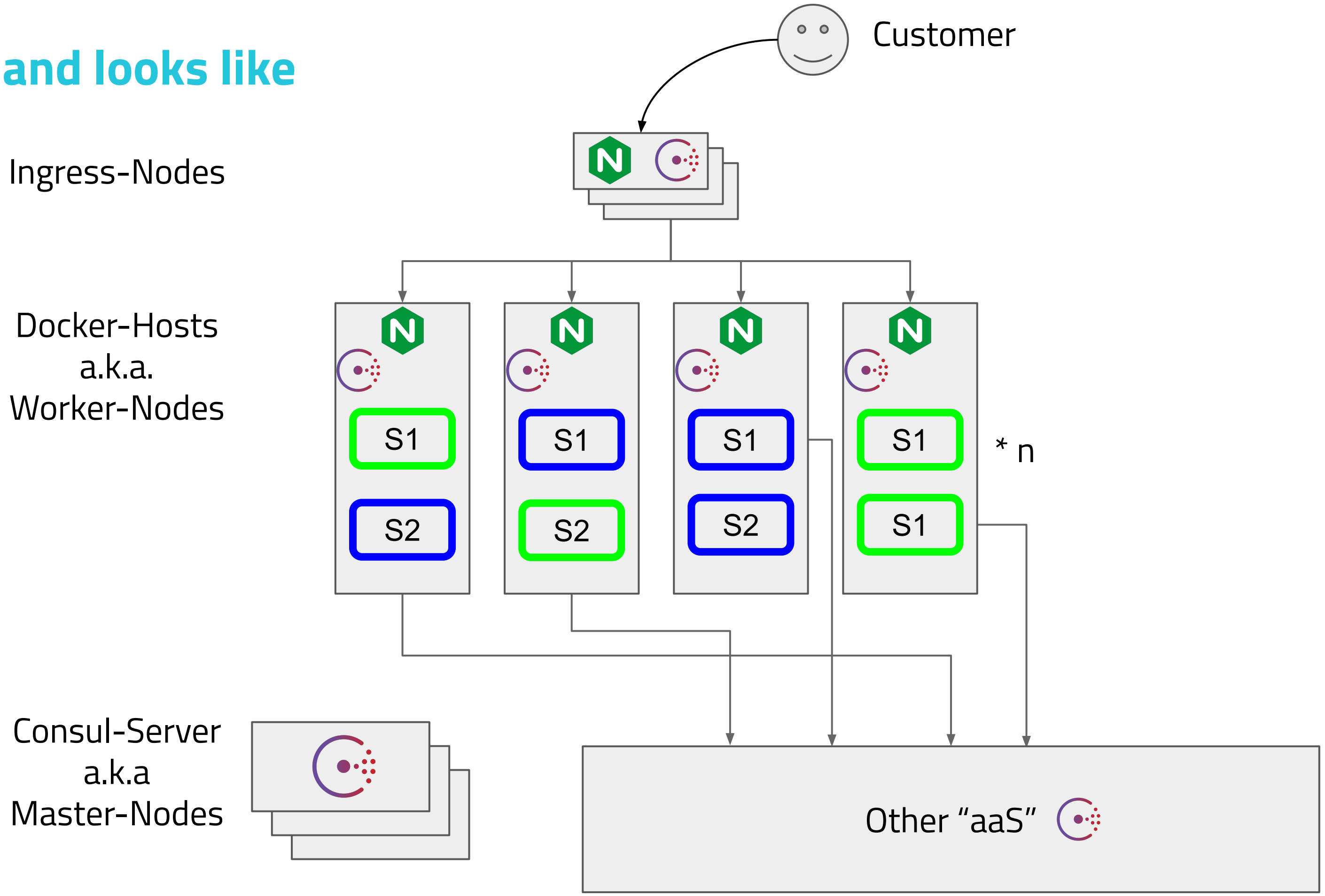
Problems with Nginx

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



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

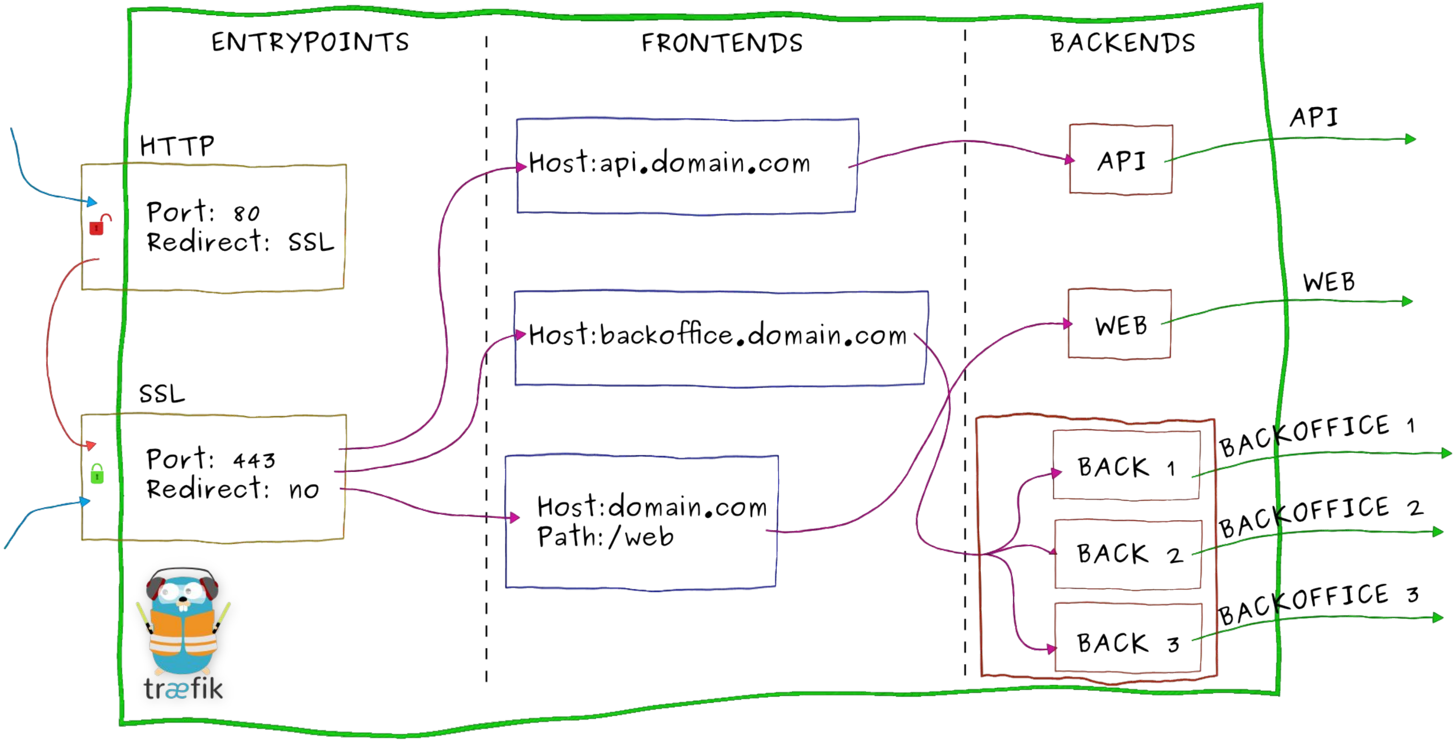
Traefik

- 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


ENTRYPOINTS
FRONTENDS
BACKENDS
HTTP
Port: 80
Redirect: SSL
SSL
Port: 443
Redirect: no
Host:api.domain.com
Host:backoffice.domain.com
Host:domain.com
Path:/web
API
WEB
BACK 1
BACK 2
BACK 3
API
WEB
BACKOFFICE 1
BACKOFFICE 2
BACKOFFICE 3
træfik

