O'REILLY®

Velocity

CONFERENCE

BUILD RESILIENT SYSTEMS AT SCALE

velocityconf.com/eu

#velocityconf

A real-life account of moving 100% to a public cloud

Antoine Guy, Principal Infrastructure Architect
Julien Simon, Chief Technology Officer

A word about us

Julien Simon

Last 10 years as VP Eng / CTO
in web startups (Digiplug, Pixmania, Criteo, Viadeo)

Disclaimer : I joined AWS two weeks ago, but don’t worry, this won’t be a sales pitch :D

Email: julsimon@amazon.com
Twitter: @julsimon

Antoine Guy

14 years of infrastructure & web ops

Tech lead for Infrastructure and Operations at Viadeo since 2011

Email: aguy@viadeoteam.com
Twitter: @anto1ne

Viadeo.com

Leading professional social network
Founded in 2005, 10M members in France

What this case study will cover

- The rationale that led to an all-in move to Amazon Web Services
- The design options and trade-offs
- Why we did or didn't alter parts of our stack
- The complete automation of our infrastructure using AWS CloudFormation and continuous integration

Technology stack

- Service-oriented platform: « Kasper » (Java)
  - CQRS: Command-Query Responsibility Segregation
  - ES: Event Sourcing
  - DDD: Domain Driven Design

- Web application: « Limbo » (node.js)
- Mobile applications: iOS and Android
- Backends: MySQL, Elasticsearch, Hbase, Spark, Hadoop
- Infrastructure: 15 racks hosted in San Francisco, 250 servers, limited use of virtualization

Infrastructure challenges

1. Aging, unsupported hardware
2. Discrepancy between software agility and infrastructure rigidity
3. Operating a physical network
4. Operating a remote data center
5. Limited disaster recovery capabilities (RTO > 1 day)

4 options

	Server upgrade	Full virtualization	New datacenter	Public cloud
Refresh hardware	Solved	Solved	Solved	Solved
Improve agility	No change	Better	No change	Better
Manage network	No change	Worse	No change	Better
Manage remote DC	No change	No change	No change	Gone
Improve DR	No change	Little better	Better	Solved
Effort needed	Low	Medium	High	Damn!
Learning curve	None	Low	Low	Ouch!
Build spend	Very high	High	Very high	None
Run spend	Little better	Better	Better	Pay as you go
Gut feeling	Meh…	Too late	Wasteful	Party on!

Why a public cloud?

- We want to focus on our real job : **building a great service**
  - No more issues with hardware vendors, no more licensing hell. Ever!
  - Experimenting, deploying & scaling are just a few clicks away

