10 YEARS OF CLOUD

PIERRE-YVES RITSCHARD (@PYR)

@pyr: Three-line Bio

- CTO & Co-founder at Exoscale
- Distributed systems and monitoring enthusiast
- Open-Source developer

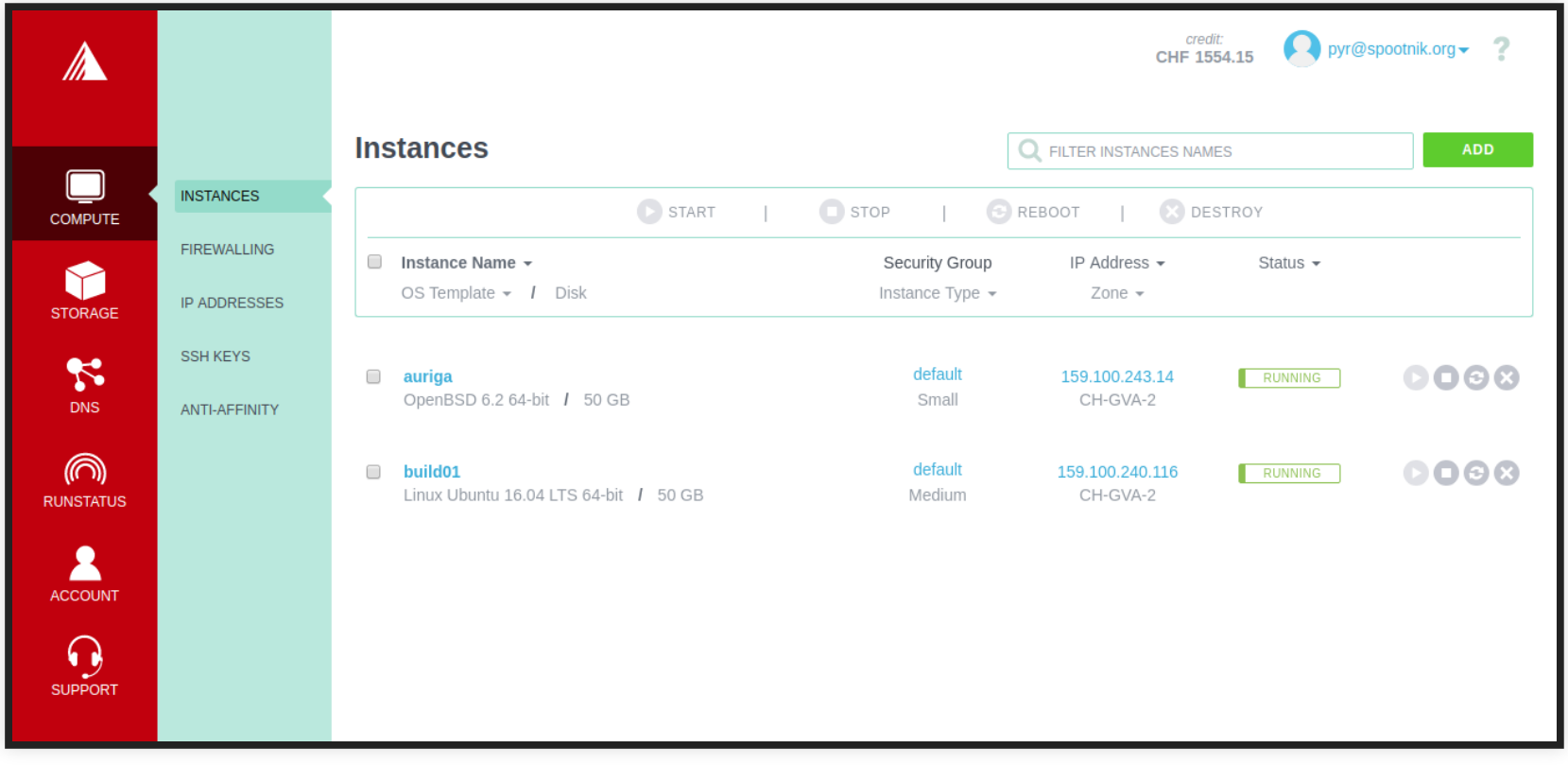
10 YEARS OF CLOUD

Building better infrastructure with open source

EXOSCALE

- Infrastructure as a service
- Zones in Frankfurt, Vienna, Zürich, Geneva, Münich, Sofia

EXOSCALE


credit:
CHF 1554.15
pyr@spootnik.org
COMPUTE
STORAGE
DNS
RUNSTATUS
ACCOUNT
SUPPORT
INSTANCES
FIREWALLING
IP ADDRESSES
SSH KEYS
ANTI-AFFINITY
Instances
FILTER INSTANCES NAMES
ADD
START
STOP
REBOOT
DESTROY
Instance Name
Security Group
IP Address
Status
OS Template
Disk
Instance Type
Zone
auriga
OpenBSD 6.2 64-bit / 50 GB
default
Small
159.100.243.14
CH-GVA-2
RUNNING
build01
Linux Ubuntu 16.04 LTS 64-bit / 50 GB
default
Medium
159.100.240.116
CH-GVA-2
RUNNING

EXOSCALE

```
provider "exoscale" {
  api_key    = "${var.exoscale_api_key}"
  secret_key = "${var.exoscale_secret_key}"
}

resource "exoscale_instance" "web" {
  template  = "Ubuntu 21.04"
  disk_size = "50g"
  profile   = "medium"
  ssh_key   = "production"
}
```

THIS WAS SUPPOSED TO BE A TALK ON OPEN SOURCE!