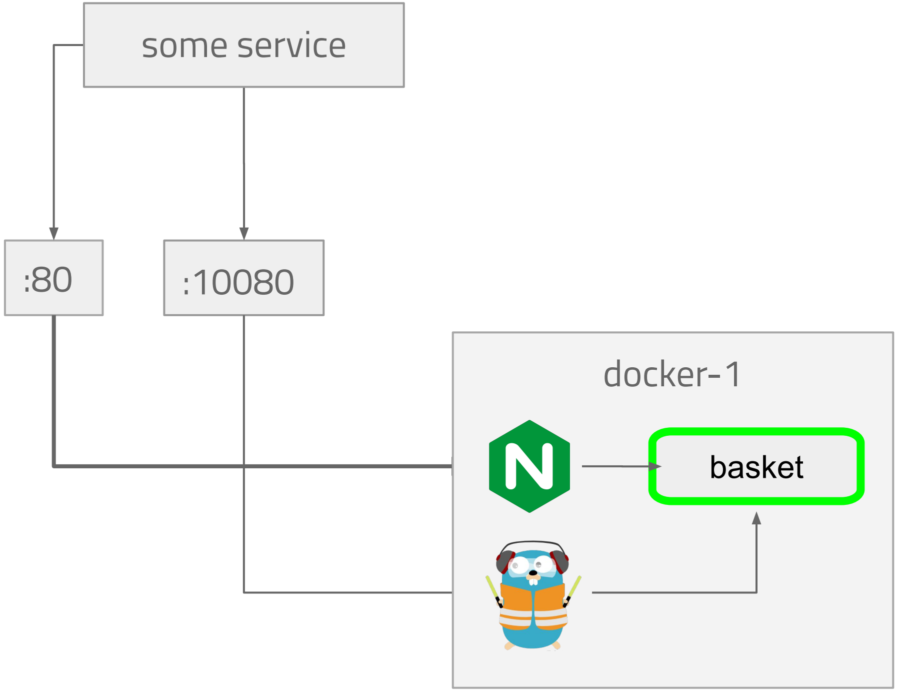
Traefik

how to migrate

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

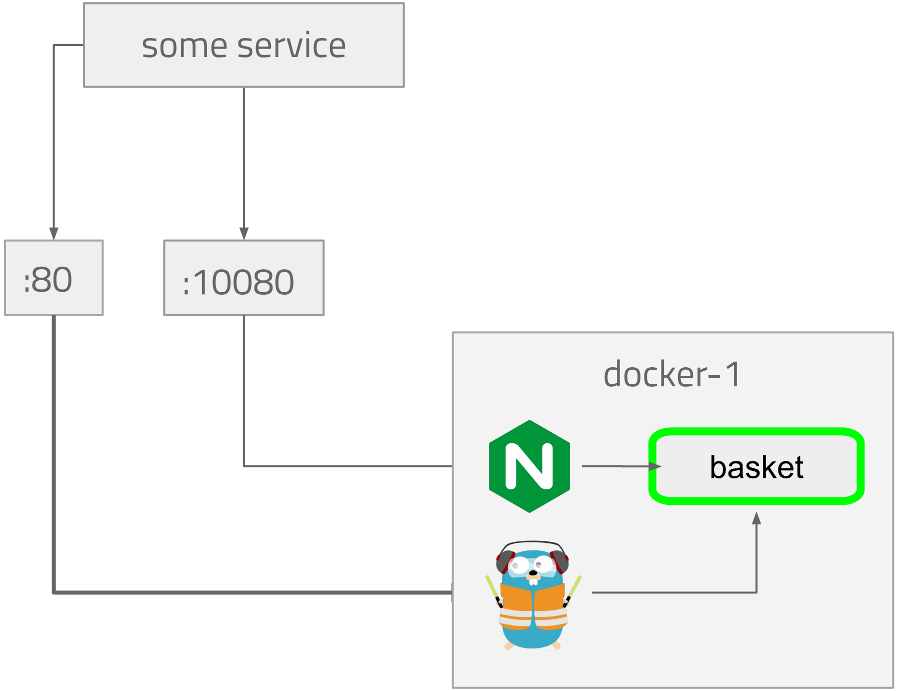
Traefik

how to migrate



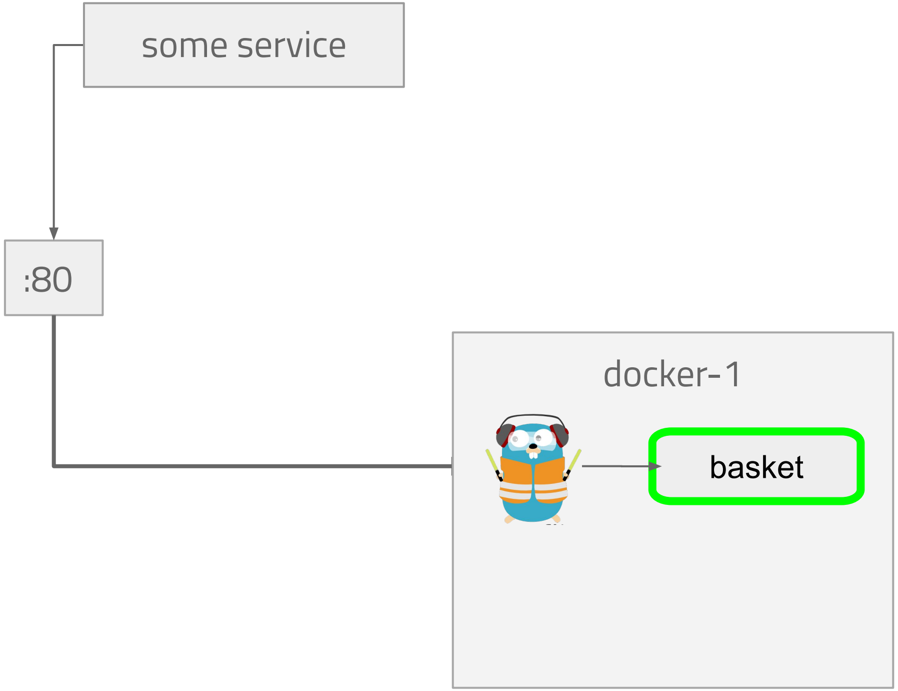
Traefik

how to migrate



Traefik

how to migrate



REWE digital

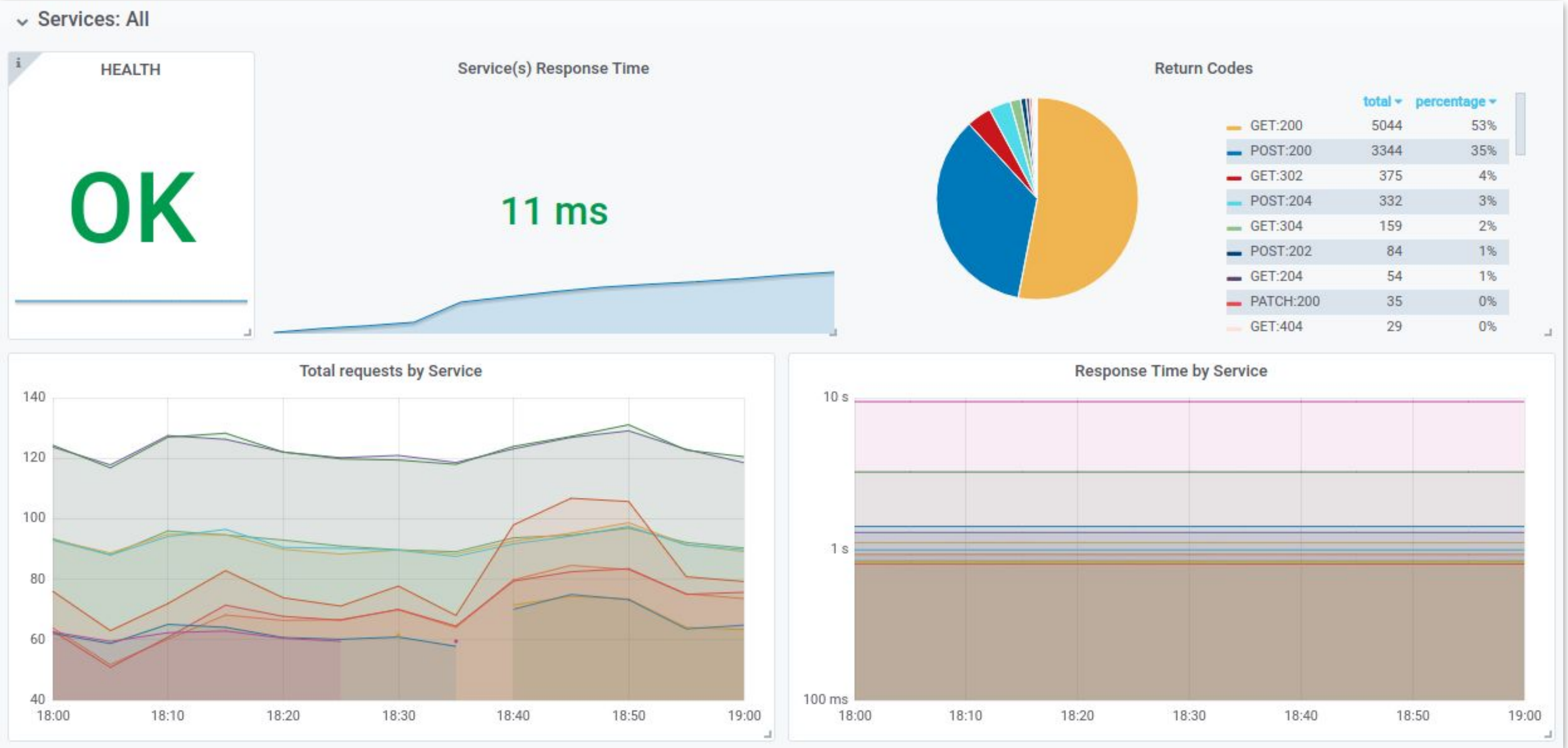
Traefik

Benefits