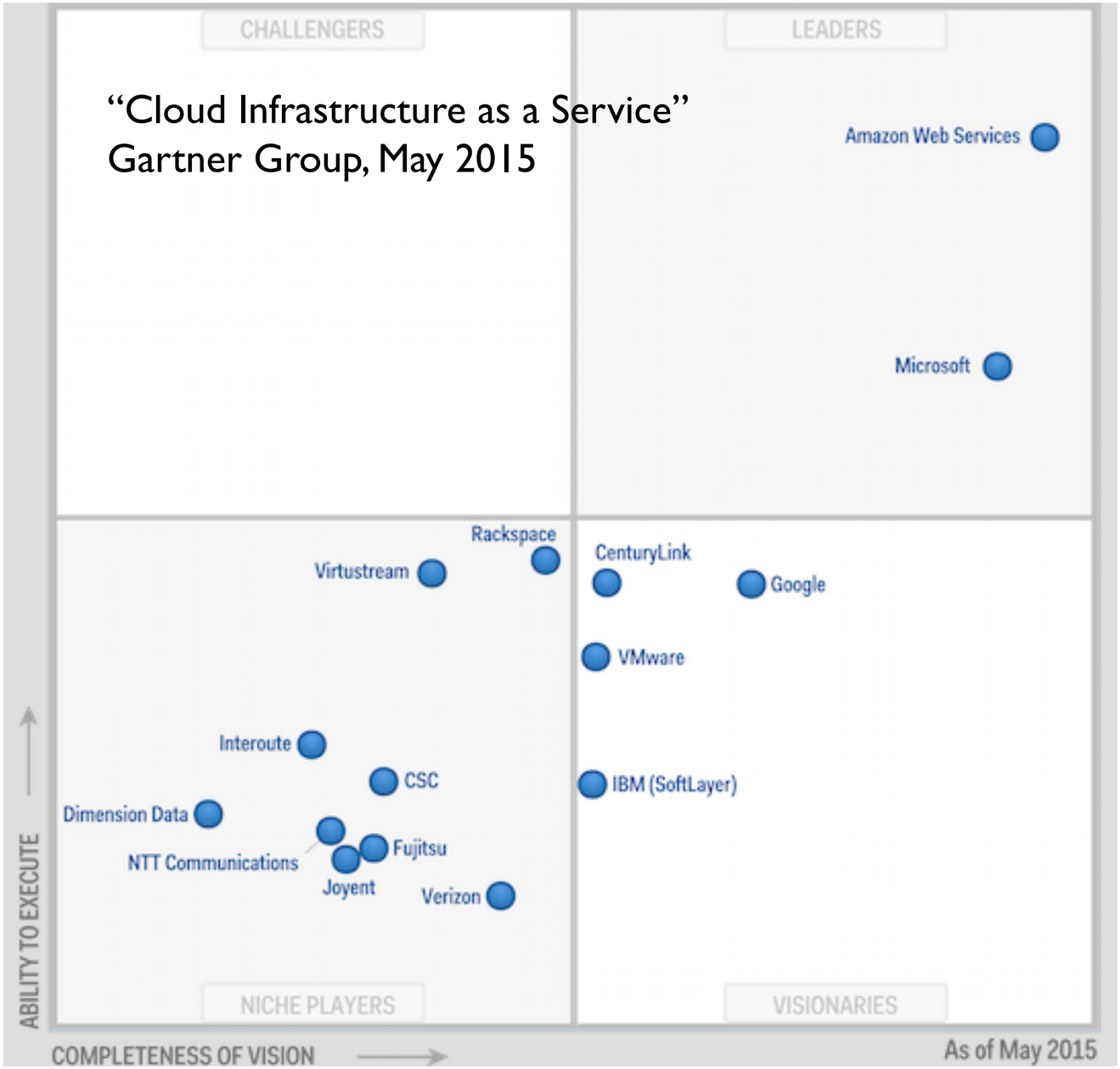
- We're **ready and eager for it** : Agile and DevOps are Viadeo values

- We want to **measure ROI** and **optimize cost**

- We've been using it for a while now **and it just works**

Why AWS

- Already using « cold » services
- Powerful PaaS technology accessible in a few clicks
- Multi region
- Lots of new features, constantly improving



Play book

- Key objectives: **automation**, **scalability**, **safety**
- Continuous integration & delivery **everywhere** (infra, config, apps)
- Replace parts of our stack only if the **benefits** are too good to pass
- All-in, but **no fork lift**: move gradually (and be ready to rollback)
- Plan & build with **temporary hybrid** run in mind

Pre-flight check

- Build a **thorough** report of your infrastructure.
  - Estimate equivalent cost in cloud. Don’t be afraid by initial amount.
  - Evaluate each part for replacement: leave as is, PaaS, SaaS ?
  - Find pain points: technical debt, relevance of moving legacy apps.

