Testen von IT-Infrastruktur

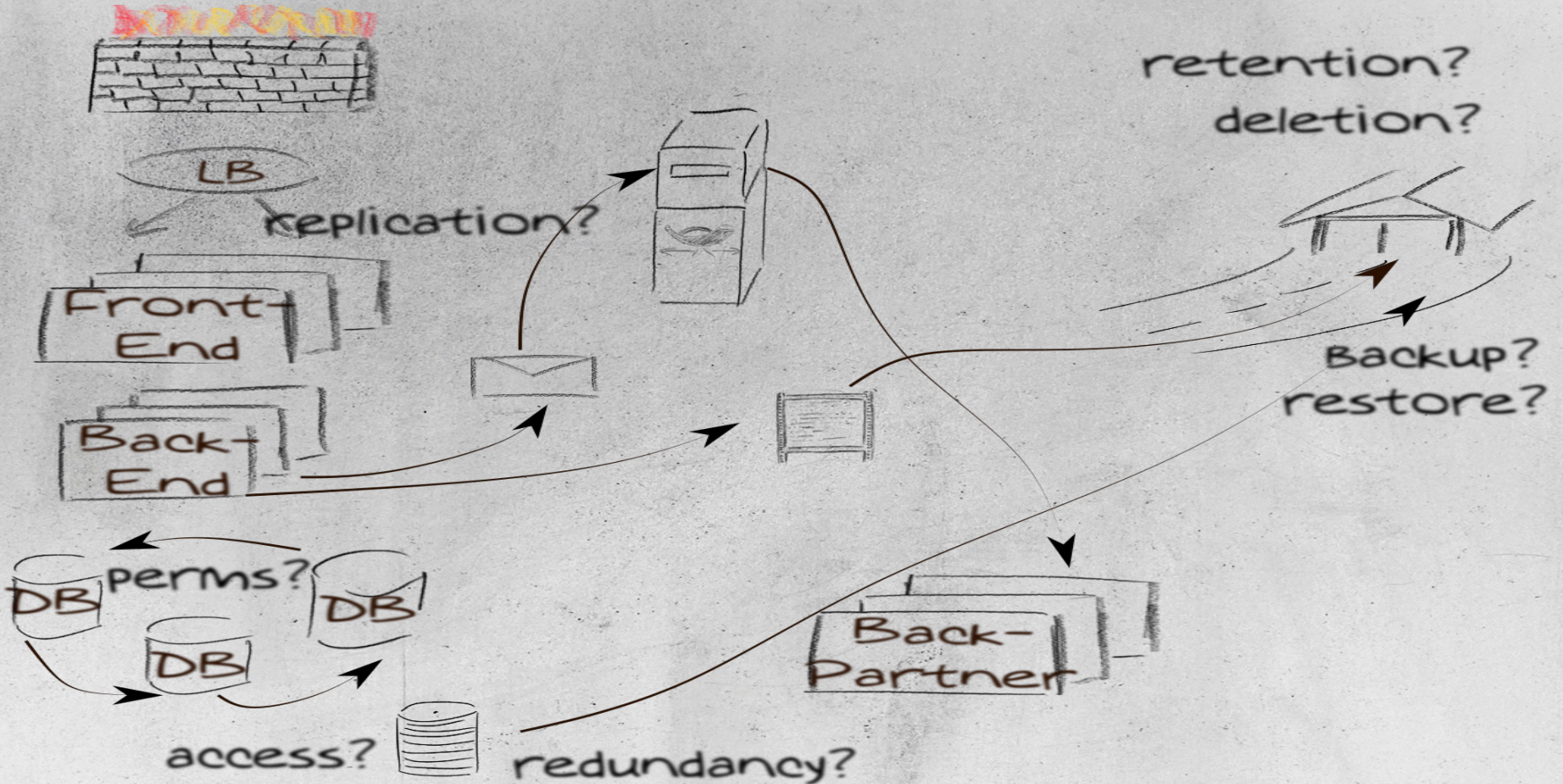
Christopher J. Ruwe <cjr@cruwe.de>
selbstständiger IT-Consultant