- Keepalive and connection problems immediately went away
- Almost real time data about service response time
- Web UI to check routes
- Rich access logs

Traefik

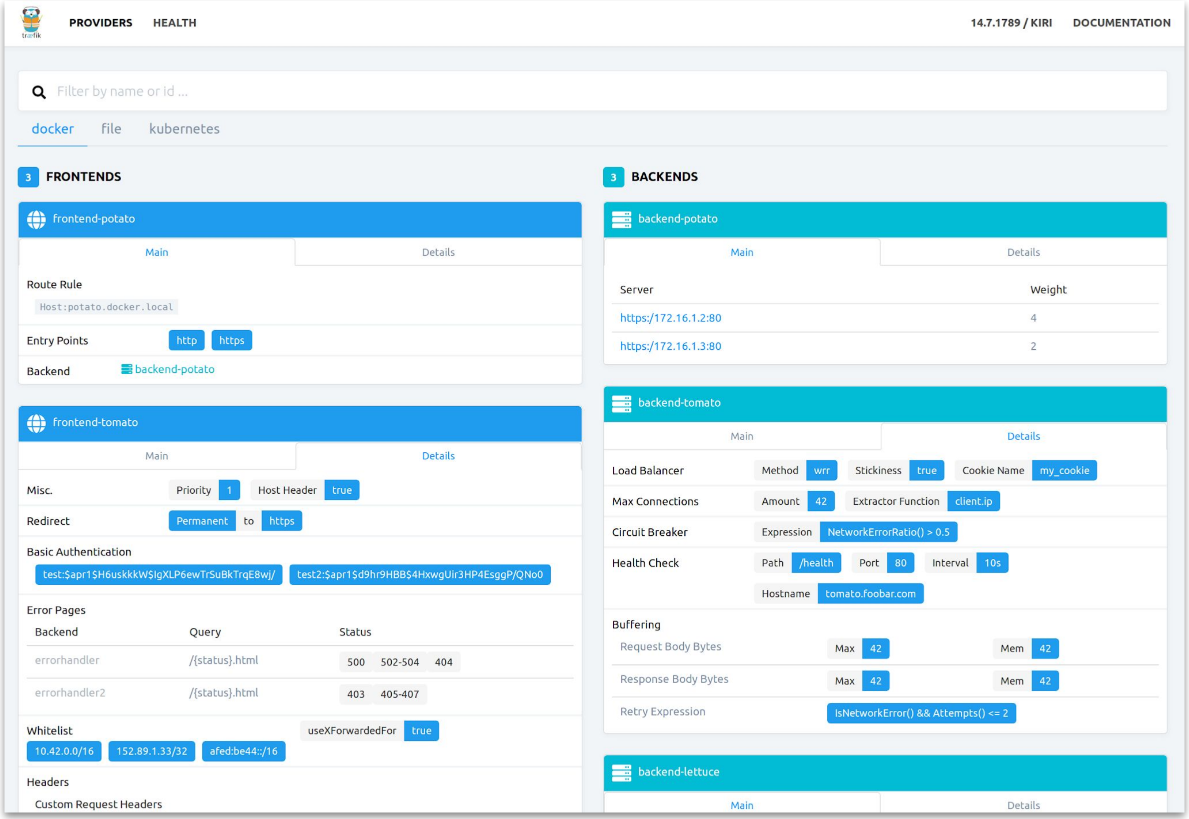
Benefits



	total	percentage
GET:200	5044	53%
POST:200	3344	35%
GET:302	375	4%
POST:204	332	3%
GET:304	159	2%
POST:202	84	1%
GET:204	54	1%
PATCH:200	35	0%
GET:404	29	0%