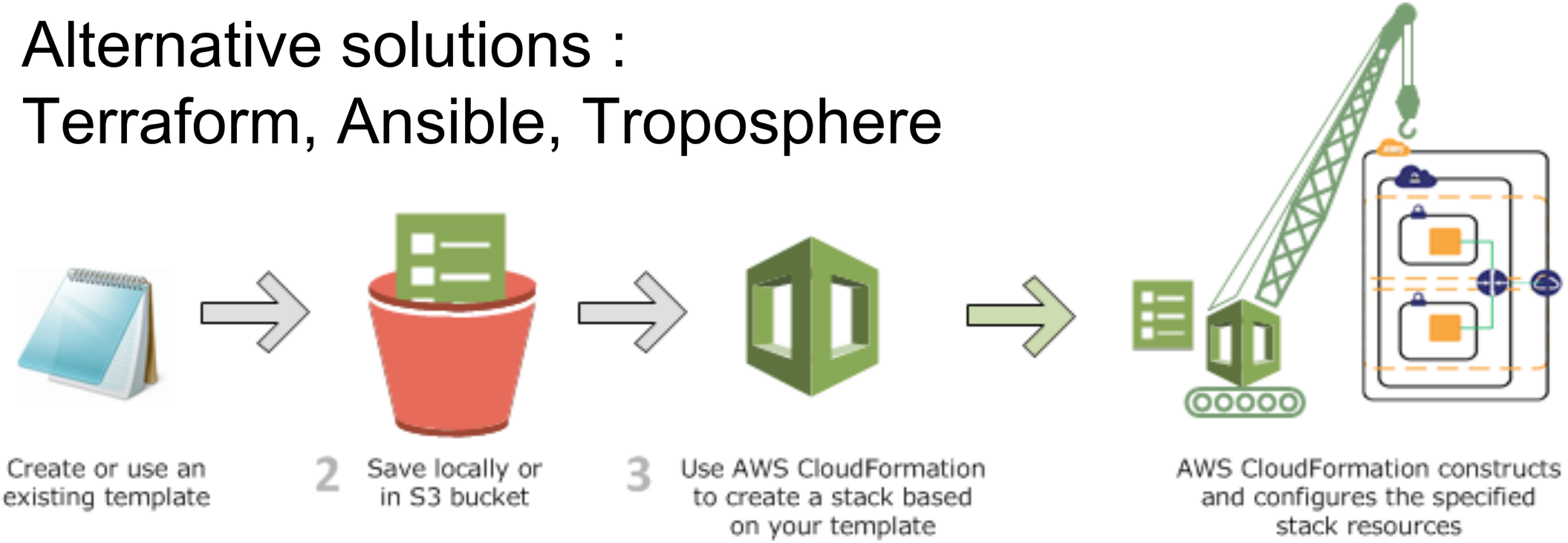
- Define a **high-level** migration plan (that you won’t follow)
  - Usually : staging, analytics, new projects, critical/legacy, close DC.
  - Expect (and allow) early adopters or emergencies to go first.