Organisatorisches

`cruwe/devopsessentials19.git`

- Git
- docker
- maven
- ruby / Bundler

Verteilte Systeme



Funktionsnachweis

Artefakte

- Frontend/Backend-Logic ✔
- DatenBank ?
- Storage ?
- Network ?
- Ingress ✔

Funktionsnachweis

Properties

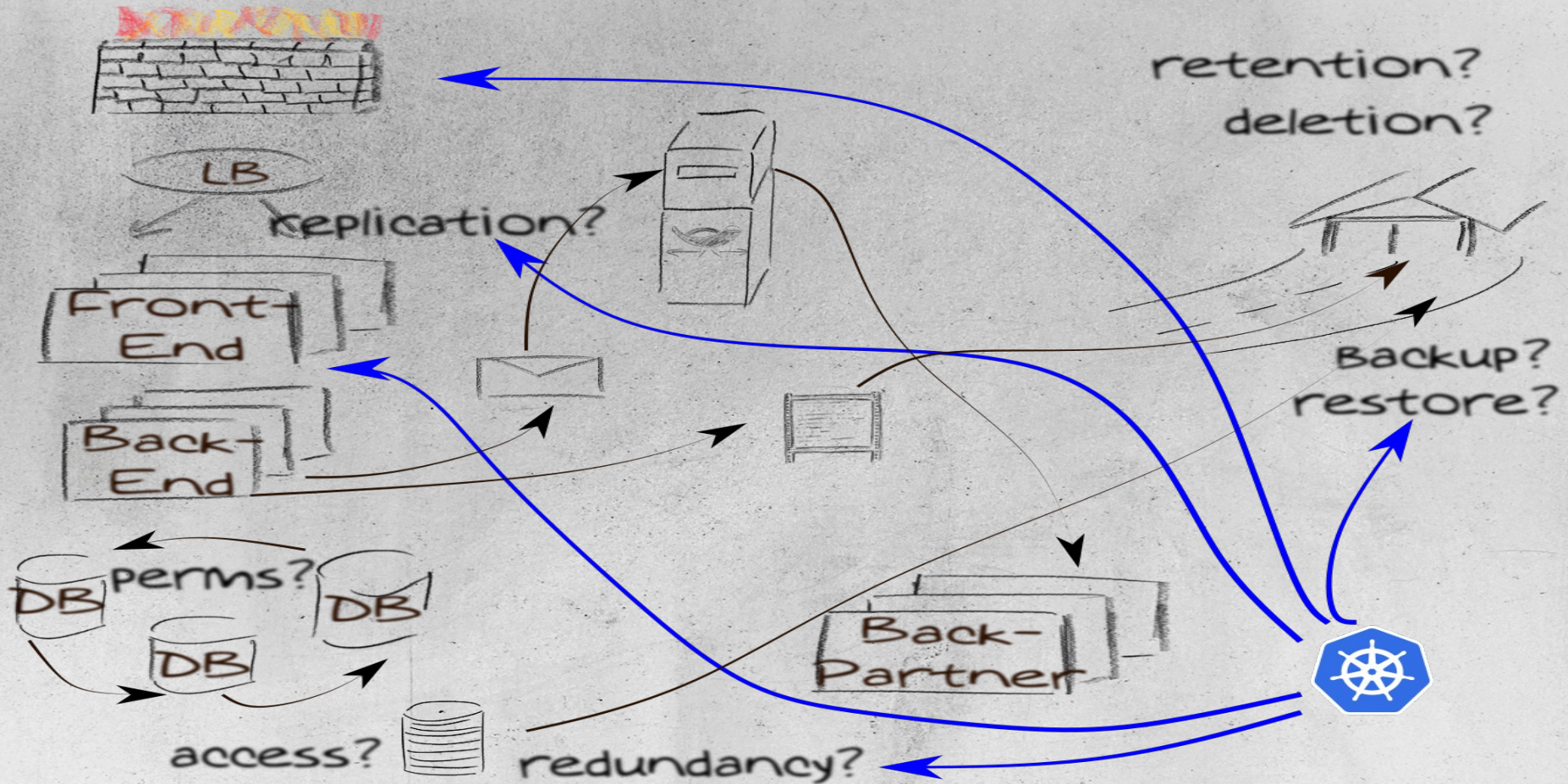
- Replikation
- Redundanz
- Permissions
- Retention / Deletion
- Restore

?