Traefik

Benefits



PROVIDERS HEALTH 14.7.1789 / KIRI DOCUMENTATION

Filter by name or id ...

docker file kubernetes

3 FRONTENDS

frontend-potato

Main Details

Route Rule

`Host:potato.docker.local`

Entry Points http https

Backend backend-potato

frontend-tomato

Main Details

Misc. Priority 1 Host Header true

Redirect Permanent to https

Basic Authentication

test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/ test2:$apr1$d9hr9HBB$4HxwgUir3HP4EsggP/QNo0

Error Pages

Backend	Query	Status
errorhandler	/{status}.html	500 502-504 404
errorhandler2	/{status}.html	403 405-407

Whitelist useXForwardedFor true

10.42.0.0/16 152.89.1.33/32 afed:be44::/16

Headers

Custom Request Headers

3 BACKENDS

backend-potato

Main Details

Server	Weight
https:/172.16.1.2:80	4
https:/172.16.1.3:80	2

backend-tomato

Main Details

Load Balancer Method wrr Stickiness true Cookie Name my_cookie

Max Connections Amount 42 Extractor Function client.ip

Circuit Breaker Expression NetworkErrorRatio() > 0.5

Health Check Path /health Port 80 Interval 10s

Hostname tomato.foobar.com

Buffering

Request Body Bytes Max 42 Mem 42

Response Body Bytes Max 42 Mem 42

Retry Expression IsNetworkError() && Attempts() <= 2

backend-lettuce

Main Details

Container Problems

Problems with standalone Swarm

also increased increased with increasing workload

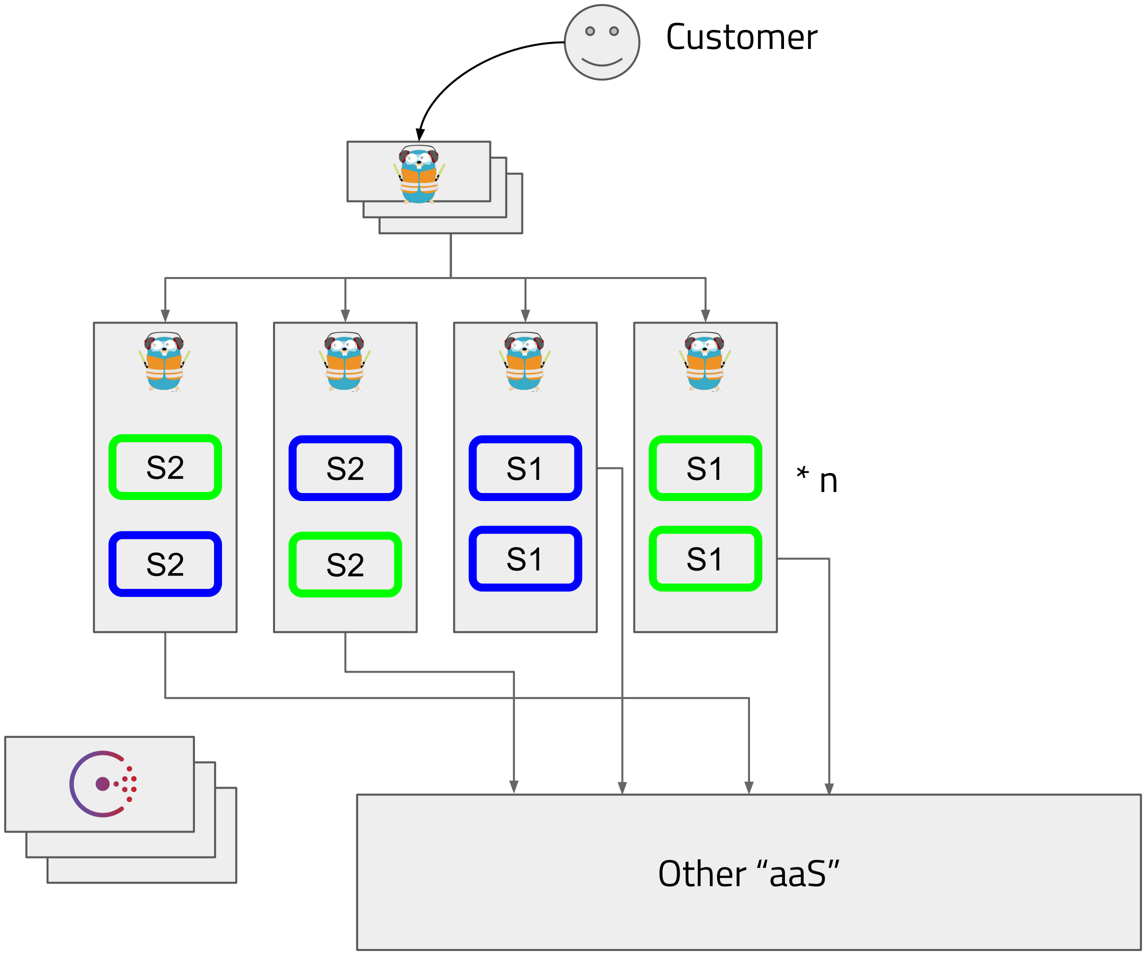
- Poor container spread
  - all service instances running on one host
- No self healing
- Manual node draining (e.g. for maintenance)
  - we're still dependent on docker-compose files
- Only few metrics

Swarm...