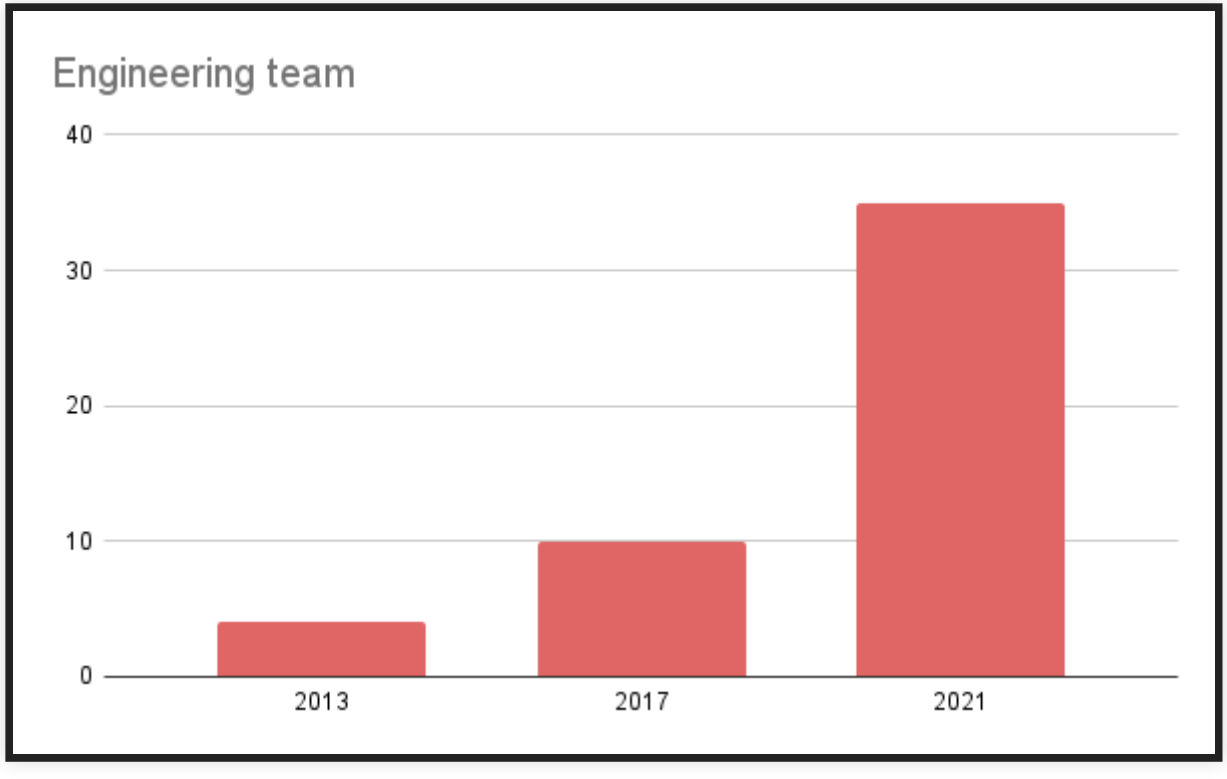
WHAT'S IN A CLOUD PROVIDER

- Datacenter operations
- Software development

SOFTWARE AT EXOSCALE

- API Gateway
- Orchestrators (VM instance, Load-Balancer, Kubernetes)
- Object storage controller
- Network controller (SDN)
- Customer management
- Metering system
- Billing
- Web portal

ENGINEERING AT EXOSCALE



ISN'T ALL OF THIS BASH, PERL, AND YAML?

STARTUPS AND OPEN SOURCE

- Ability to quickly make changes at any position of the stack
- Ability to tailor software to specific needs
- There is a long relationship between core infrastructure and Open Source

OPEN SOURCE AT EXOSCALE: A TIMELINE

2012: THE EARLY DAYS

We started with

- 3 people
- A bit of time
- A product idea

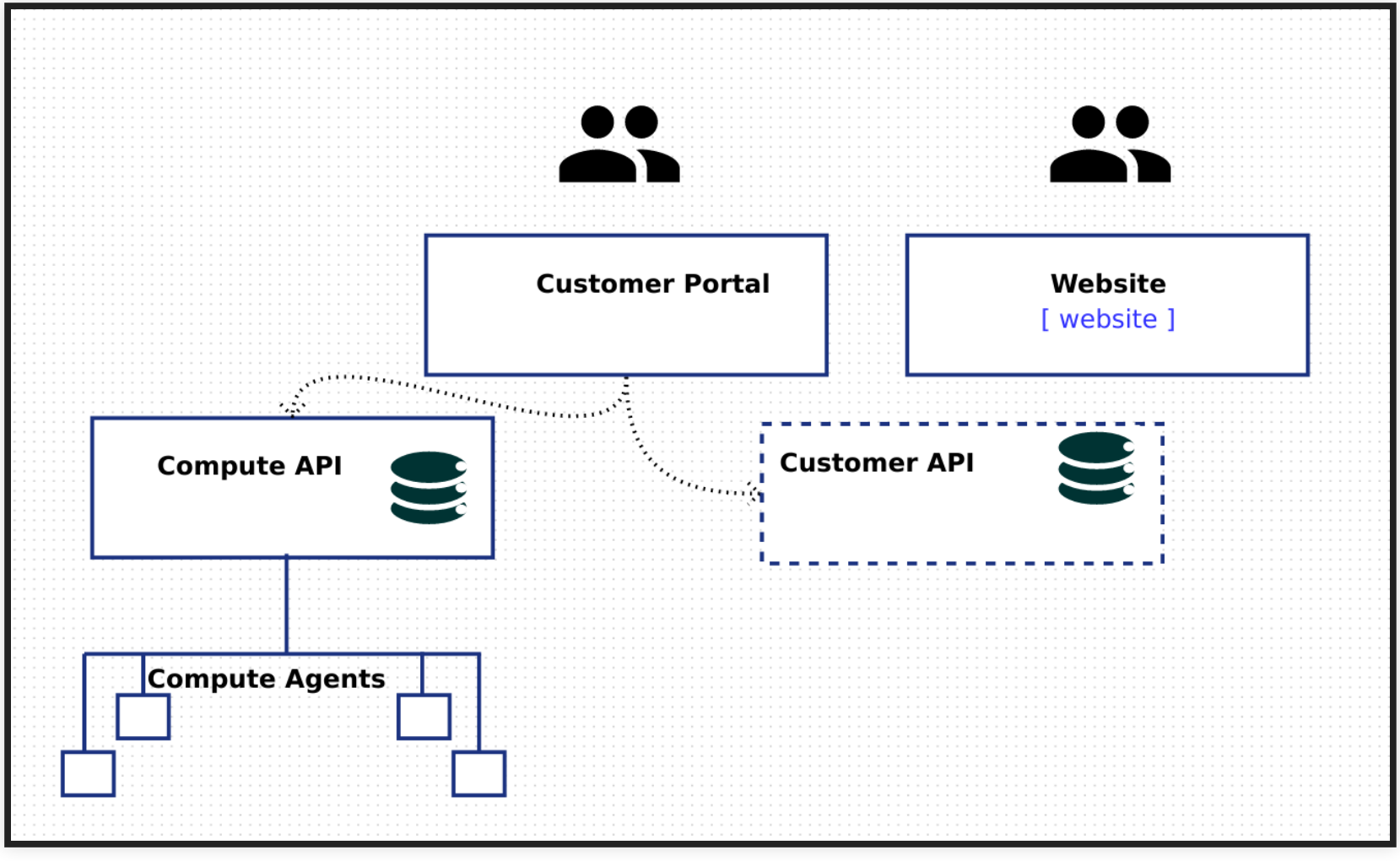
A DIFFERENT CLOUD PROVIDER

- Not yet another virtual datacenter product
- Integration with automation tooling
- Integration in language-specific libraries
- Focus on horizontally-scalable applications
  - Local storage
  - Security groups