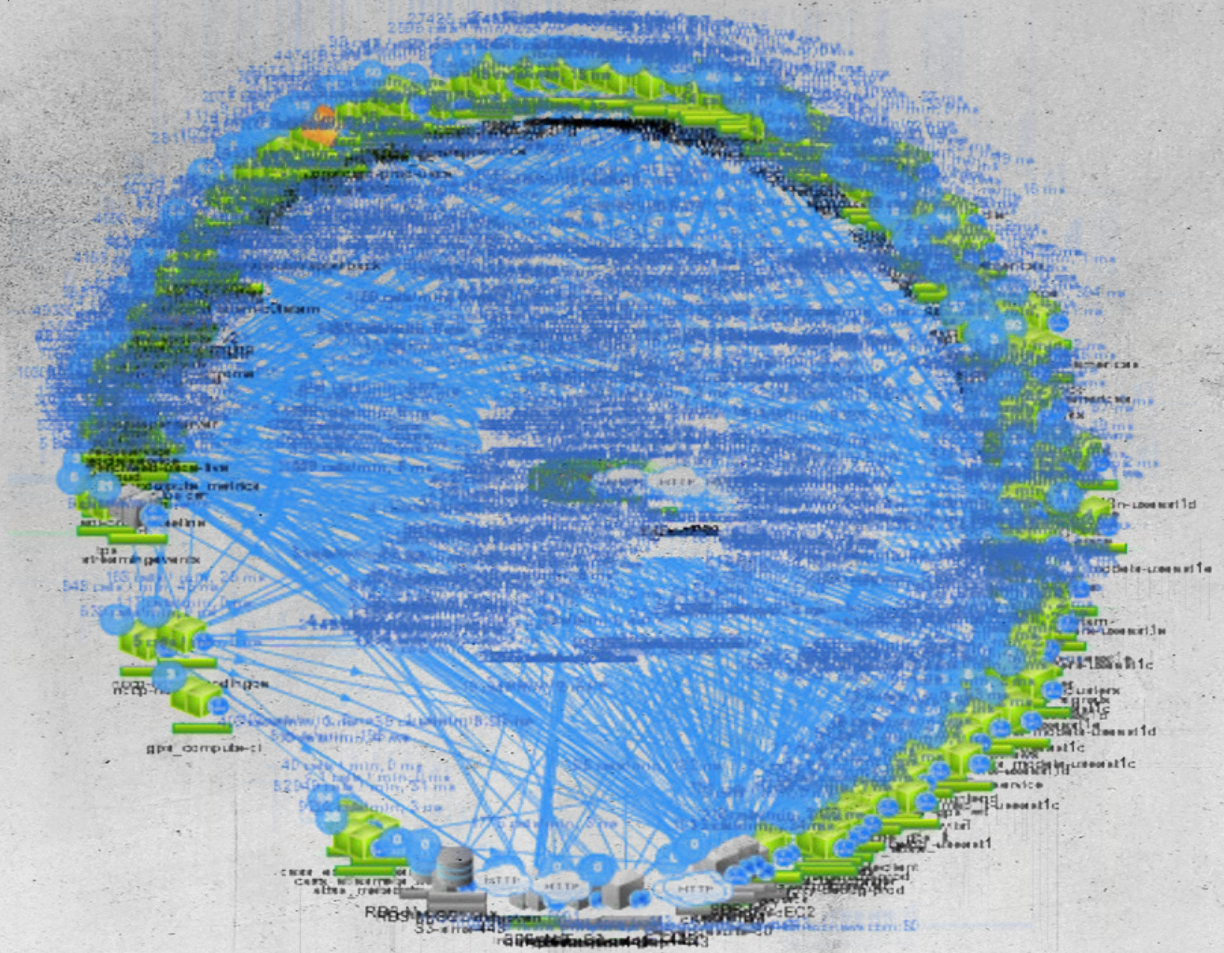
Stakeholder

- Regulierungsbehörden (BaFin / BSI)
- Kapitalgeber
- der PO?
- das „Business"?

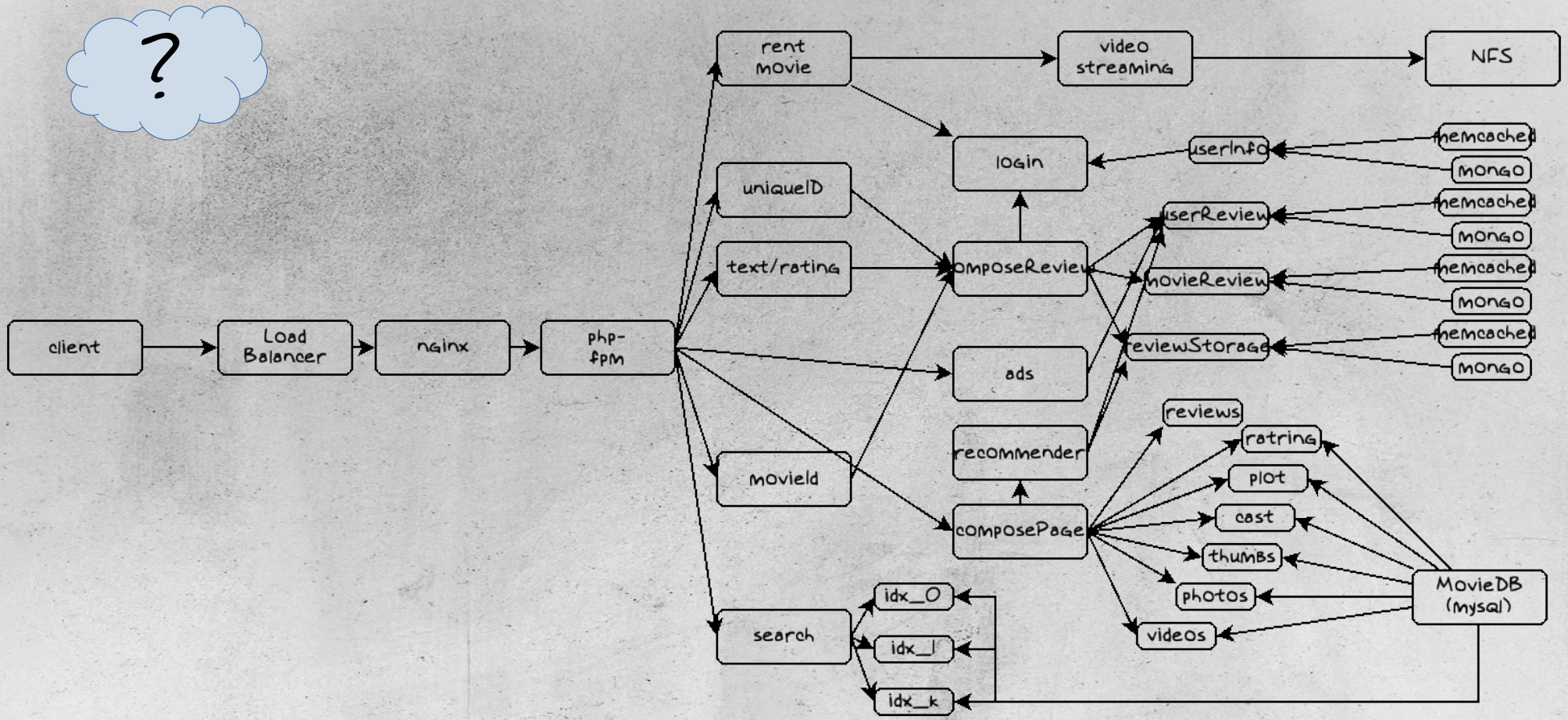
Deployment


LB
replication?
Front-End
Back-End
DB
perms?
DB
DB
access?
redundancy?
Back-Partner
retention?
deletion?
backup?
restore?