Ingress-Nodes

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

Consul-Server
&
Docker-Swarm
master



Customer

S2

S2

S2

S2

S1

S1

S1

S1

* n

Other “aaS”

We want this

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

Possible replacements

RANCHER

docker SWARM

HashiCorp Nomad

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

Nomad

Benefits

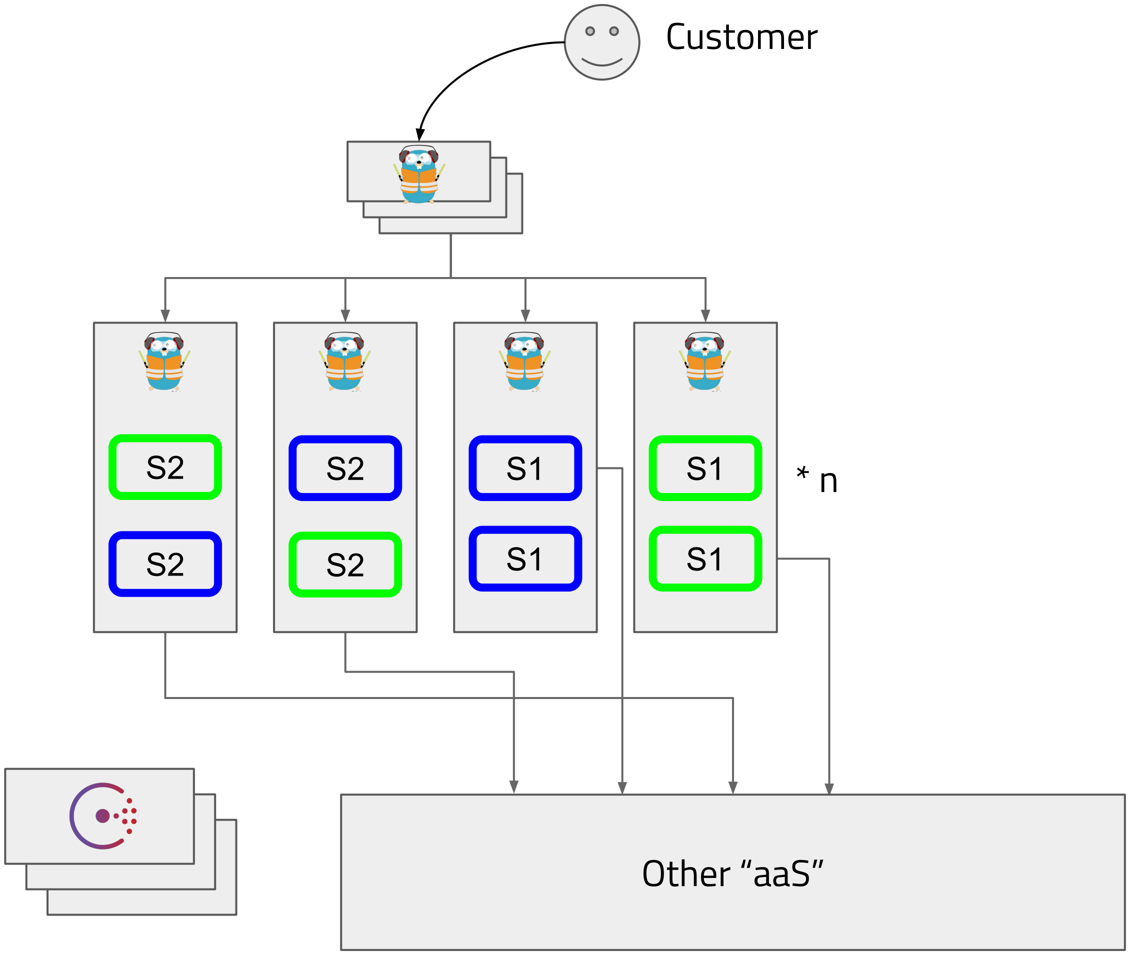
- Not limited to Docker
    - Rkt and LXC
- Not limited to Containers
    - Jar files
    - Binaries
    - VMs

Swarm...

Ingress-Nodes

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

Consul-Server
&
Nomad-Server



Customer

S2

S2

S2

S2

S1

S1

S1

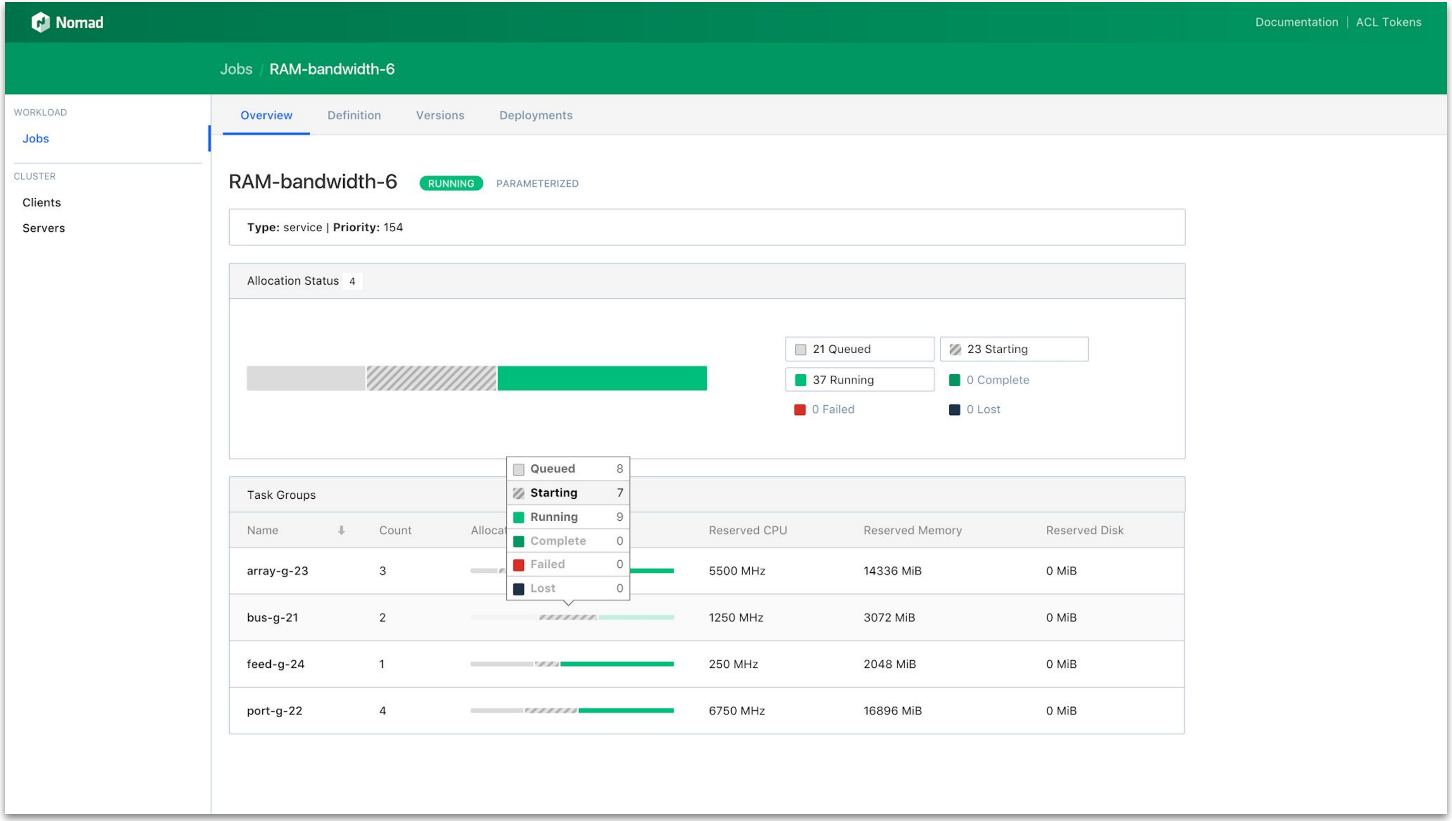
S1

* n

Other “aaS”

