https://bit.ly/2zJJ3Fh

Sponsors

datree.io

aws

Production Engineering on AWS

Wifi: Guest

Pass: Spheres@@2018

AWS Israel Community

- Founded - Feb **2013**
- **81** meetups with **~6000** Members
- Monthly meetups
- No Marketing, No bullshit
- All AWS: AI, BigData, Serverless, Containers, etc

MEET THE TEAM

Shimon Tolts

Arthur Schmunk

Tal Hibner

Niv Yungelson

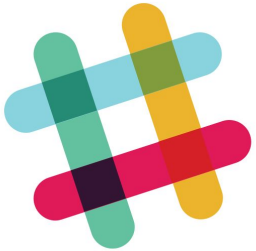
Eitan Sela

Andrei Burd

Doron Rogov

Boaz Ziniman

Join the Community!



https://bit.ly/2zJJ3Fh

https://www.meetup.com/AWS-IL/

https://www.meetup.com/AWS-IL/

aws.org.il

AWS User Group Israel

HOME ABOUT COMING MEETUPS SPEAKERS THE LEADERSHIP TEAM

Coming meetups

Coming meetups

- Big Data on AWS - 2018-07-16 18:00 @ AWS Offices.

Past meetups

2018

- Guest Meetup: AWS Cloud Financial Governance Practice

Upcoming Meetup

Bad Practices On AWS - 22/10

Production Engineering on AWS

- Cloud-Native distributed tracing using open source - Jaeger and opentracing

  by Itiel Shwartz from Rookout

- Blue/green deployment, canary releases and database consistency by

  Tomer and Evgeny from Quali

Distributed tracing

- Jaeger 101 -

About me

- Worked at **eBay**
- Worked at **Forter** as a backend engineer.
- Joined **Rookout** as a first developer and production engineer
- @itielshwartz on both Github and Twitter
- Also have a personal blog at: https://etlsh.com

Agenda

Intro:

1. State of mind for this Meetup (Super important!)
2. What is Distributed tracing, do i need it?
3. What is open tracing?
4. What is jaeger?

Zero to hero using Jaeger:

1. hello-world example
2. Jaeger terminology
3. Full blown distributed app

Wrap up

1. Demo wrap up
2. Jaeger architecture
3. Opentracing Secret ability

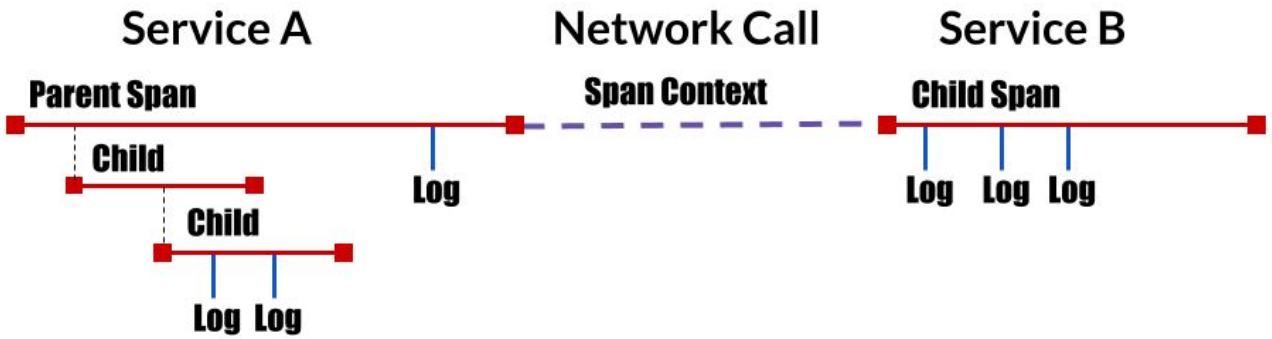
Before we begin (State of mind)

- The system will fail
- Your code is not perfect
- Other people code is even less perfect
- Practice new tools at daytime, don’t start using them in crisis mode
- The system will fail
- Each minute you spend adding logs and metrics can reducde your Mean Time to Resolve (MTTR)
- Keep in mind the developer that’s going to get a pager isn’t the one that wrote the code
- Try to be nice to him - he is going to need it
- The system will fail

As you can probably see i (tried) to emphasize the fact that your system is going to fail, this **DOESN'T** mean i think you write bad code - only that we usually have much more trust in our code/infra then we should :)

What is distributed tracing?