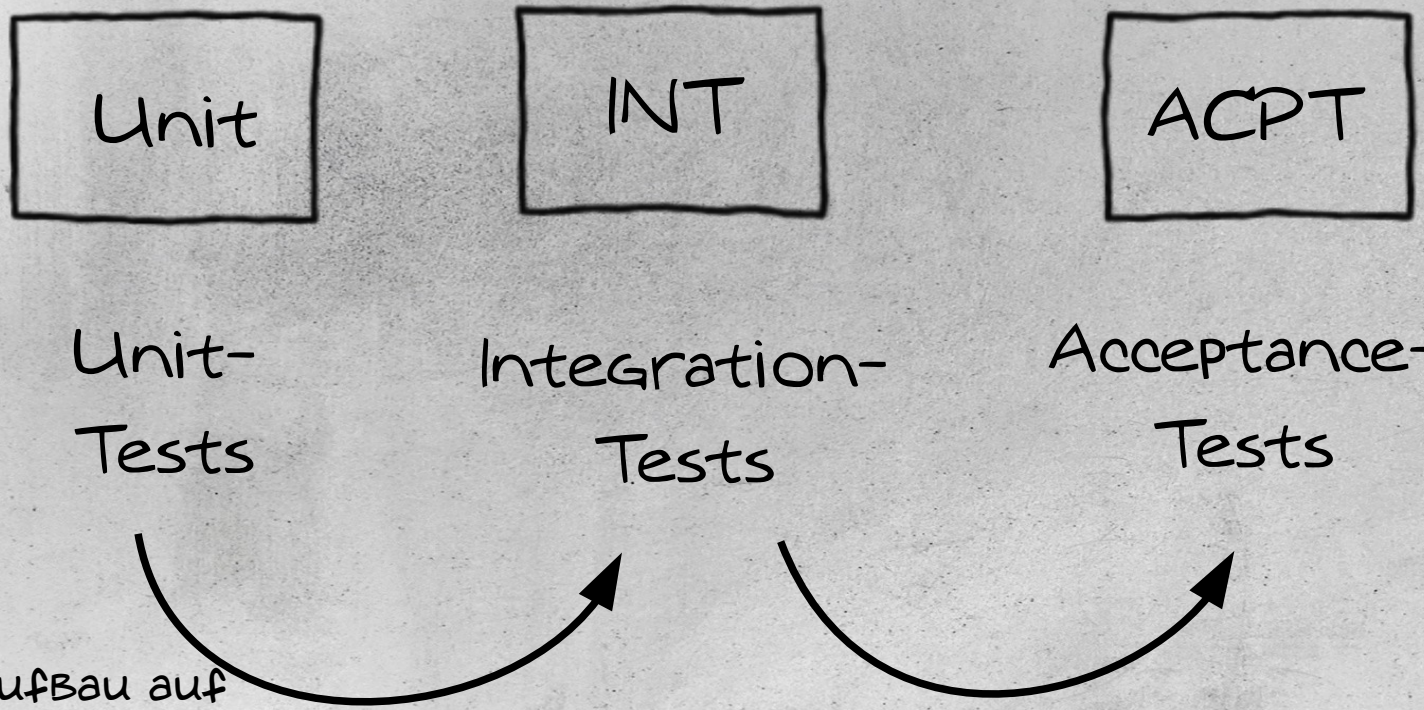
Distribution



Distribution



Testing - Steps



- Unit
  - catalogue
  - deployment
- Integration
  - Behaviour
- Acceptance / End2End
  - Behaviour

Beispiele für eine Unit

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: monitoring-prometheus
  namespace: default
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    spec:
      serviceAccountName: monitoring-prometheus
```