Nomad

Benefits



RAM-bandwidth-6 RUNNING PARAMETERIZED

Type: service | **Priority:** 154

Allocation Status 4

▢ 21 Queued	▨ 23 Starting
🟩 37 Running	🟩 0 Complete
🟥 0 Failed	■ 0 Lost

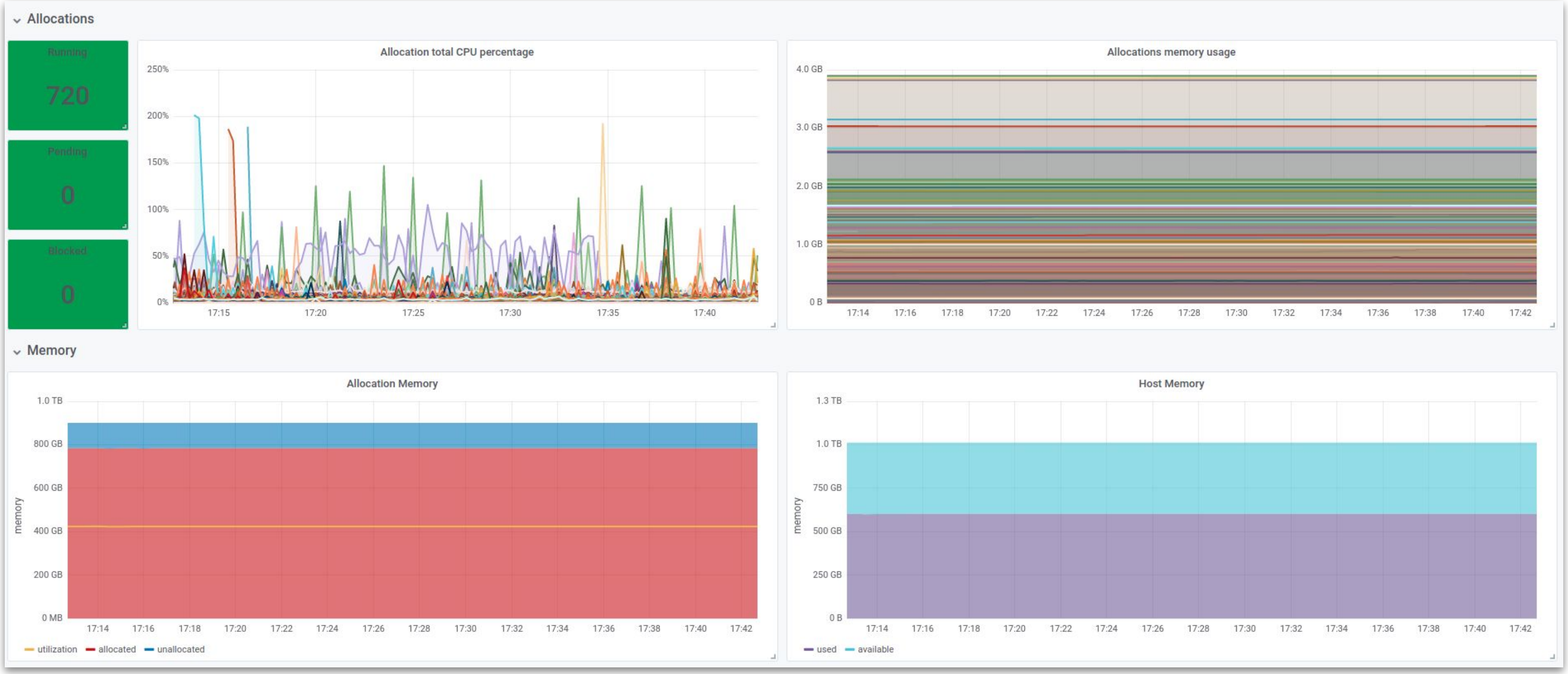
▢ Queued	8
▨ Starting	7
🟩 Running	9
🟩 Complete	0
🟥 Failed	0
■ Lost	0

Task Groups

Name	Count	Allocat	Reserved CPU	Reserved Memory	Reserved Disk
array-g-23	3		5500 MHz	14336 MiB	0 MiB
bus-g-21	2		1250 MHz	3072 MiB	0 MiB
feed-g-24	1		250 MHz	2048 MiB	0 MiB
port-g-22	4		6750 MHz	16896 MiB	0 MiB