With distributed tracing, we can track requests as they pass through multiple services, emitting timing and other metadata throughout, and this information can then be reassembled to provide a complete picture of the application's behavior at runtime - buoyant



Mental model of distributed tracing - Opentracing

Do i need distributed tracing?

As companies move from monolithic to multi-service architectures, existing techniques for debugging and profiling begin to break down.

Previously, troubleshooting could be accomplished by isolating a single instance of the monolith and reproducing the problem.

With microservices, this approach is no longer feasible, because no single service provides a complete picture of the performance or correctness of the application as a whole.

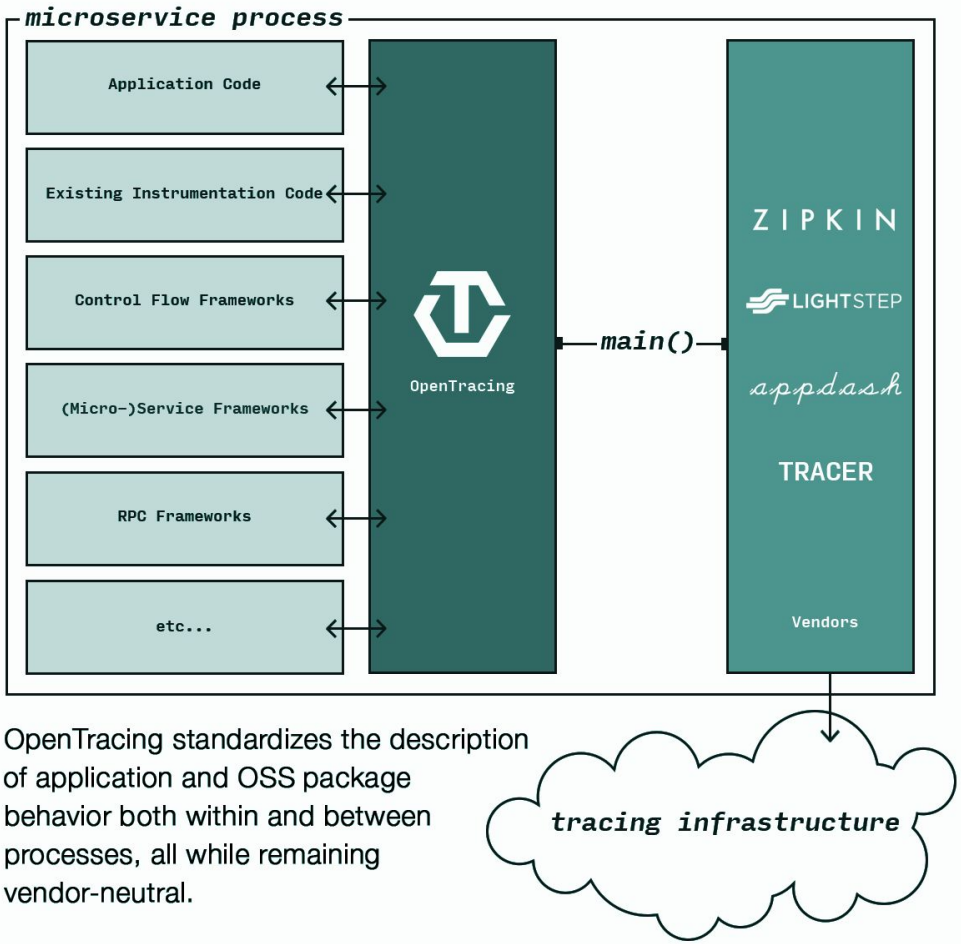
We need new tools to help us manage the real complexity of operating distributed systems at scale. - buoyant

What is opentracing?

The problem is that distributed tracing has long harbored a dirty secret: the necessary source code instrumentation has been complex, fragile, and difficult to maintain.

This is the problem that OpenTracing solves.
Through standard, consistent APIs in many languages (Java, Javascript, Go, Python, C#, others), the OpenTracing project gives developers clean, declarative, testable, and vendor-neutral instrumentation.

OpenTracing has focused on standards for explicit software instrumentation.



OpenTracing standardizes the description of application and OSS package behavior both within and between processes, all while remaining vendor-neutral.

What is Jaeger?

Jaeger, inspired by Dapper and OpenZipkin, is a distributed tracing system released as open source by Uber Technologies.