```yaml
- name: prometheus-server
  image: prom/prometheus
  args:
    - --config.file=/etc/config/prometheus.yml
    - --storage.tsdb.path=/data
    - --web.console.libraries=/etc/prometheus/console_libraries
    - --web.console.templates=/etc/prometheus/consoles
    - --web.enable-lifecycle
    - --log.format=json
```

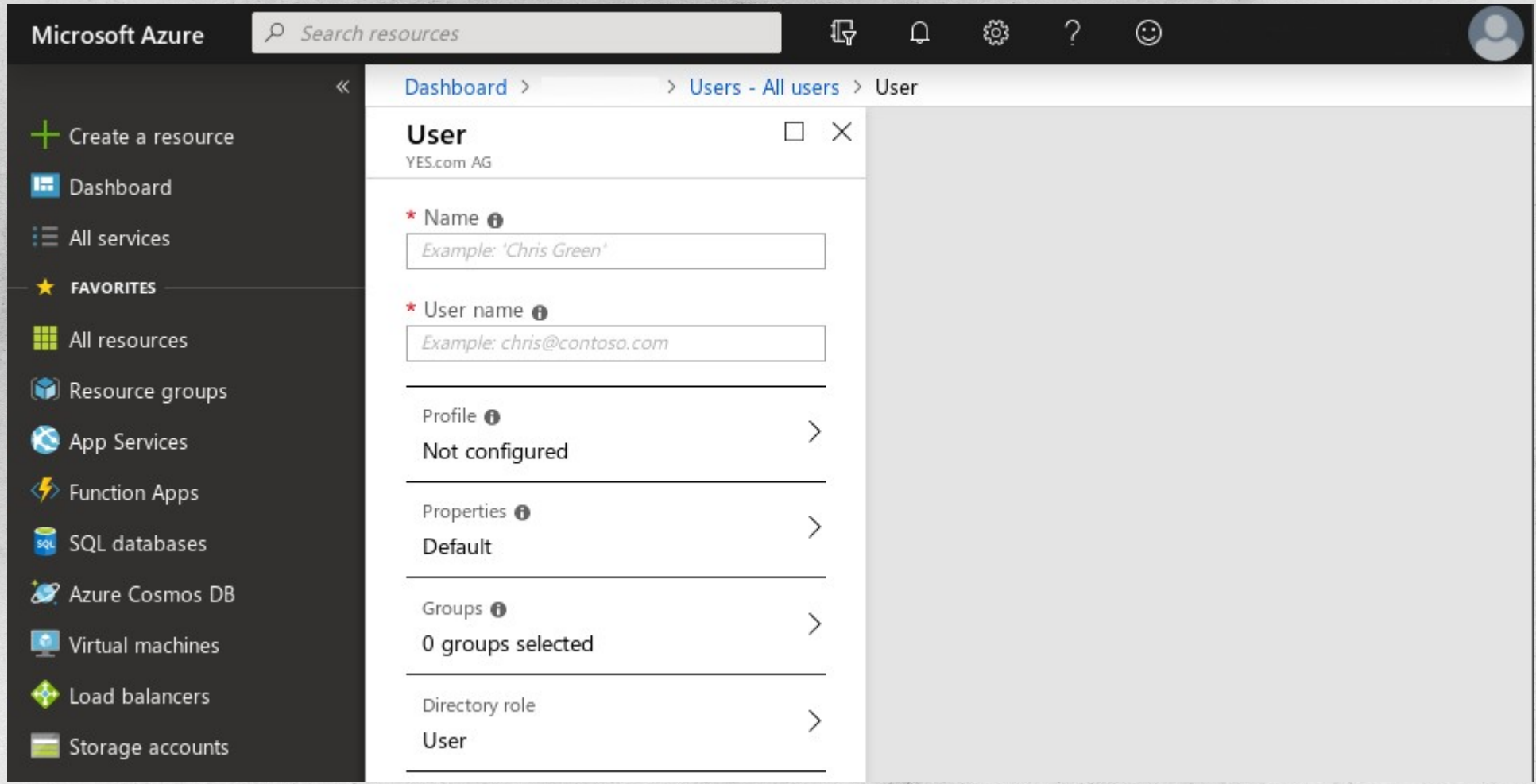
```yaml
initContainers:
  - name: "init-chown-data"
    image: "busybox:latest"
    imagePullPolicy: "IfNotPresent"
    resources: {}
    # 65534 is the nobody user that prometheus uses.
    command: ["chown", "-R", "65534:65534", "/data"]
    volumeMounts:
      - name: storage-volume
        mountPath: /data
        subPath: "prometheus-server"
```

```yaml
volumes:
  - name: config-volume
    configMap:
      name: monitoring-prometheus
  - name: storage-volume
    hostPath:
      path: /data/prometheus
```

Beispiele für eine Unit

```
resource "azurerm_managed_disk" "promdb_blockvolshared_serivces" {
  create_option         = "Empty"
  disk_size_gb          = 75
  location              = "${var.az_params["location"]}"
  name                  = "promdb.shared_services.${var.cluster_params["name"]}"
  resource_group_name   = "${module.kubernetes.cluster_resource_group_name}"
  storage_account_type = "Standard_LRS"
}
```

Beispiel für eine Unit?



Integrations-Test

```
[cjr@mccarthy:/home/cjr/]
$ curl -IL https://nasa.gov
HTTP/1.1 301 Moved Permanently
Date: Sat, 01 Jun 2019 11:47:34 GMT
Content-Type: text/html
Content-Length: 194
Connection: close
Location: https://www.nasa.gov/
Server: NASA
Strict-Transport-Security: max-age=31536000
```