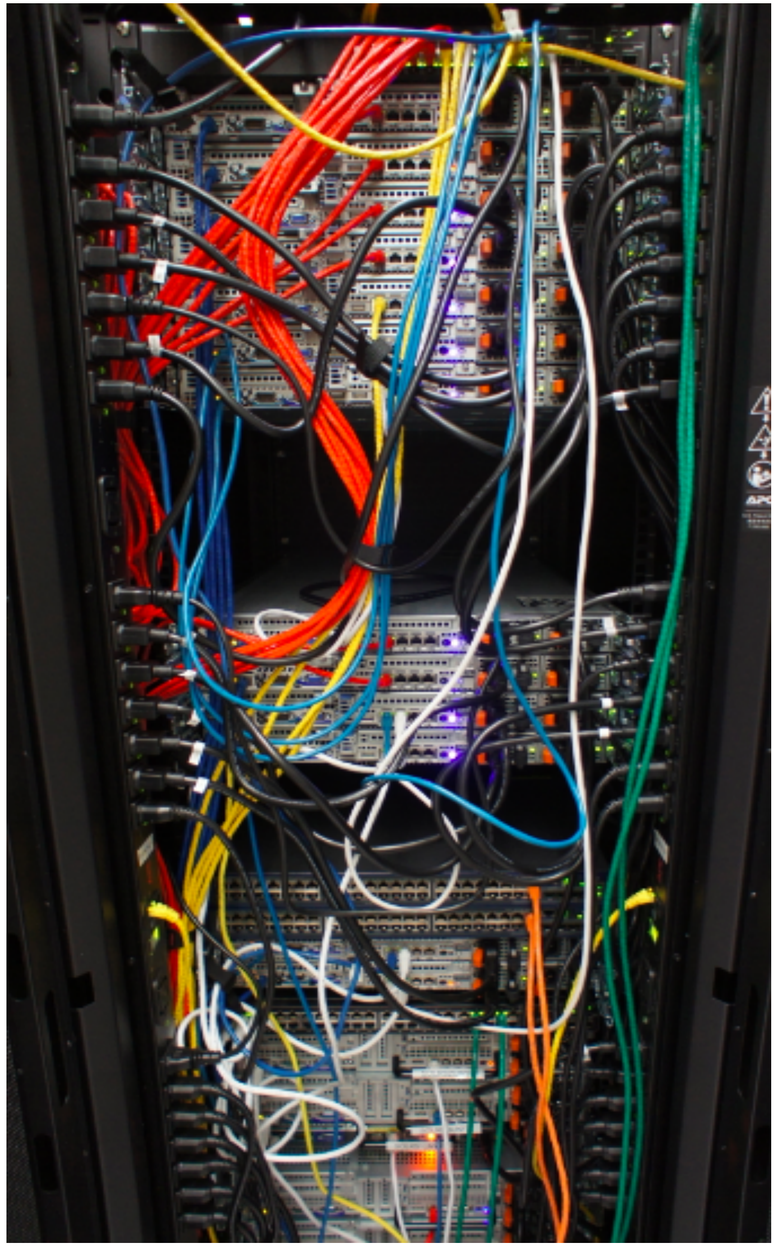
Infrastructure as code, for real

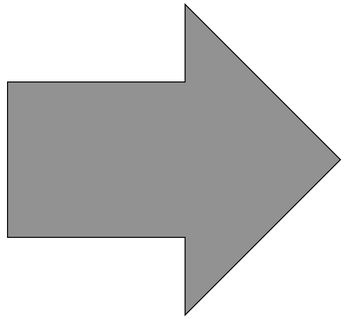
- **Versioned**, **auditable** blueprints (developers can contribute)
- **Quick to deploy**, **repeatable**, **tested** infrastructure
- Enables CI/CD for infrastructure (just like everything else)
- Deploy anywhere (for multi-region cloud)
- Cloud can now be its own DR