It can be used for monitoring microservices-based distributed systems:

- Distributed context propagation
- Distributed transaction monitoring
- Root cause analysis
- Service dependency analysis
- Performance / latency optimization

Getting started - The Monolith

https://github.com/itielshwartz/jaeger-hello-world/tree/step-1-the-monolith

Getting started - Monolith going wild

https://github.com/itielshwartz/jaeger-hello-world/tree/step-2-the-monolith-going-wild

Jaeger terminology - Span/ Trace

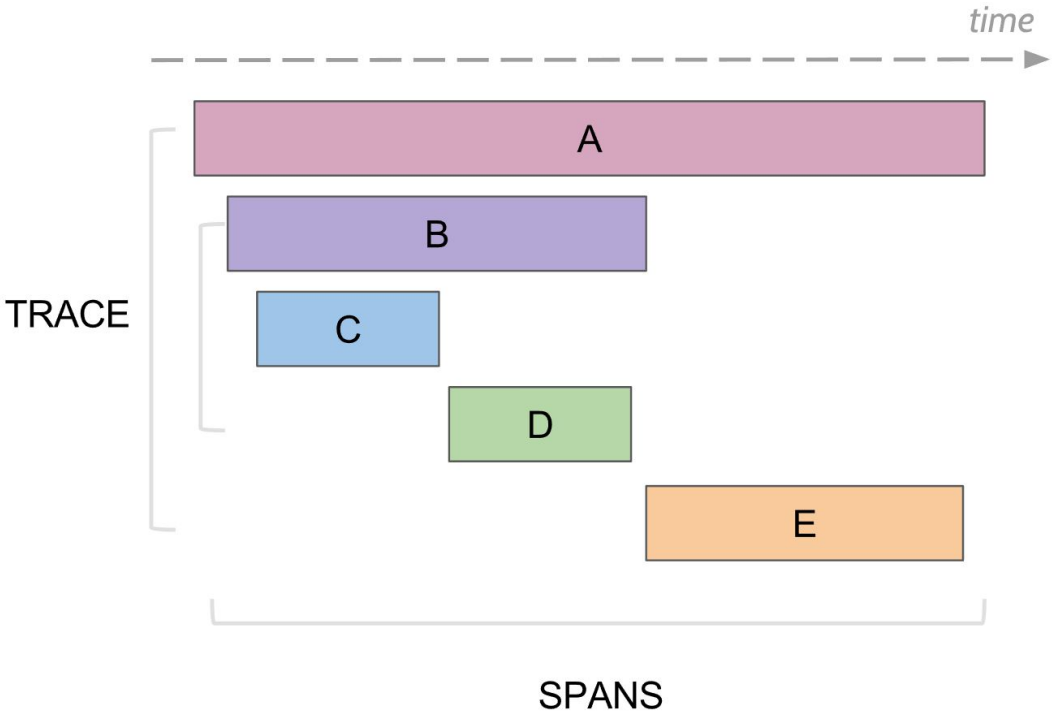
Span

A **span** represents a logical unit of work in Jaeger that has an operation name, the start time of the operation, and the duration. Spans may be nested and ordered to model causal relationships.

Trace

A **trace** is a data/execution path through the system, and can be thought of as a directed acyclic graph of spans.

Jaeger terminology - Span/ Trace



Getting started - Adding Jaeger

https://github.com/itielshwartz/jaeger-hello-world/tree/step-3-adding-jaeger

Config Jaeger part II - Multiple spans

https://github.com/itielshwartz/jaeger-hello-world/tree/step-4-multiple-spans

Jaeger architecture - Tag/Log

The recommended solution is to annotate spans with tags or logs.

Tag:

A *tag* is a key-value pair that provides certain metadata about the span.

Log:

A *log* is similar to a regular log statement, it contains a timestamp and some data, but it is associated with span from which it was logged.

When and why?