```
HTTP/2 200
content-type: text/html
last-modified: Fri, 31 May 2019 19:37:23 GMT
server: AmazonS3
date: Sat, 01 Jun 2019 11:47:14 GMT
cache-control: max-age=300
age: 23
strict-transport-security: max-age=31536000
x-cache: Hit from cloudfront
via: 1.1 ad5f86bd8cf229b8836b7c71c182bcd2.cloudfront.net
(CloudFront)
```

Unit vs Integration Testing

- Unit tendenziell
  - Korrektheit des Code
  - Korrektheit des Algorithmus
- Integration tendenziell
  - Korrektheit des Verhaltens

Test-Scope

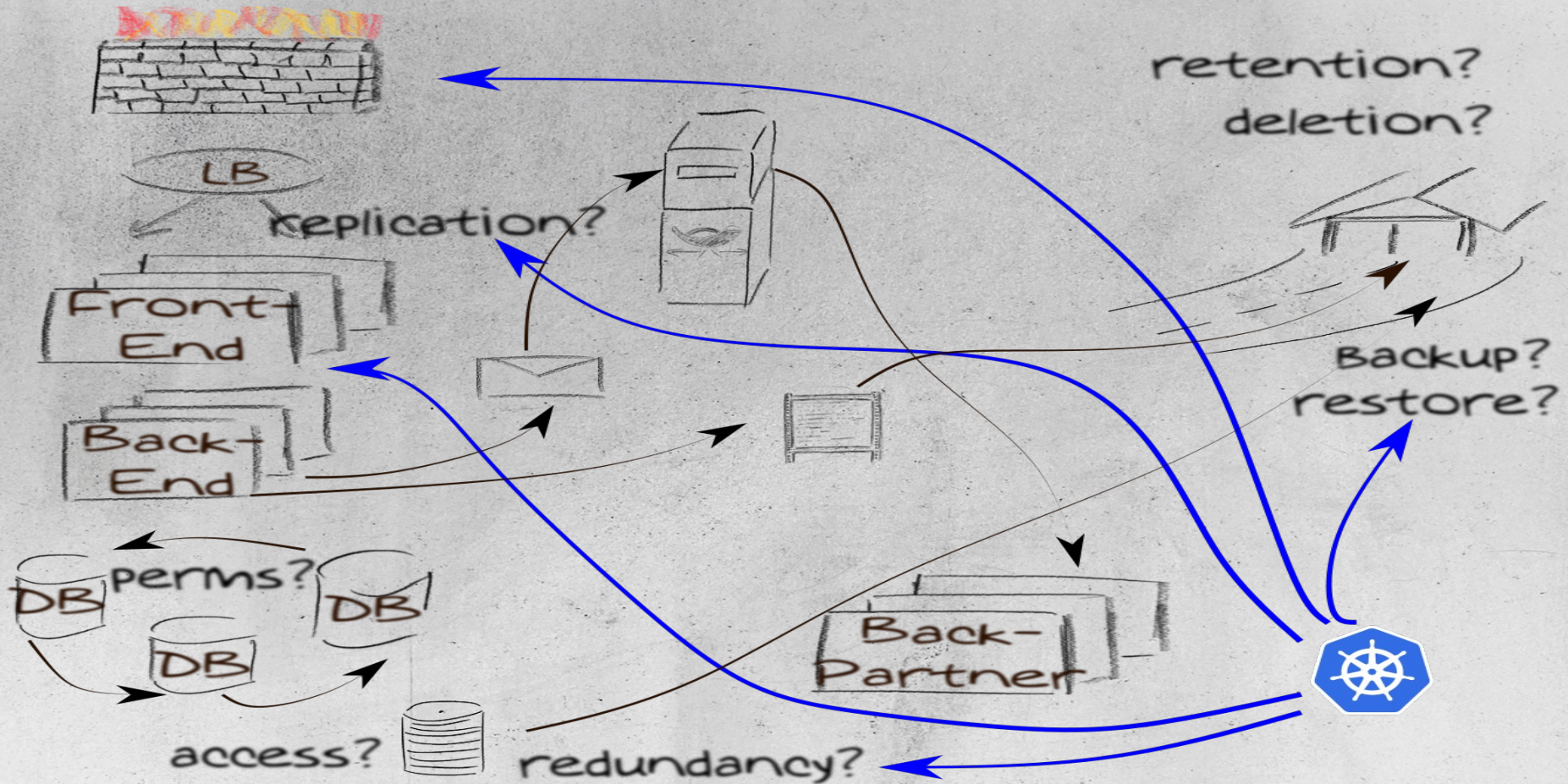
- Unit muß man vollständig unter Kontrolle haben

- wie ist das mit Fremdsoftware?
- wie ist das mit Cloud-Ressourcen?

Aussagekraft / Wertigkeit

- es nützt nix, wenn die Unit richtig ist, aber gleichzeitig die defaults des Providers ein völlig anderes Verhalten bewirken
- es nützt aber auch nix, wenn das Verhalten nur zufälligerweise richtig ist

Deployment


LB
replication?
Front-End
Back-End
DB
perms?
DB
DB
access?
redundancy?
Back-Partner
retention?
deletion?
backup?
restore?

Fallunterscheidungen

- Images
  - atomar
  - at Build time
- Container
  - atomar?
  - at compile time
- Cloud
  - ???

ServerSpec

```ruby
require 'spec_helper'

describe package('ssh'), :if => os[:family] == 'ubuntu' do
  it { should be_installed }
end

describe service('ssh'), :if => os[:family] == 'ubuntu' do
  it { should be_enabled }
  it { should be_running }
end

describe port(22) do
  it { should be_listening }
end
```

ServerSpec

- bond, bridge, default_gateway, ip6tables, ipfilter, ipnat, iptables, routing_table , interface, host
- cgroup, docker_container, docker_image, lxc, port
- kernel_module | linux_audit_system | linux_kernel_parameter
- package, ppa, yumrepo
- service, cron, process, selinux, selinux_module
- file, x509_certificate, x509_private_key
- group, user, mail_alias
- command

ServerSpec - Types

```
module Serverspec::Type
  class Host < Base
    def resolvable?(type)
      @runner.check_host_is_resolvable(@name, type)
    end

    def reachable?(port, proto, timeout)
      @runner.check_host_is_reachable(@name, port, proto, timeout)
    end

    def ipaddress
      @runner.get_host_ipaddress(@name).stdout.strip
    end
    def ipv4_address
      @runner.get_host_ipv4_address(@name).stdout.strip
    end
    def ipv6_address
      @runner.get_host_ipv6_address(@name).stdout.strip
    end
  end
end
```

ServerSpec - Runners

```
class Specinfra::Command::Aix::Base::Host < Specinfra::Command::Base::Host
  class << self
    def check_is_resolvable(name, type)
      if type == "dns"
        %Q[lookup=$(nslookup -timeout=1 #{escape(name)} | grep -A1 'Name:' | grep
Address | awk -F': ' '{print $2}'); if [ "$lookup" ]; then $(exit 0); else $(exit 1);
fi]
      elsif type == "hosts"
        "grep -w -- #{escape(name)} /etc/hosts"
      else
        "host #{escape(name)}"
      end
    end

    def get_ipaddress(name)
      "host #{escape(name)} | awk '{print $3}'"
    end
  end
end
```

ServerSpec - Matchers

- wie RSpec
  - exist, match, start_with, ...
- spezifische
  - be_mounted, contain, be_writable
- eigene kann man dann selbst schreiben

InSpec - Control