OUR MINIMAL STACK

- Apache Cloudstack
- Puppet
- Good old MySQL
- An in-house customer management tool
- Python + AngularJS
- Riemann

OUR MINIMAL STACK

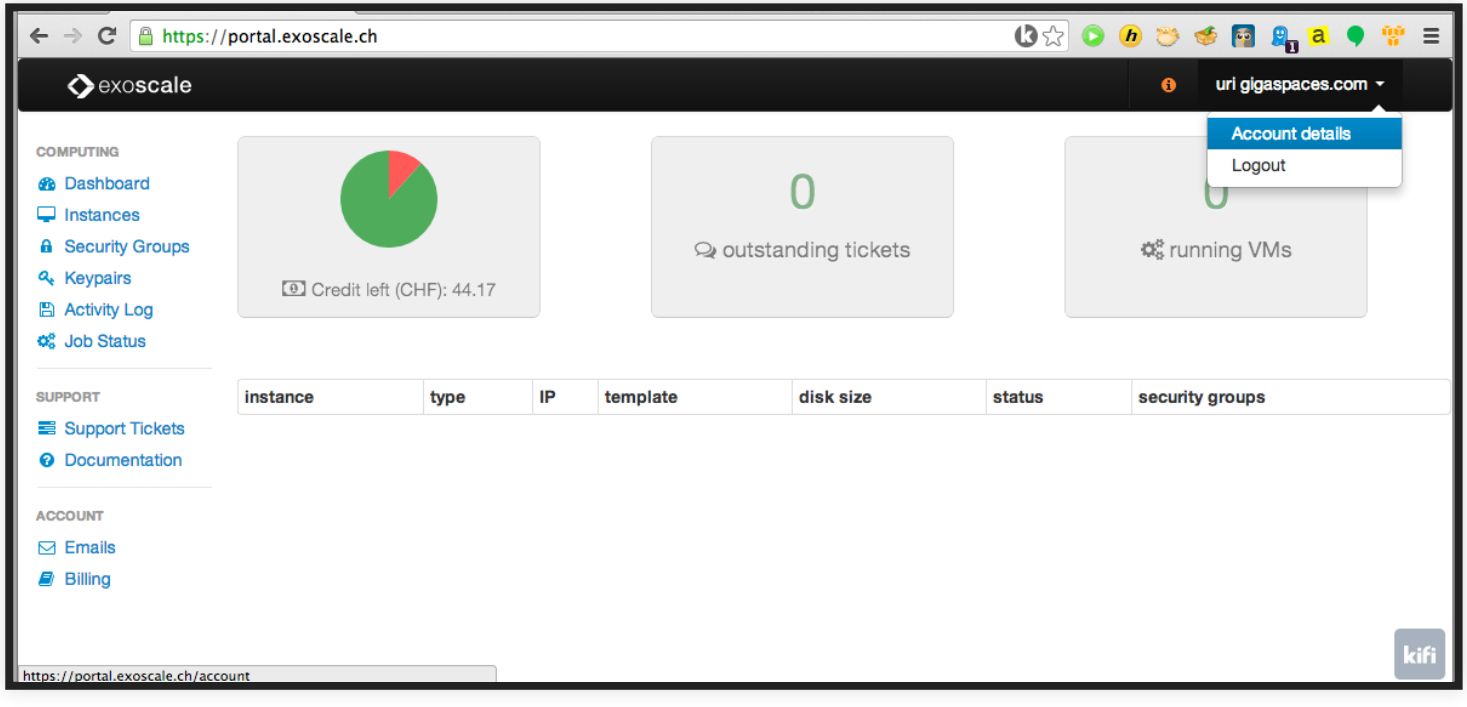


RIEMANN

- The common saying back then was **monitoring sucks**
- Push-based model was a great fit for our use case
- A great opportunity to contribute, as Riemann was in early stages
  - Submitted many fixes
  - Quickly became core maintainers