When should we use tags vs. logs? The tags are meant to describe attributes of the span that apply to the whole duration of the span. For example, if a span represents an HTTP request, then the URL of the request should be recorded as a tag because it does not make sense to think of the URL as something that's only relevant at different points in time on the span. On the other hand, if the server responded with a redirect URL, logging it would make more sense since there is a clear timestamp associated with such event. The OpenTracing Specification provides guidelines called Semantic Conventions for recommended tags and log fields.

https://github.com/yurishkuro/opentracing-tutorial/tree/master/python/lesson01#annotate-the-trace-with-tags-and-logs

Config Jaeger part III - Tags and Log

https://github.com/itielshwartz/jaeger-hello-world/tree/step-5-tags-and-logs

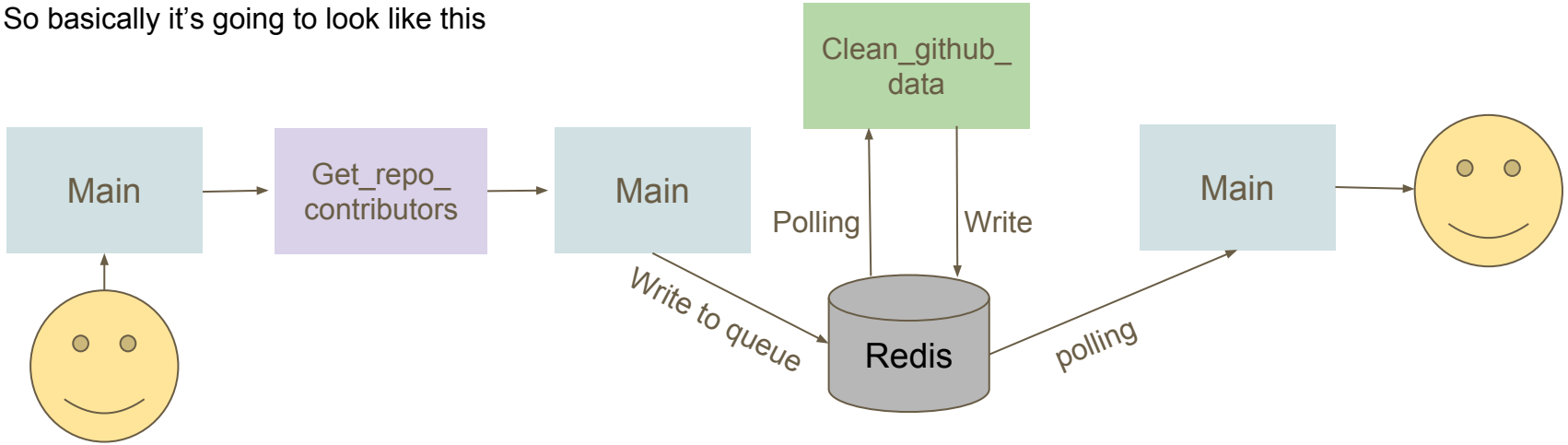
Going distributed

Until now we had single server (what kind of defy the purpose of distributed tracing).
Now let's split our monolith into small parts - we will still have a main server (customer facing) but not we will split get_repo_contributors And clean_github_data Into two different service.

Get_repo_contributors - Will be a flask server (same as our main)

Clean_github_data - Will Consume data from redis (pushed to it by the master)

So basically it's going to look like this



Going distributed - Single span

https://github.com/itielshwartz/jaeger-hello-world/tree/step-6-distribute-single-span

Going distributed - Multiple span

https://github.com/itielshwartz/jaeger-hello-world/tree/step-7-distribute-multiple-spans

Demo wrap up

We now have successfully transformed a monolith beast into a set of small microservices - without losing visibility.

The nice thing about opentracing is that it allow us to move from jaeger to datadog to other solution without (almost) needing to rewrite our code.

The other cool thing about it is that you **don’t need to do everything i just did in this demo**!

There are official wrappers for most of the common framework those tools allow you you to integrate with opentracing and jager without needing to think about “how do i pass the headers inside the request?” or “ how do i read the headers to start a new span?”