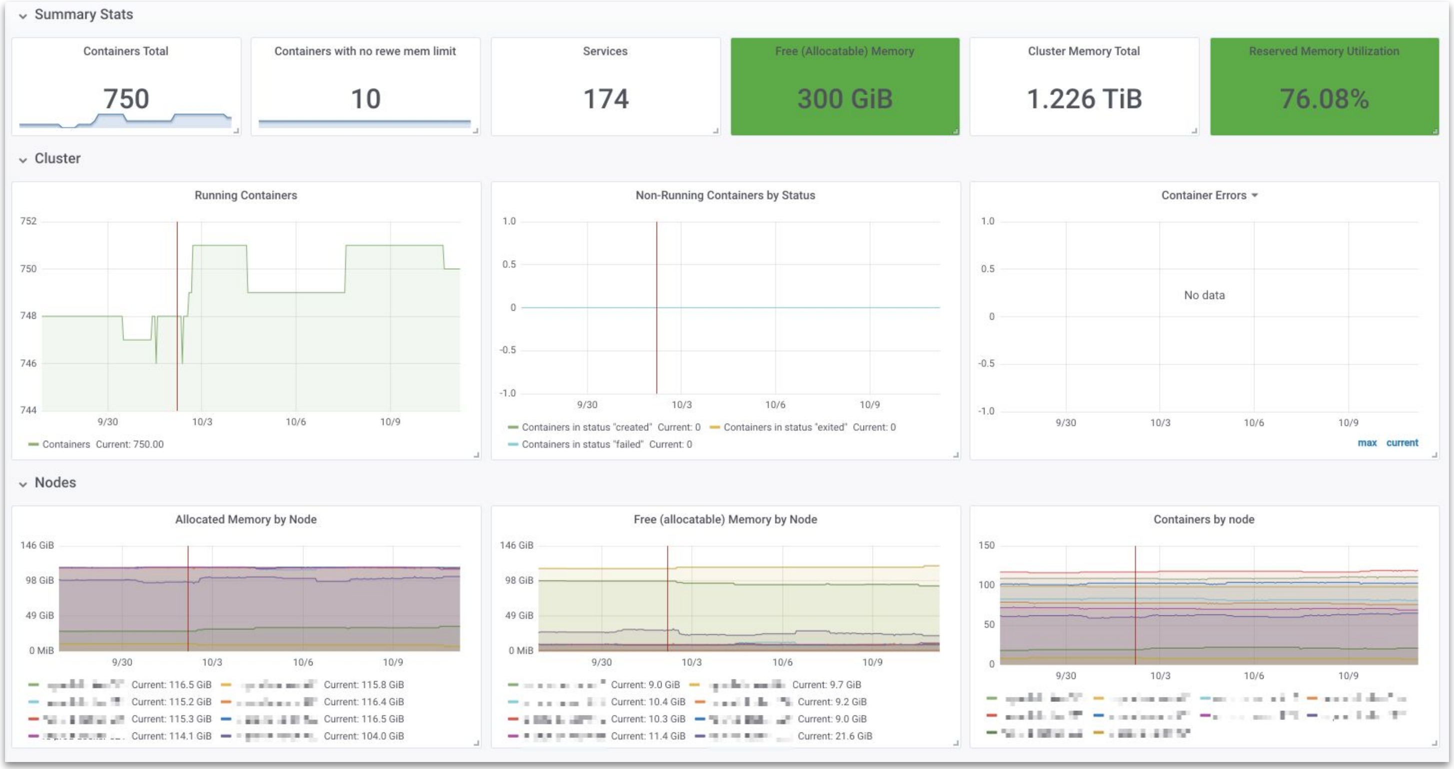
Nomad

Benefits - Cluster Level



Nomad

Benefits - Cluster Level



Summary Stats

Containers Total	Containers with no rewe mem limit	Services	Free (Allocatable) Memory	Cluster Memory Total	Reserved Memory Utilization
750	10	174	300 GiB	1.226 TiB	76.08%

Cluster

Running Containers

Containers Current: 750.00

Non-Running Containers by Status

Containers in status "created" Current: 0
Containers in status "exited" Current: 0
Containers in status "failed" Current: 0

Container Errors

Nodes

Allocated Memory by Node

[redacted] Current: 116.5 GiB
[redacted] Current: 115.8 GiB
[redacted] Current: 115.2 GiB
[redacted] Current: 116.4 GiB
[redacted] Current: 115.3 GiB
[redacted] Current: 116.5 GiB
[redacted] Current: 114.1 GiB
[redacted] Current: 104.0 GiB

Free (allocatable) Memory by Node

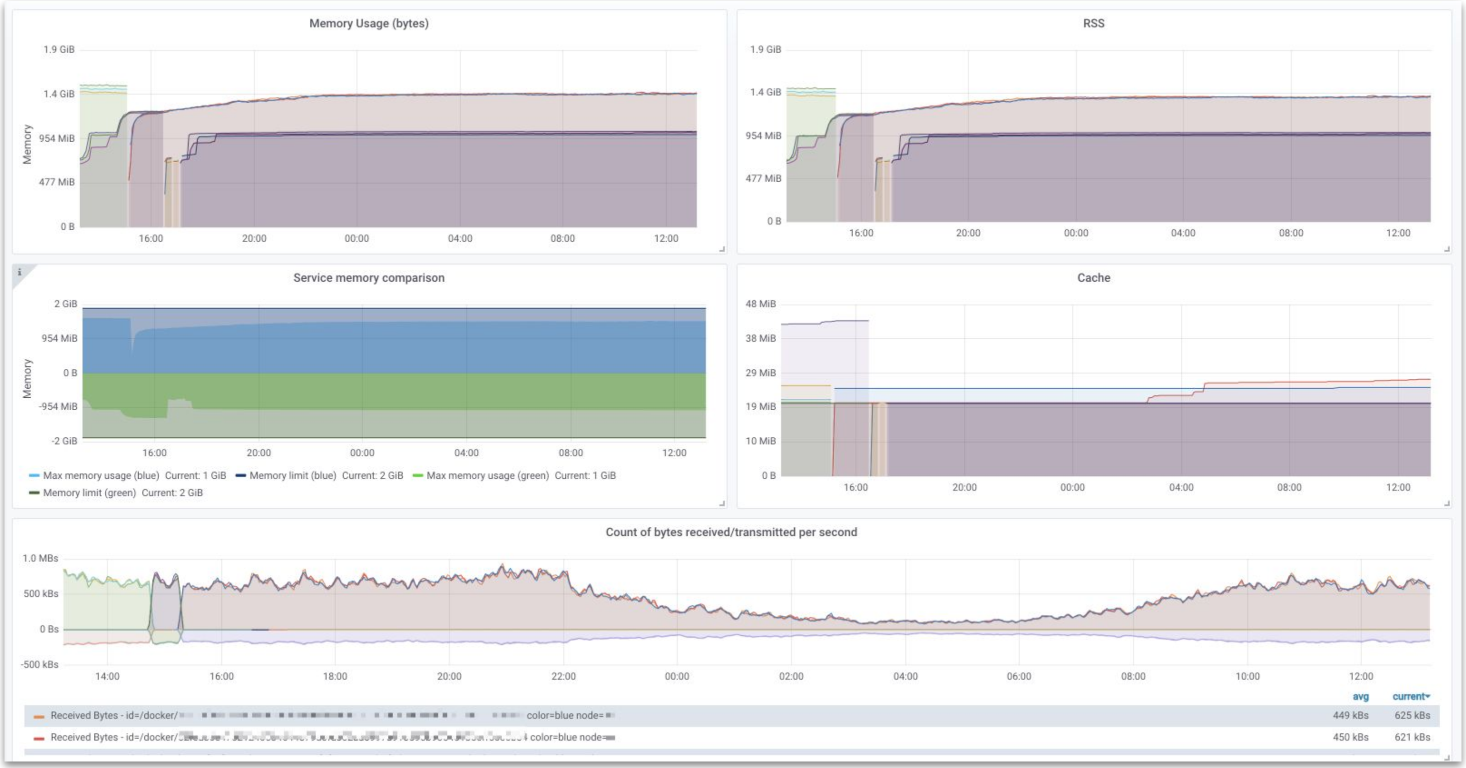
[redacted] Current: 9.0 GiB
[redacted] Current: 9.7 GiB
[redacted] Current: 10.4 GiB
[redacted] Current: 9.2 GiB
[redacted] Current: 10.3 GiB
[redacted] Current: 9.0 GiB
[redacted] Current: 11.4 GiB
[redacted] Current: 21.6 GiB

Containers by node

[redacted]