2013: GOING LIVE

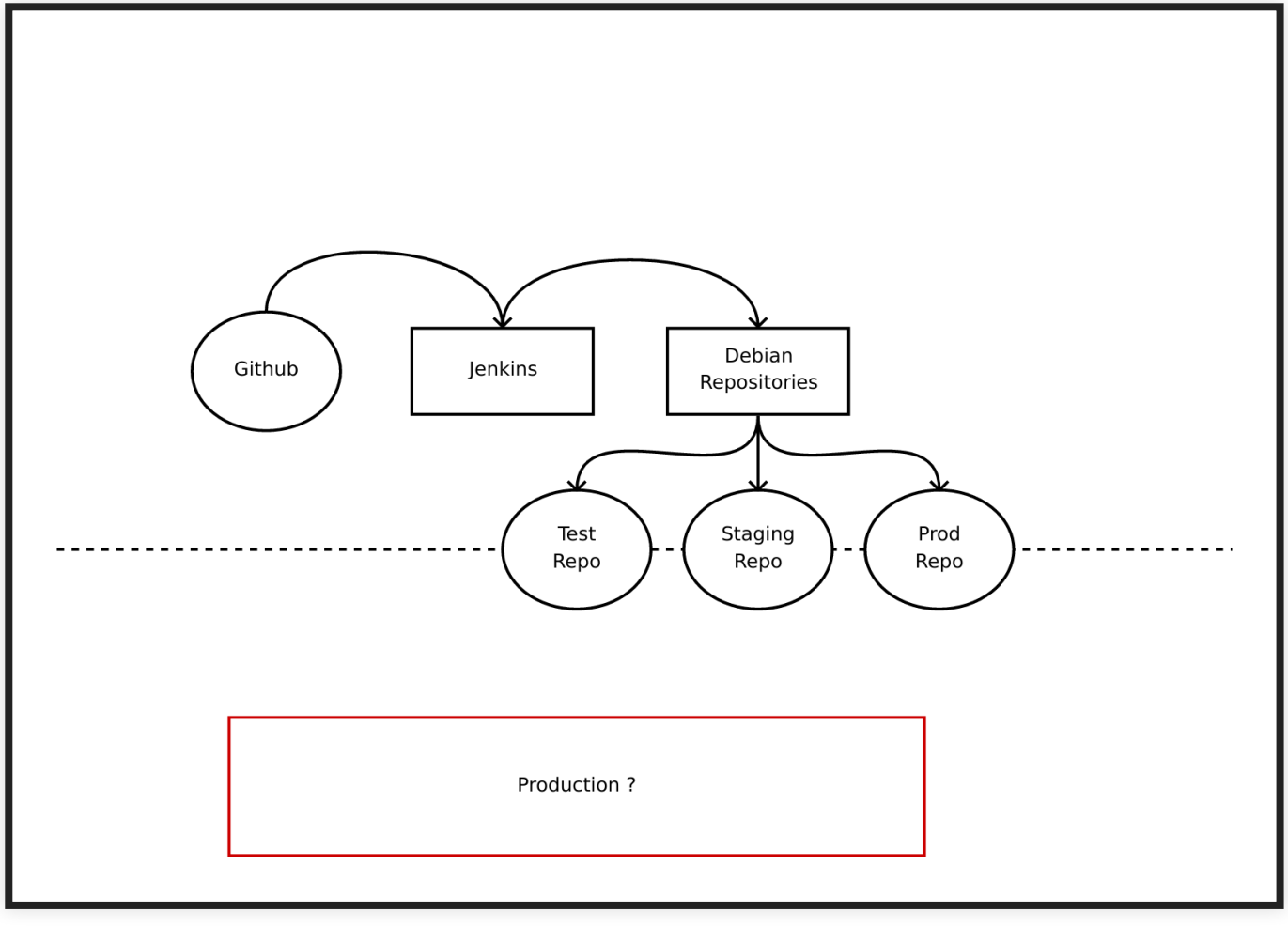
BACKEND DEVELOPERS DOING FRONTEND



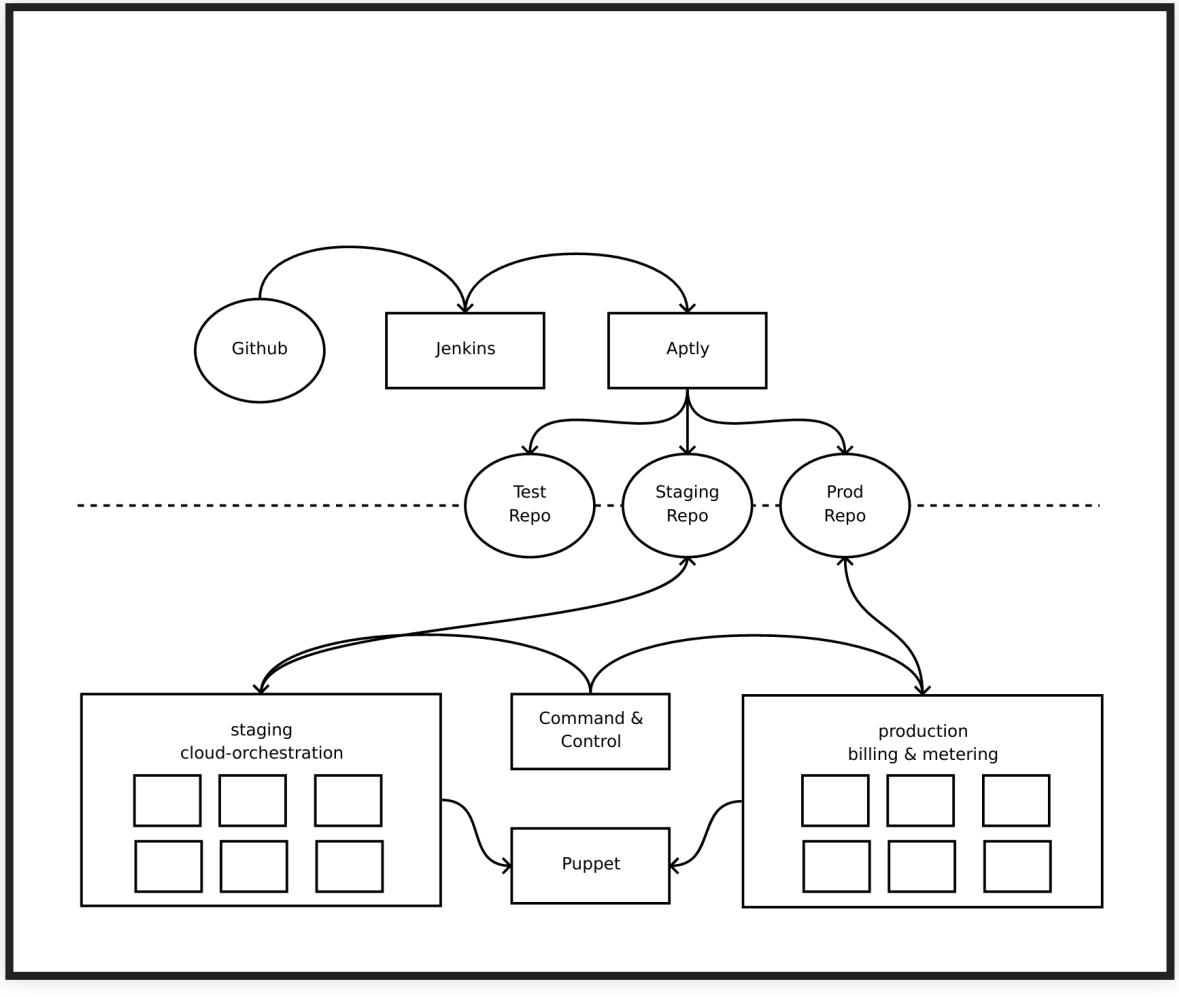
THINGS OUR EARLY ADOPTERS ENJOYED

- Vagrant support
- Security groups instead of firewalling
- A public IP per instance