```
title 'Check webserver user'

control 'nginx-user?' do
  impact 0.7
  title 'Does the user for the webserver exist?'
  desc 'An optional description...'

  describe user('www-data') do
    it { should exist }
    its('uid') { should eq 33 }
    its('gid') { should eq 33 }
    its('group') { should eq 'www-data' }
    its('groups') { should eq ['www-data'] }
    its('home') { should eq '/var/www' }
    its('shell') { should eq '/usr/sbin/nologin' }
  end
end
```

InSpec - Resources

- OS:
  - apt, cpan, cran, ..., auditd, ..., Bond, Bridge, ..., Bsd_service, ...
- AWS
- Azure
- GCP

InSpec Control

```
├── cache
│
├── config.json
├── controls
│   └── arm_yescom_spec.rb
│
├── inspec.lock
├── inspec.yml
├── libraries
│   ├── arm_custom_matchers.rb
│   └── azure_functors.rb
├── out
│   ├── example.json
│   ├── index.html
│   ├── junit.xml
│   ├── report.json
│   └── report.txt
└── vendor
    ├── 876c6d8941 5eaf433796c71738a2934 64da21260aec571fd16d66cea436b25eeb
    └── d17bf8c442edled48446 5edbee483ffc2a80682b12f46512a07d42f5adb71d4f
```

InSpec - Custom Matcher

```ruby
RSpec::Matchers.define :include_all_arm_res_in do |expected_a|
  missing_a = []

  match do |actual_obj|
    left_a, criterion = expected_a
    right_a = AzureRMSets::ARMSetFunc.unwrap(actual_obj, criterion)

    missing_a = AzureRMSets::ARMSetFunc.set_difference(left_a, right_a, true)
    missing_a.empty?
  end

  failure_message do
    error_s = RSpec::Matchers::EnglishPhrasing.list(missing_a)
    "Resource(s) #{error_s} *should* be present."
  end

  match_when_negated do # |actual_obj|
  end

  failure_message_when_negated do
  end
end
```

hadolint - keine fragwürdigen Konstrukte

- DL3001: vim, shutdown in container?
- DL3009: delete apt-get lists
- DL3011: valid port range 0 < p < 65535
- SC1078: did you forget to close this double-quoted string?

Unit-Tests auf Dockerfiles

```
[cjr@mccarthy:/home/cjr/]

$ curl -L
https://raw.githubusercontent.com/docker-library/mysql/26380f33a0fcd07dda35e37516eb24eaf962845c/5.7/
Dockerfile \

  | hadolint -


/dev/stdin:6 DL3008 Pin versions in apt get install. Instead of `apt-get install <package>` use `apt-
get install <package>=<version>`

/dev/stdin:10 SC2155 Declare and assign separately to avoid masking return values.

/dev/stdin:10 DL3008 Pin versions in apt get install. Instead of `apt-get install <package>` use `apt-
get install <package>=<version>`

/dev/stdin:25 DL3008 Pin versions in apt get install. Instead of `apt-get install <package>` use `apt-
get install <package>=<version>`

/dev/stdin:38 SC2155 Declare and assign separately to avoid masking return values.

/dev/stdin:55 SC2028 echo may not expand escape sequences. Use printf.

/dev/stdin:55 DL3015 Avoid additional packages by specifying `--no-install-recommends`

/dev/stdin:55 DL4006 Set the SHELL option -o pipefail before RUN with a pipe in it
```

Integration Test auf Images

- statische Checks
  - file existence, file content, metadata, license
- dynamische Checks
  - command / retval, std{out,err}

Integration-Tests auf Images

```
$ container-structure-test test --image debian:buster  --config structure.yaml


=== RUN: Command Test: PATH

--- FAIL

stdout: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Error: Expected string '.*gem/ruby/.*/bin' not found in output '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'


=== RUN: File Existence Test: Root

--- FAIL

Error: / has incorrect permissions. Expected: -rw-r--r--, Actual: drwxr-xr-x

Error: / has incorrect user ownership. Expected: 1000, Actual: 0

Error: / has incorrect group ownership. Expected: 1000, Actual: 0


=== RUN: Metadata Test

--- FAIL

Error: env var PATH value /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin does not match expected value: .*gem/ruby/
2.5.0/bin

Error: Image entrypoint [] does not match expected entrypoint: [inspec]