AWS CloudFormation

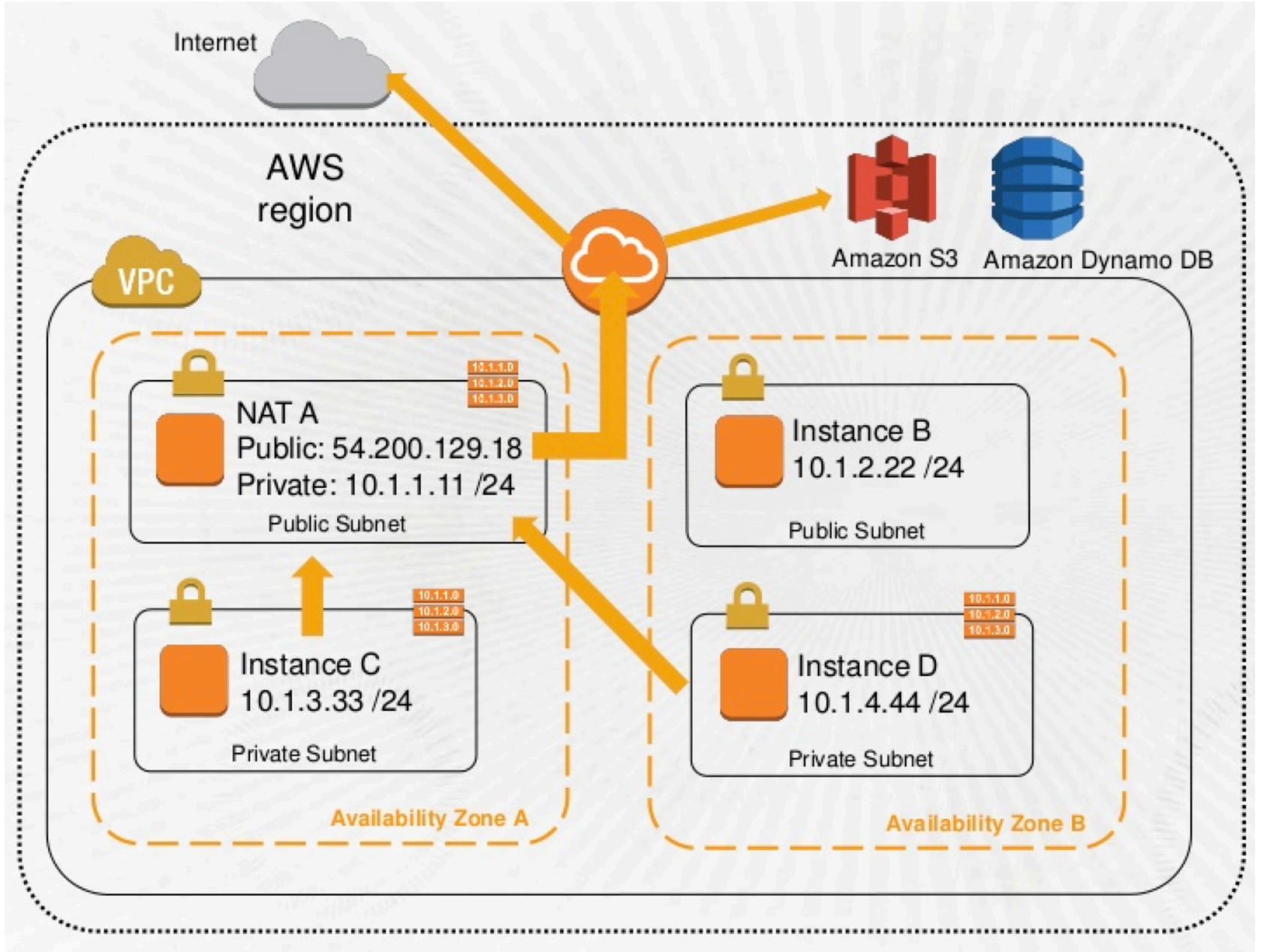
- **Json-based** descriptive “language” to create a set of AWS resources called a **stack**.
- Alternative solutions :
  Terraform, Ansible, Troposphere







```
"Conditions" : {
  "HaveNoOtherRoles"  : { "Fn::Equals" : [{"Ref" : "OtherRoles"}, ""]},
  "HaveEbs"           : { "Fn::Not" : [{ "Fn::Equals" : [{"Ref" : "EbsVolumeSize"}, "0"]}] },
  "HaveEbsSnapshotId" : { "Fn::Not" : [{ "Fn::Equals" : [{"Ref" : "EbsSnapshotId"}, ""]}] },
  "HaveAdditionalTagKey": { "Fn::Not" : [{ "Fn::Equals" : [{"Ref" : "AdditionalTagKey"}, ""]}] },
  "HaveAdditionalTagValue": { "Fn::Not" : [{ "Fn::Equals" : [{"Ref" : "AdditionalTagValue"}, ""]}] },
  "HaveSSL": { "Fn::Not" : [{ "Fn::Equals" : [{"Ref" : "SSLPort"}, "0"]}] },
  "IsHTTP" : { "Fn::Equals" : [{"Ref" : "ElbProtocol"}, "HTTP"]},
  "HaveSpotPrice" : { "Fn::Not" : [{ "Fn::Equals" : [{"Ref" : "SpotPrice"}, ""]}]}
},
"Resources": {
  "AutoScalingGroup": {
    "Type": "AWS::AutoScaling::AutoScalingGroup",
    "UpdatePolicy" : {
      "AutoScalingRollingUpdate" : {
      "MaxBatchSize" : "1",
      "MinInstancesInService" : "0",
      "PauseTime" : "PT15M",
      "WaitOnResourceSignals": "true"
      }
    },
    "Properties": {
      "LaunchConfigurationName": { "Ref": "LaunchConfig" },
      "LoadBalancerNames": [ { "Ref": "ElasticLoadBalancer" } ],
      "MinSize": { "Ref": "MinPoolSize" },
      "MaxSize": { "Ref": "MaxPoolSize" },
      "AvailabilityZones": { "Fn::FindInMap": ["AZConfig", "AvailabilityZones", "all"] },
      "VPCZoneIdentifier": { "Ref": "EC2SubnetsIds" },
      "Tags" : [
        { "Fn::If": [
            "HaveAdditionalTagKey",
            {
              "Key" : { "Ref": "AdditionalTagKey" },
              "Value": {
                "Fn::If": [
                  "HaveAdditionalTagValue",
                  {"Ref": "AdditionalTagValue"},
                  ""
                ]
              },
              "PropagateAtLaunch": "true"
            },
            {"Ref" : "AWS::NoValue"}
          ]
        },
        { "Key" : "Name", "Value" : { "Fn::Join" : [ "-", [ { "Ref" : "ServiceName"}, {"Ref" : "EnvironmentName"
        { "Key" : "cost", "Value" : { "Ref" : "Cost" }, "PropagateAtLaunch": "true" },
        { "Key" : "environment", "Value": { "Ref" : "EnvironmentName"}, "PropagateAtLaunch": "true" }
      ]
    }
```