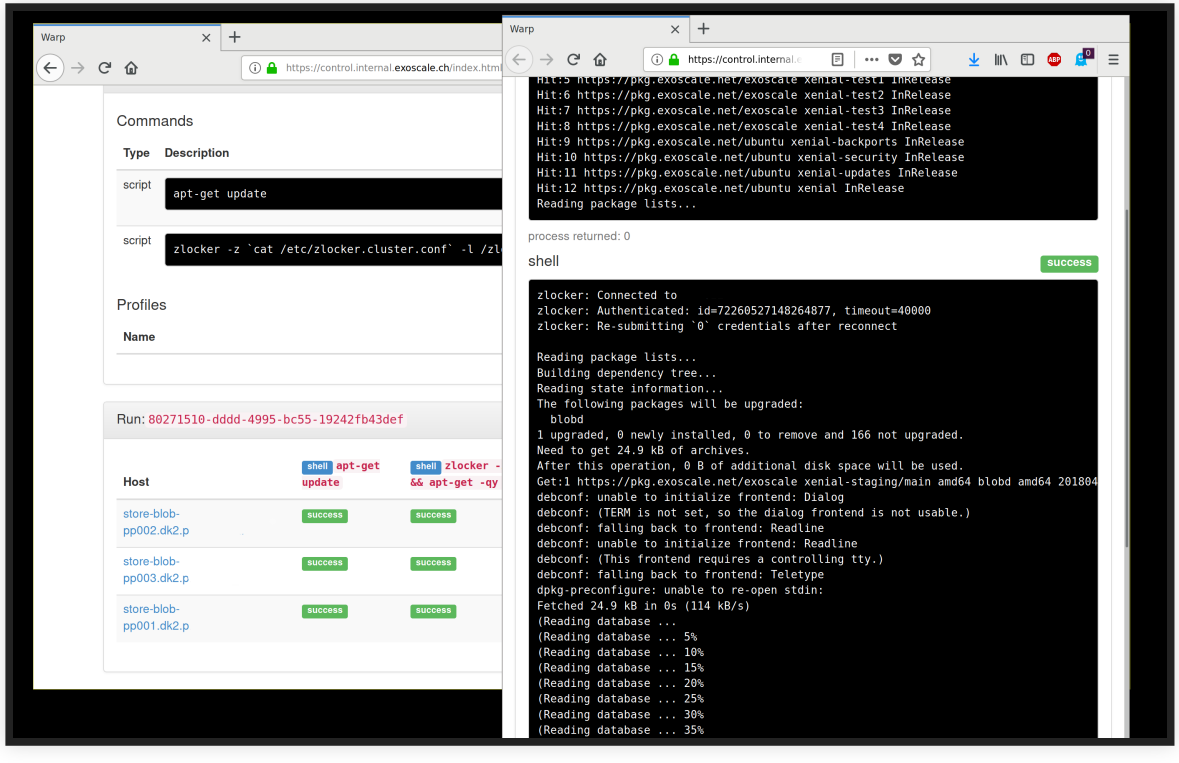
IMPROVING RELEASE AUTOMATION



WARP


Github
Jenkins
Aptly
Test Repo
Staging Repo
Prod Repo
staging cloud-orchestration
Command & Control
production billing & metering
Puppet

WARP


Warp
https://control.internal.exoscale.ch/index.html
Commands
Type
Description
script
apt-get update
script
zlocker -z `cat /etc/zlocker.cluster.conf` -l /zl
Profiles
Name
Run: 80271510-dddd-4995-bc55-19242fb43def
Host
shell apt-get update
shell zlocker - && apt-get -qy
store-blob-pp002.dk2.p
store-blob-pp003.dk2.p
store-blob-pp001.dk2.p
success
Warp
https://control.internal.e
Hit:6 https://pkg.exoscale.net/exoscale xenial-test2 InRelease
Hit:7 https://pkg.exoscale.net/exoscale xenial-test3 InRelease
Hit:8 https://pkg.exoscale.net/exoscale xenial-test4 InRelease
Hit:9 https://pkg.exoscale.net/ubuntu xenial-backports InRelease
Hit:10 https://pkg.exoscale.net/ubuntu xenial-security InRelease
Hit:11 https://pkg.exoscale.net/ubuntu xenial-updates InRelease
Hit:12 https://pkg.exoscale.net/ubuntu xenial InRelease
Reading package lists...
process returned: 0
shell
success
zlocker: Connected to
zlocker: Authenticated: id=72260527148264877, timeout=40000
zlocker: Re-submitting `0` credentials after reconnect
Reading package lists...
Building dependency tree...
Reading state information...
The following packages will be upgraded:
blobd
1 upgraded, 0 newly installed, 0 to remove and 166 not upgraded.
Need to get 24.9 kB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 https://pkg.exoscale.net/exoscale xenial-staging/main amd64 blobd amd64 201804
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Fetched 24.9 kB in 0s (114 kB/s)
(Reading database ...
(Reading database ... 5%
(Reading database ... 10%
(Reading database ... 15%
(Reading database ... 20%
(Reading database ... 25%
(Reading database ... 30%
(Reading database ... 35%

WARP

- Open Source
- TLS client certificate-based authentication
- IRC support
- ~~Haskell~~ Go agent
- Prefigured our inclination for Clojure at the orchestration layer

2014: THE YEAR OF STORAGE

OBJECT STORAGE

- The obvious choice for our crowd
- Architecturally simpler than distributed block storage
- A good complement to our local storage backed instances

OBJECT STORAGE NEEDS

- S3 is the sole player in that field: we need API compatibility
- The only alternative at the time was bad HTTP extensions

OBJECT STORAGE IN THE WILD

- Ceph
- Riak-CS
- Swift
- Costly vendor-backed solutions

WRITING AN OBJECT STORE

- We focused on how to store large objects
- Tempted by a description of the (non-OpenSource) approach by Datastax on top of Cassandra

CHOOSING CASSANDRA

- Great library support
- Simple for us to operate
- Very few moving parts
- Our implementation could remain fully stateless

WE WERE (ALMOST) YOUNG AND (WAY TOO) NAIVE

How are could it be?