Error: Image workdir  does not match config workdir: /home/inspector
```

GOSS

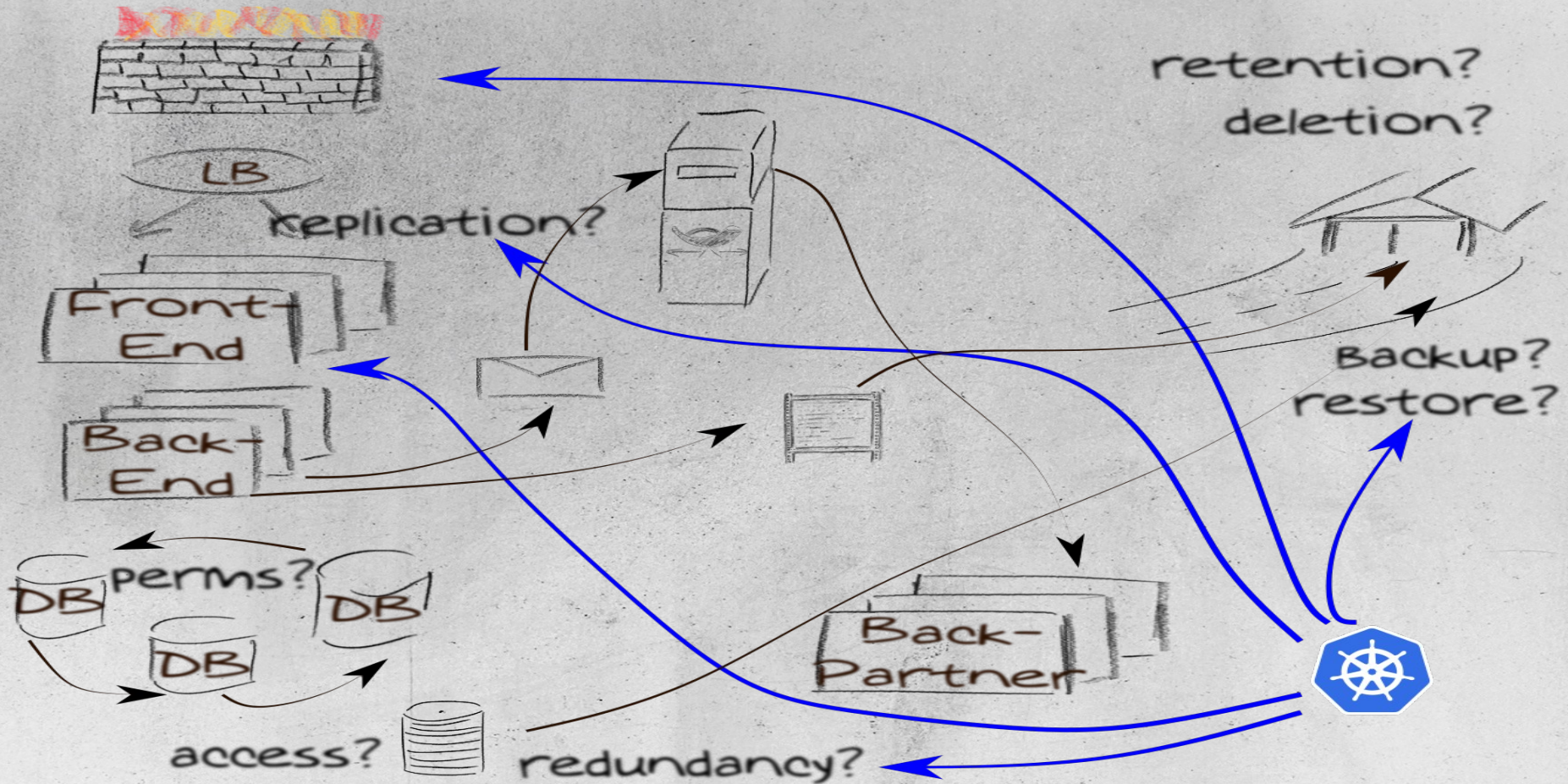
- ServerSpec als yaml
- Test-Erzeugung aus Bekanntem / vorhandenen Stage (goss autoadd ssh)
- Test-Aufbauten aus docker und docker-compose

Problem?

@DEVOPS_BORAT

- In Startup, we have great capability for churn out solution. Please send problem, we are pay good money.

Deployment


LB
replication?
Front-End
Back-End
DB
perms?
DB
DB
access?
redundancy?
Back-Partner
retention?
deletion?
backup?
restore?

Fehler finden?

- LoadBalancer:
  - Balanciert nicht: warum? LB oder Back falsch!
  - session stickyness: falsche Annahme zu Verteilungs-Eigenschaften!
- Ansatzpunkte:
  - Traffic permanent überwachen und vergleichen

Fehler finden?

- Network:
  - Ganz Getrennt
  - Traffic Blockiert
- Ansatzpunkte:
  - Traffic permanent überwachen und vergleichen

Fehler finden?

- Permissions / Access:
  - config-Fehler
- Ansatzpunkte:
  - die **geschützten Objekte** prüfen, nicht die Aktoren

Fehler finden?

- Retention / Deletion
  - config-Fehler
  - wenn man denn die korrekten Werte überhaupt kennt
- Ansatzpunkte:
  - die **Objekte** prüfen, nicht die Aktoren

Wo entseht der Fehler?

- auf OSI-8
  - direkte Fehlkonfiguration
  - Änderungen bei Fremdbezug ohne Kommunikation
  - Zusammenhänge nicht verstanden haben
- Fehler erkennen
  - post-Deployment
  - kontinuierlich

Vorteil durch Testen?

- man hat es **spezifiziert** / aufgeschrieben
- man kann es dann permanent prüfen

- permant riecht nach Monitoring!
- **es ist der Input fürs Monitoring!**

Fazit

- Container und Cloud haben nicht viel Logik -> Unit-Tests lohnen nicht
- Container- / Cloud-Orchestierte Umgebungen sind volatil -> korrektes Verhalten muß ständig geprüft werden

Fazit

- Testen um des Testens willen ist unsinnig
- es muß zielgenau in Kenntnis der Schwachstellen der eigenen Dynamik implementiert sein
- das ist fast immer Compliance / Security ! (ich sehe hier keine harte Trennung)

Gute Tests

- DevSec (https://dev-sec.io/baselines/)
- CIS-Benchmarks -> InSpec

- Last-Tests!
- Das Blatt Papier!