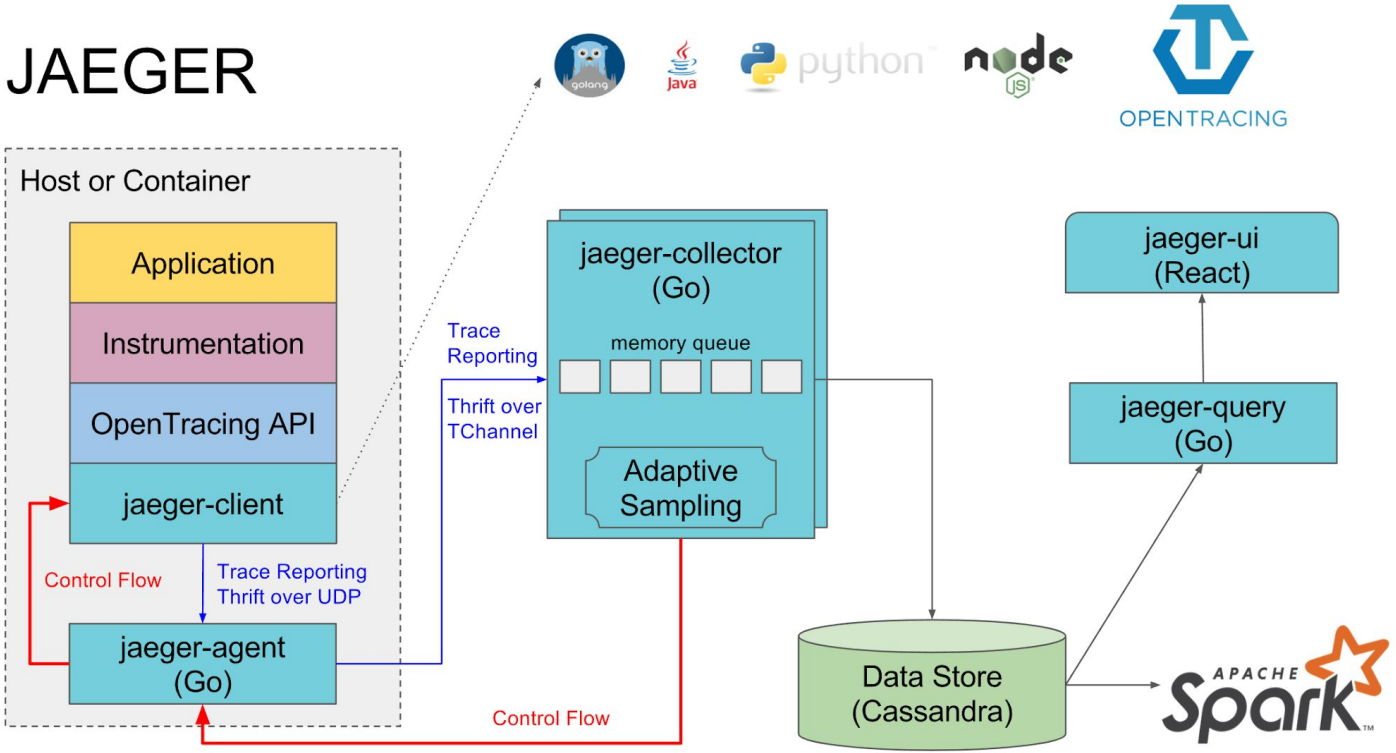
Examples”

- urllib2
- requests
- SQLAlchemy

- MySQLdb
- Tornado HTTP client
- redis

- Flask
- Django
- More

Jaeger Architecture



Jaeger Architecture

Agent

The Jaeger **agent** is a network daemon that listens for spans sent over UDP, which it batches and sends to the collector. It is designed to be deployed to all hosts as an infrastructure component. The agent abstracts the routing and discovery of the collectors away from the client.

Collector

The Jaeger **collector** receives traces from Jaeger agents and runs them through a processing pipeline. Currently our pipeline validates traces, indexes them, performs any transformations, and finally stores them.

Jaeger's storage is a pluggable component which currently supports Cassandra and ElasticSearch.

Query

Query is a service that retrieves traces from storage and hosts a UI to display them.

Opentracing Secret ability

Context propagation

With OpenTracing instrumentation in place, we can support general purpose *distributed context propagation* where we associate some metadata with the transaction and make that metadata available anywhere in the distributed call graph. In OpenTracing this metadata is called *baggage*, to highlight the fact that it is carried over in-band with all RPC requests, just like baggage. opentracing-tutorial

The client may use the Baggage to pass additional data to the server and any other downstream server it might call.

```
# client side
span.context.set_baggage_item('auth-token', '.....')

# server side (one or more levels down from the client)
token = span.context.get_baggage_item('auth-token')
```

Questions?

Quali

Our approach to B/G deployment

Tomer Admon
Evgeny Khaliper

Who we are?

Tomer Admon
Team Leader
Quali
tomer.a@quali.com

Evgeny Khaliper
Senior Developer
Quali
evgeny.k@quali.com

"Quali builds sandbox automation software that provides self-service, on-demand access to application and IT infrastructure environments across private, public, and hybrid-cloud."