WHAT WE DIDN'T ANTICIPATE

It's not all about actual data storage

- The S3 API is a beast
- The S3 API is under specified
- The S3 API is not versioned
- The S3 API client landscape is a mess

A QUICK DIGRESSION: S3 REQUESTS

Operation: put object `foo` in bucket `bar`:

```
PUT /foo
Host bar.sos-ch-dk-2.exo.io
Authorization: AWS ....

<...>
```

A QUICK DIGRESSION: S3 REQUESTS

Operation: update acl for object `foo` in bucket `bar`:

```
PUT /foo?acl
Host bar.sos-ch-dk-2.exo.io
Authorization: AWS ....
X-Amz-ACL: bucket-owner-full-control
```

A QUICK DIGRESSION: S3 REQUESTS

Operation: Copy object `bim` from bucket `bam` to object `foo` in bucket `bar`:

```
PUT /foo
Host bar.sos-ch-dk-2.exo.io
Authorization: AWS ....
X-Amz-Copy-Source: /bim/bam
X-Amz-Copy-Source-If-Unmodified-Since: ARE YOU KIDDING ME?
```

BY THE WAY

- Storing terrabytes of data on off-the-shelf hardware doesn't come by easy either
- Input and output payloads of arbitrary lengths aren't easy
- The standard web stack doesn't cut it

2015: SCALING UP

GROWING PAINS

- More customers mean more logs, metrics
- We also grew the amount of data points we were tracking
- The standard ELK stack shows its limits

ELK ISSUES

- Logstash takes a substantial amount of resources on every node
- Going directly to elastic ties the availability of the logging infrastructure

OPEN SOURCE TO THE RESCUE

- Syslog-NG already on all our compute nodes
- Collectd as well
- Nascent Kafka usage in other parts of our stack

IMPROVING SYSLOG-NG

```
destination {
  kafka(
    config(
      "metadata.broker.list" => "kafka:9094"
      "security.protocol" => "SSL"
      "ssl.key.location" => "/etc/ssl/kafka.key"
      "ssl.certificate.location" => "/etc/ssl/kafka.pem"
      "ssl.ca.location" => "/etc/ssl/ca.pem"
      "queue.buffering.max.ms" => "1000"
     )
     topic("logs")
     template("${.JSON_SANITIZED}")
  );
};
```

IMPROVING COLLECTD

```
LoadPlugin write_kafka
<Plugin write_kafka >
  Property "metadata.broker.list" "kafka:9094"
  Property "security.protocol" "SSL"
  Property "ssl.ca.location" "/etc/ssl/ca.pem"
  Property "ssl.certificate.location" "/etc/ssl/kafka.pem"
  Property "ssl.key.location" "/etc/ssl/kafka.key"
  <Topic "metrics">
    GraphitePrefix "collectd."
    GraphiteEscapeChar "."
    StoreRates true
    GraphiteAlwaysAppendDS false
    Key "${hostname}"
    Format "Graphite"
  </Topic>
</Plugin>
```

IMPROVING COLLECTD WORK

- Direct riemann sink
- Local aggregations for relevant metrics

2017: TOO MUCH DATA

SUDDEN S3 PICKUP IN USAGE

- Our initial implementation limits the throughput
- Tail latencies go through the roof
- Cassandra is just not great at doing dense nodes
  - We knew this going in
  - We hit the wall hard

WE NEED A NUMBER OF NEW API CAPABILITIES

- V4 signatures are becoming the norm for S3
- Better ACL support is needed
- The docker registry exercises all weird properties of the API

WE FIND A GOOD PAPER

- Ambry attacks the same problem space
- The paper lays out a great strategy

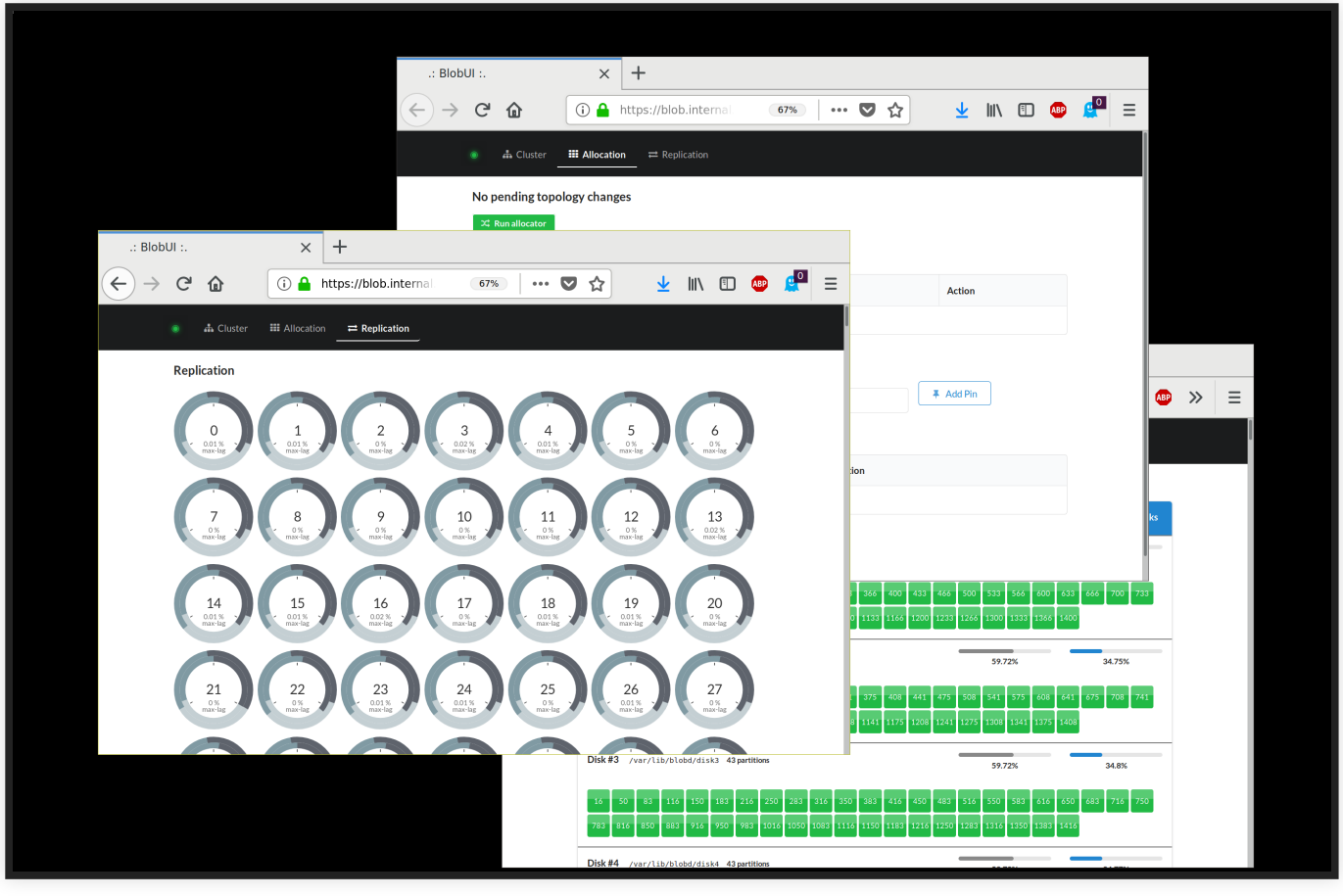
LET'S WRITE A DISTRIBUTED SYSTEM FROM SCRATCH

What could go wrong?