Nomad

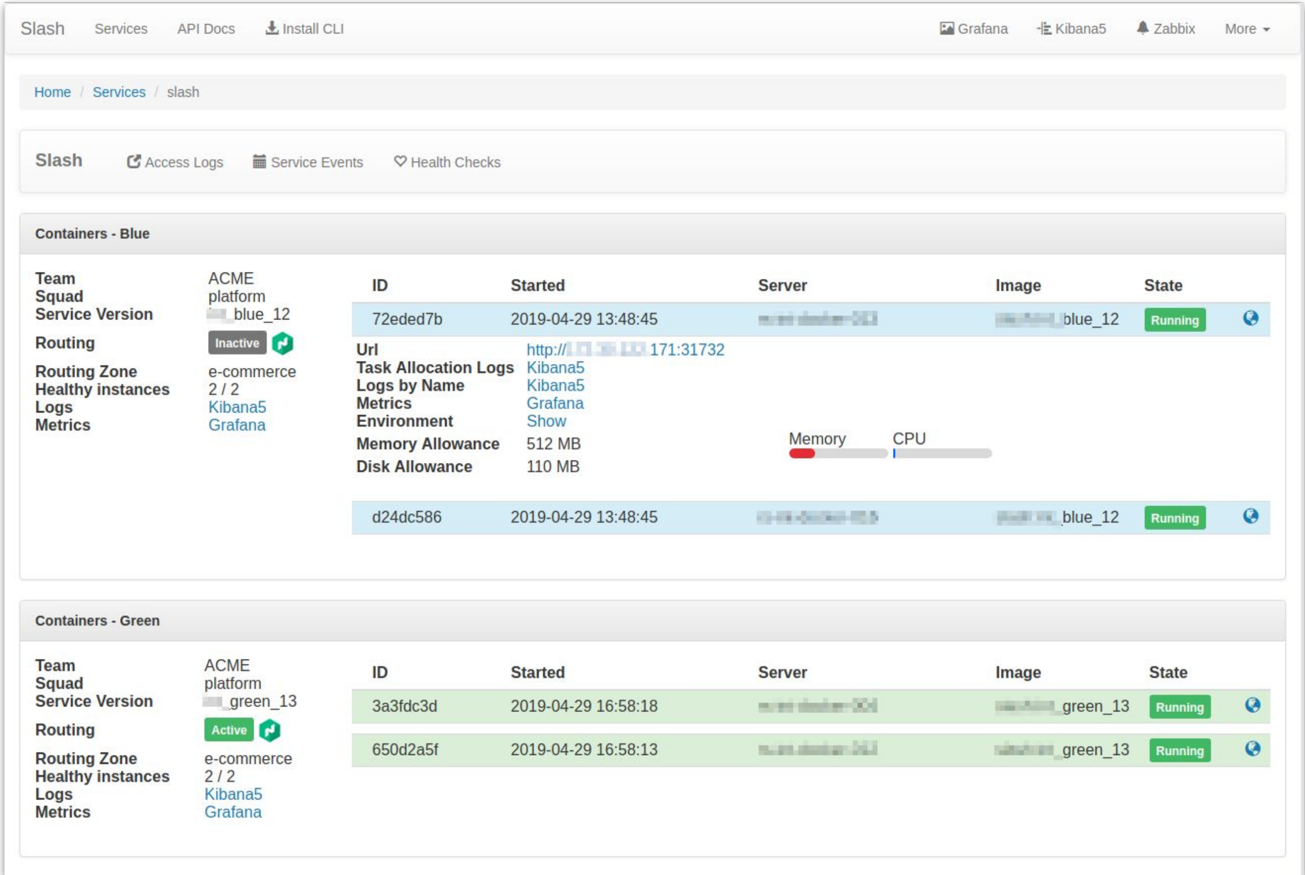
Benefits - Service Level



	avg	current
Received Bytes - id=/docker/[redacted] color=blue node=[redacted]	449 kBs	625 kBs
Received Bytes - id=/docker/[redacted] color=blue node=[redacted]	450 kBs	621 kBs

Nomad

Benefits



Slash Services API Docs Install CLI Grafana Kibana5 Zabbix More

Home / Services / slash

Slash Access Logs Service Events Health Checks

Containers - Blue

Team	ACME
Squad	platform
Service Version	[redacted]_blue_12
Routing	Inactive
Routing Zone	e-commerce
Healthy instances	2 / 2
Logs	Kibana5
Metrics	Grafana

ID	Started	Server	Image	State
72eded7b	2019-04-29 13:48:45	[redacted]	[redacted]_blue_12	Running
d24dc586	2019-04-29 13:48:45	[redacted]	[redacted]_blue_12	Running

Url	http://[redacted]171:31732
Task Allocation Logs	Kibana5
Logs by Name	Kibana5
Metrics	Grafana
Environment	Show
Memory Allowance	512 MB
Disk Allowance	110 MB

Memory CPU

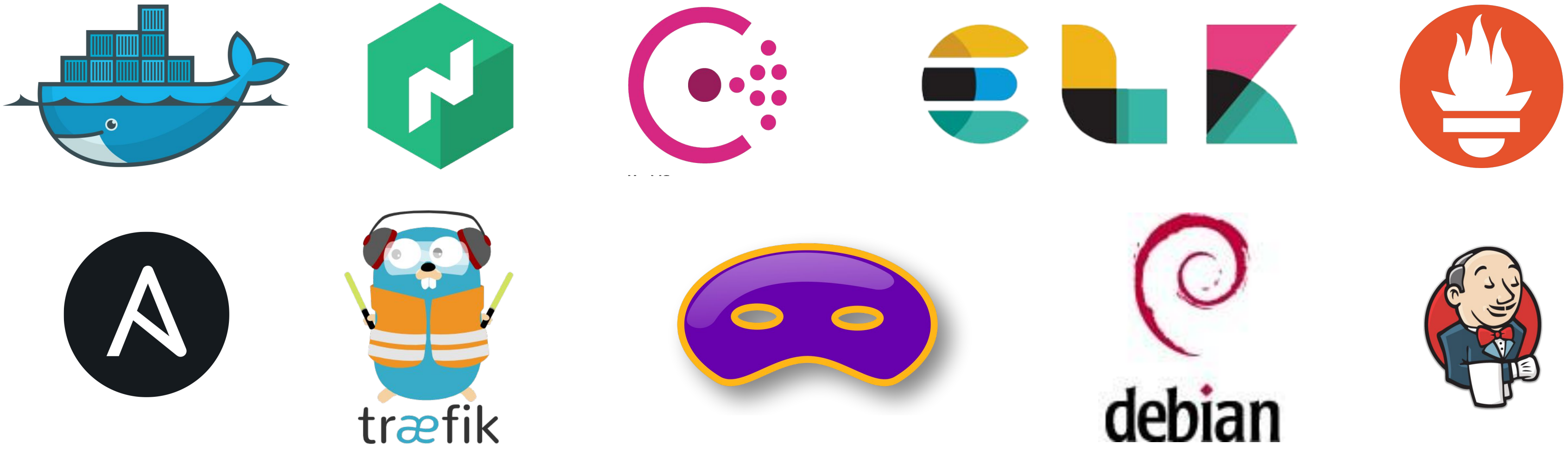
Containers - Green

Team	ACME
Squad	platform
Service Version	[redacted]_green_13
Routing	Active
Routing Zone	e-commerce
Healthy instances	2 / 2
Logs	Kibana5
Metrics	Grafana

ID	Started	Server	Image	State
3a3fdc3d	2019-04-29 16:58:18	[redacted]	[redacted]_green_13	Running
650d2a5f	2019-04-29 16:58:13	[redacted]	[redacted]_green_13	Running

State of 2019

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



State of 2019

And we're also using

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