Quali

Agenda

- Blue, Green and everything in between.
- B/G & Immutable environments?
- B/G @ Quali
- DB Backward/Forward compatibility
- Q&A

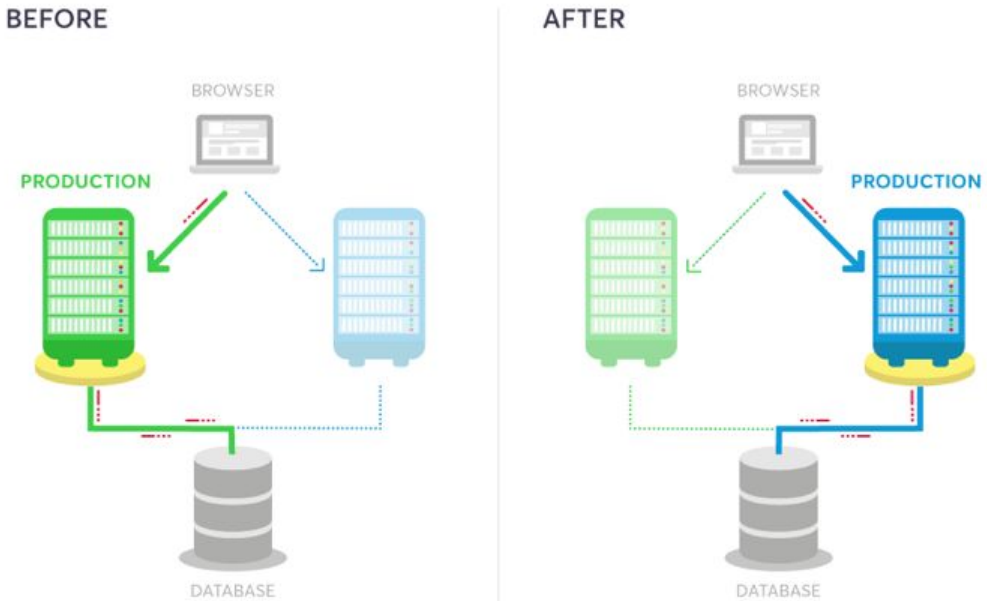
Continuous deployment & B/G

"Continuous Delivery is the ability to get changes of all types – including new features, configuration changes, bug fixes and **experiments** – into production or **into the hands of users**, safely and quickly in a sustainable way"

Quali

C/D & B/G

- One implementation of CD
- No downtime
- Reduced risk
- Increased confidence
- Test on ‘production to be’



Quali

B/G: Exposure controllers

- **On/Off = Red/Black**
- **Knob = Canary**
- **Not A/B testing**

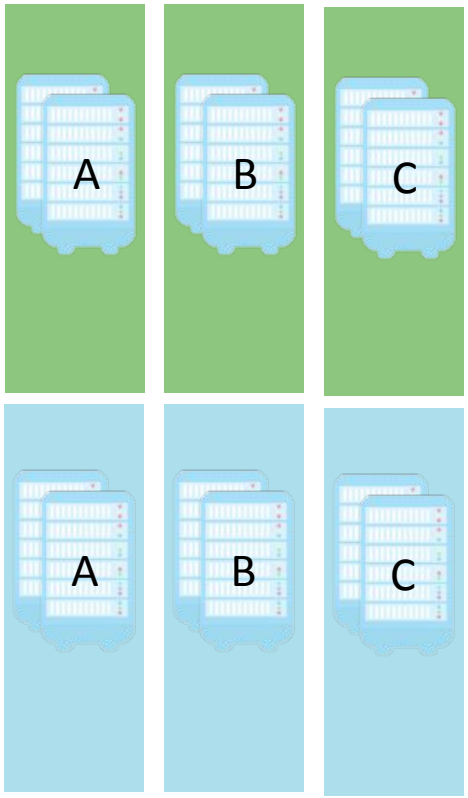
Quali

Immutable environments?