KEY DECISIONS

- A storage agent in C
- Orchestration in Clojure
- Zookeeper for agent discovery
- Cassandra for metadata storage

UI


.: BlobUI :.
https://blob.internal
67%
Cluster
Allocation
Replication
No pending topology changes
Run allocator
Action
Add Pin
Replication
Disk #3 /var/lib/blobd/disk3 43 partitions
Disk #4 /var/lib/blobd/disk4 43 partitions

OPEN SOURCE STATUS

- Still unreleased
- Will happen with our next large batch

2018: SECURITY, API, AND KUBERNETES

SPECTRE AND MELTDOWN

- Established crisis communication channels with other key providers
  - Relying on vanilla open source helps
- Large scale automation of Linux Kernel roll-outs

BUILDING ON KUBERNETES

- We previously bet on Mesos
- Traction seemed to be stronger around Kubernetes
- Other players were still relevant (notably Swarm)
- Need to gain knowledge before being ready to sell

KUBERNETES EXTENSIONS

- Kubernetes autoscaler
- Kubernetes cloud controller manager
- Kubernetes external DNS support

2019: RETHINKING OUR CORE ORCHESTRATION FACILITIES

NEW ORCHESTRATOR NEEDS

- NLB
- Kubernetes
- more to come

ORCHESTRATOR REDUX

- Flexible data layer access
- Finite state machines to keep state transitions under control
- Workflow engines to perform side-effects

FLEXIBLE DATA LAYER ACCESS

Controlling the mapping betwen row and column names in the database

Specific table names can be provided by using a vector as the argument for `entity` or `entity-from-spec`:

```
(make-schema
  (entity-from-spec [::invoice-line/invoice-line :invoiceline]
    ...))
```

Specific column names can be provided by using the `column-name` helper:

```
(make-schema
  (entity-from-spec ::network/network
    (column-name :ip6address :ip6)
    ...))
```

Mutations

With querying sorted, mutations need to be expressed. Here, **seql** takes the approach of making mutations separate, explictit, and validated. As with most other **seql** features, mutations are implemented with a key inside the entity description.

At its core, mutations expect two things:

- A **spec** of their input
- A function of this input which must yield a proper **honeysql** query map, or collection of **honeysql** query map to be performed in a transaction.

For the common case of inserting, updating, or deleting records from the database, a couple of schema helpers are provided.

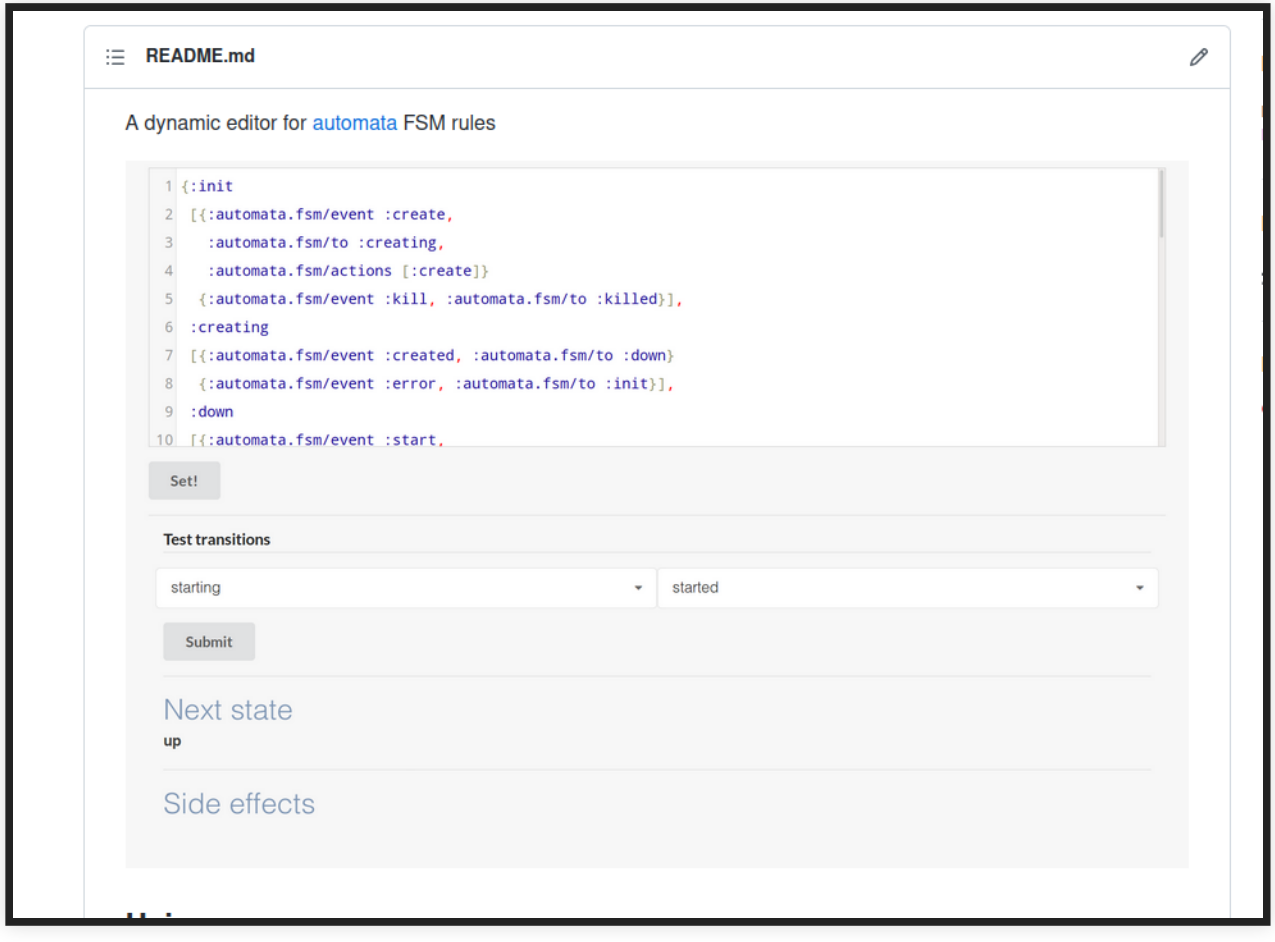
Inserting records with `add-create-mutation`