Internet
AWS
region
VPC
Amazon S3
Amazon Dynamo DB
NAT A
Public: 54.200.129.18
Private: 10.1.1.11 /24
Public Subnet
Instance B
10.1.2.22 /24
Public Subnet
Instance C
10.1.3.33 /24
Private Subnet
Instance D
10.1.4.44 /24
Private Subnet
Availability Zone A
Availability Zone B

Step 1: automate your infra build

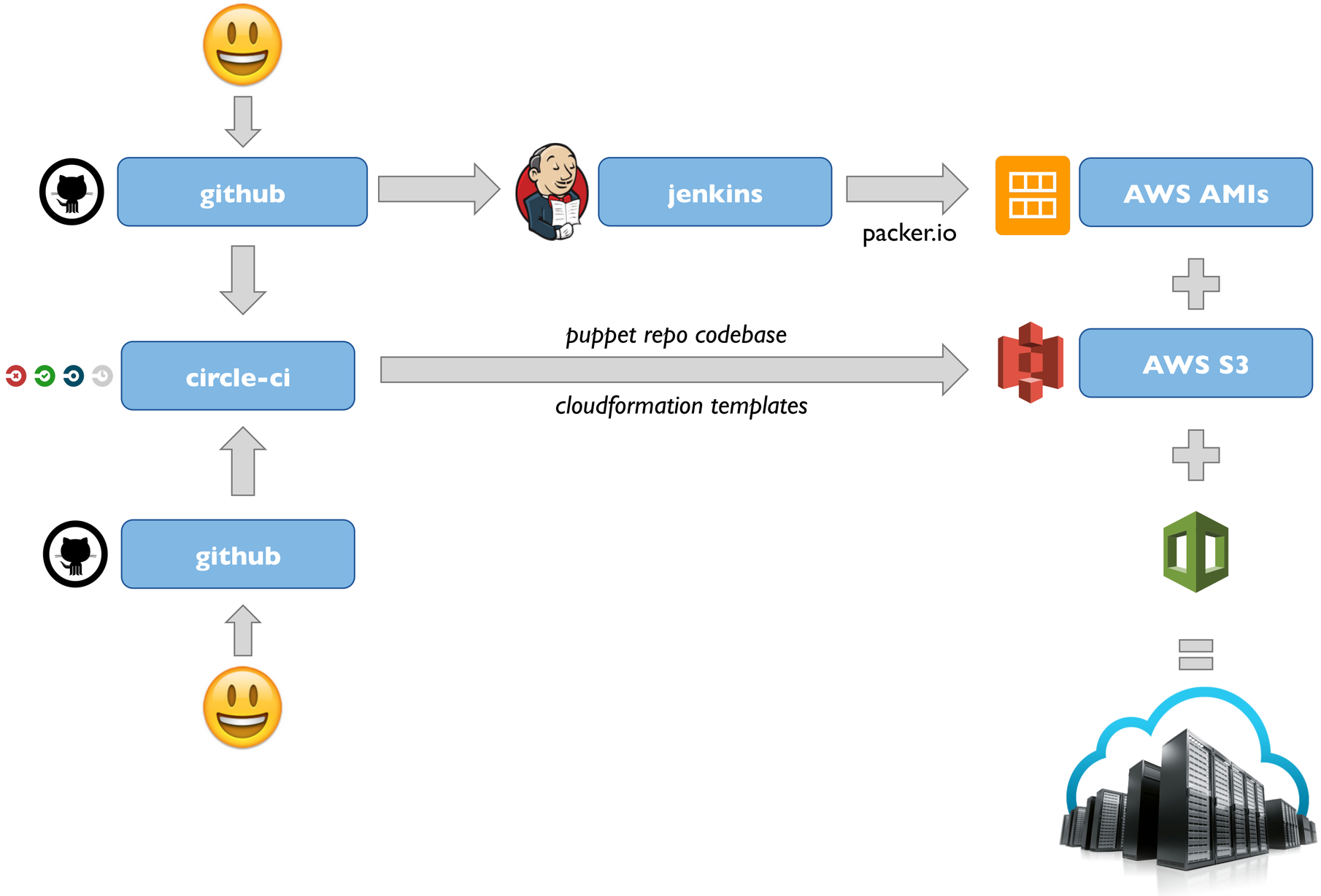
- `$ aws cloudformation validate-template`
- Write your test cases in the language of your choice (many SDKs available)
- Use `git` with a trigger to a CI tool (`circle-ci` or other)
- If tests pass : deploy (in your staging environment first)