To allow record insertion, use the `add-create-mutation` helper:

```
(entity-from-spec ::account/account
          (has-many :users    [:id ::user/account-id])
          (has-many :invoices [:id ::invoice/account-id])
          (add-create-mutation))
```

https://github.com/exoscale/seql

FINITE STATE MACHINES



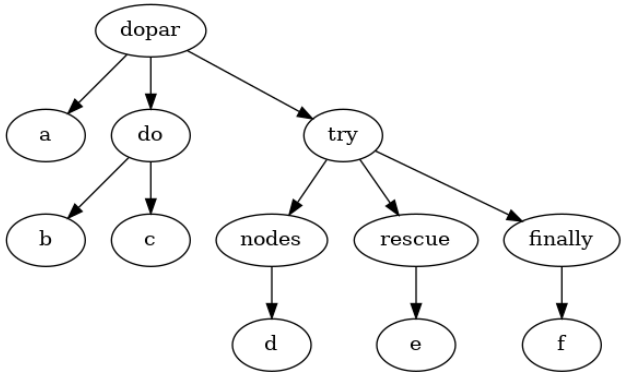
https://github.com/exoscale/automata

WORKFLOW ENGINES

So a simple program like:

```
(dopar!!
  (log!! "a")
  (do!!
    (log!! "b")
      (log!! "c"))
  (try!!
    (log!! "d")
      (rescue!! (log!! "e"))
      (finally!! (log!! "f"))))
```

Would result in the following tree:



This is what `ablauf.ast` provides, with a corresponding spec.

Job execution

Now that a simplistic but sufficient AST exists, comes the question of its execution. It would be trivial to walk the above tree and execute things as they are found. The notion of parallel nodes makes things a bit less obvious.

https://github.com/exoscale/ablauf

LOOKING BACK

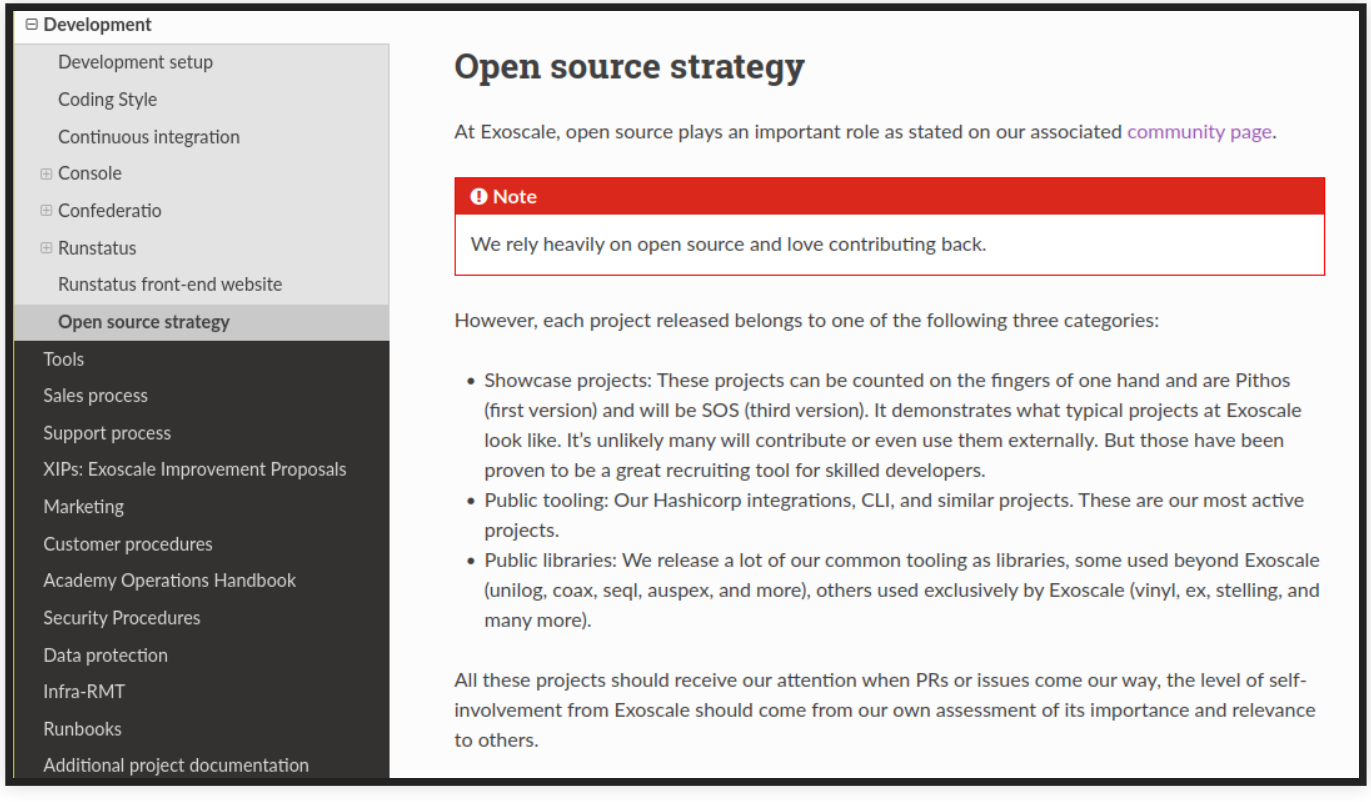
OPEN-SOURCE AT EXOSCALE

- Apache Cloudstack was a great way to bootstrap our service
- High dependence on Open Source Databases
- Some key low level components (Qemu, Libvirt)

OPEN-SOURCE INVOLVEMENT

- Several lighthouse projects: the Exoscale CLI, Pithos, Cyanite
- Several Apache developers
- Key contributions to major projects
- Integration in key infrastructure projects: Terraform, Packer, Vault
- Numerous core libraries open sourced

ENGINEERS LOVE GOOD DOCUMENTATION



THANKS

- Questions