Step 2: build and CI your images

- **Automate** your baking:
  1 for each role from your config management

- **Bake** automatically with `packer` or `aminator`

- **Baked** or **bootstrapped** ?
  - Fully baked for autoscale services
  - Bootstrapped for non critical, fast-changing services


github
jenkins
packer.io
AWS AMIs
circle-ci
puppet repo codebase
cloudformation templates
AWS S3
github

CloudFormation best practices

- Use AWS CloudFormer as a **starting point**
- **Reuse** as much as possible
  - Nested stack, but not too much, max 1 level (hello UPDATE_ROLLBACK_FAILED).
  - Parameterized template : env, region, name, size, etc..
- Use stacks to perform **green/blue** deployments
- **Tag** everything!

AWS: lessons learned (1)

- Plan your VPC network **early**. Watch ARC403
- Hybrid run requires good **connectivity**
  → DirectConnect helps, but watch NET406

- ELB **is level 3** only, with limited options

- CloudFront performance not great across continents
- A 2nd CDN may be required, if only for peace of mind

AWS: lessons learned (2)

- EMR != 24/7 Hadoop cluster
- EMR in VPC with private DNS is very difficult

- MySQL: RDS doesn't allow **myISAM**

- Limits: **check** them, raise them. Same everywhere
- No more static naming/addressing: use **service discovery**
- Make sure everything is **multi-AZ** and **tagged**

Tech is only half the work

- Identify high-level **stakeholders** and understand THEIR **goals**
- Involve your Legal/Finance departments **early**
  - Budgeting: cost overlap, AWS support, etc.
  - Early termination of legacy infrastructure contracts (there will be blood)
- Build an **A-team** (dev, ops, security)
- Work on **awareness** and **knowledge transfer**

Current status

- **Staging** environment: done and running
- **Production** web traffic served from AWS

- Infrastructure live in 3 regions (us-east-1, us-west-2, eu-west-1)
- Multiple inter-connected VPC

- ≈ 100 AWS EC2 instances (half production, half development)
- 2 AWS Redshift clusters (5 instances each)

Next steps

- Start moving backend servers to AWS
- Live Hadoop cluster → EMR
- Clean-up MySQL stack before we can use RDS
- Elasticsearch will remain EC2-based for now
- Optimize: reserved instances, spot instances, auto-scale

Conclusion

- 5 years ago : «Cloud computing? Why?»
- Now : «NO cloud computing? Why?»
- Some good reasons, but many outdated ones
- Cloud computing a fashion? We don’t think so
- Infrastructure is now digital, like photos, music, movies, money, etc.
- We’re making the most of it, it’s a great engineering challenge!

O'REILLY®

Velocity

CONFERENCE

BUILD RESILIENT SYSTEMS AT SCALE

aguy@viadeoteam.com

julsimon@amazon.com

velocityconf.com/eu

#velocityconf