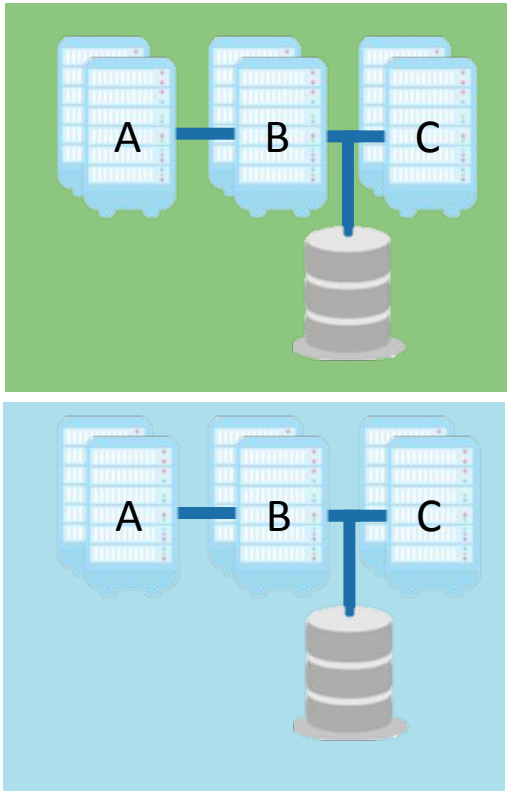
Quali

Immutable environments?

- Two main Flavors:
  - Service B/G
  - Environment B/G



VS



Quali

Immutable environments?

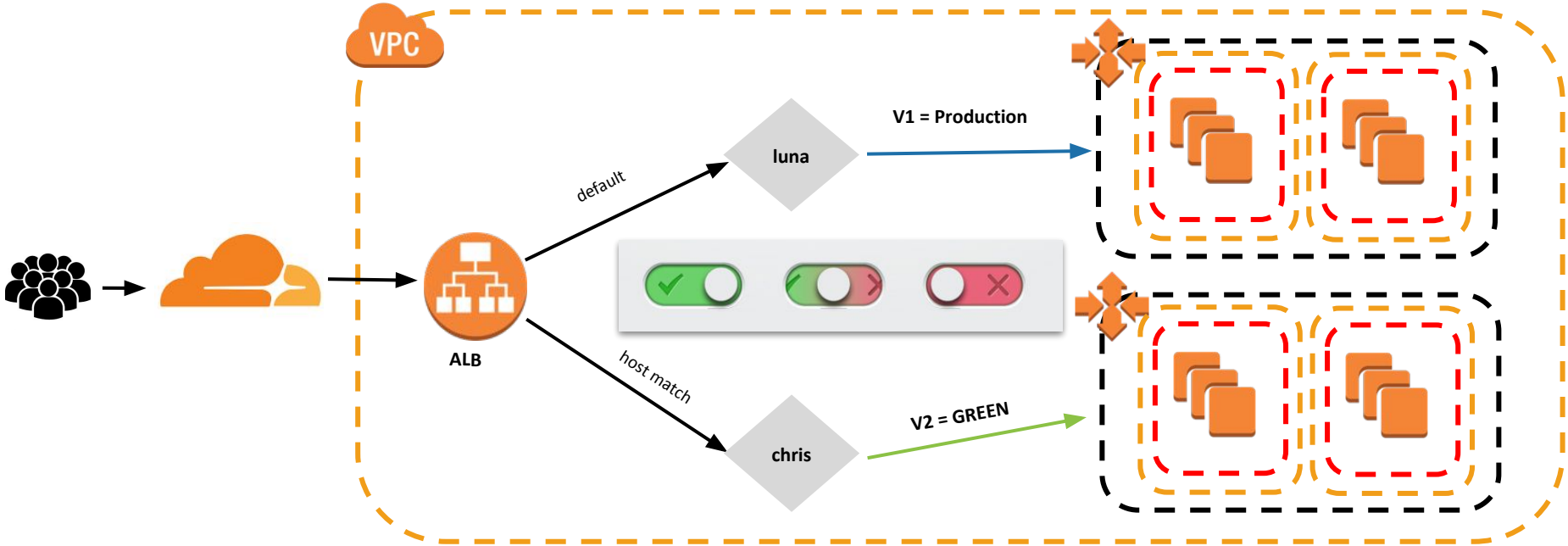
- **Pros**
  - **Easy to maintain**
  - **Enforce quick deployment time**
  - **Backward compatibility is easy**
  - **Repeatable – same procedure every time**
  - **Easy to test**

- **Cons**
  - **Not suitable for huge applications**
  - **Slow to deploy → slower feedback**
  - **Cost**

Quali

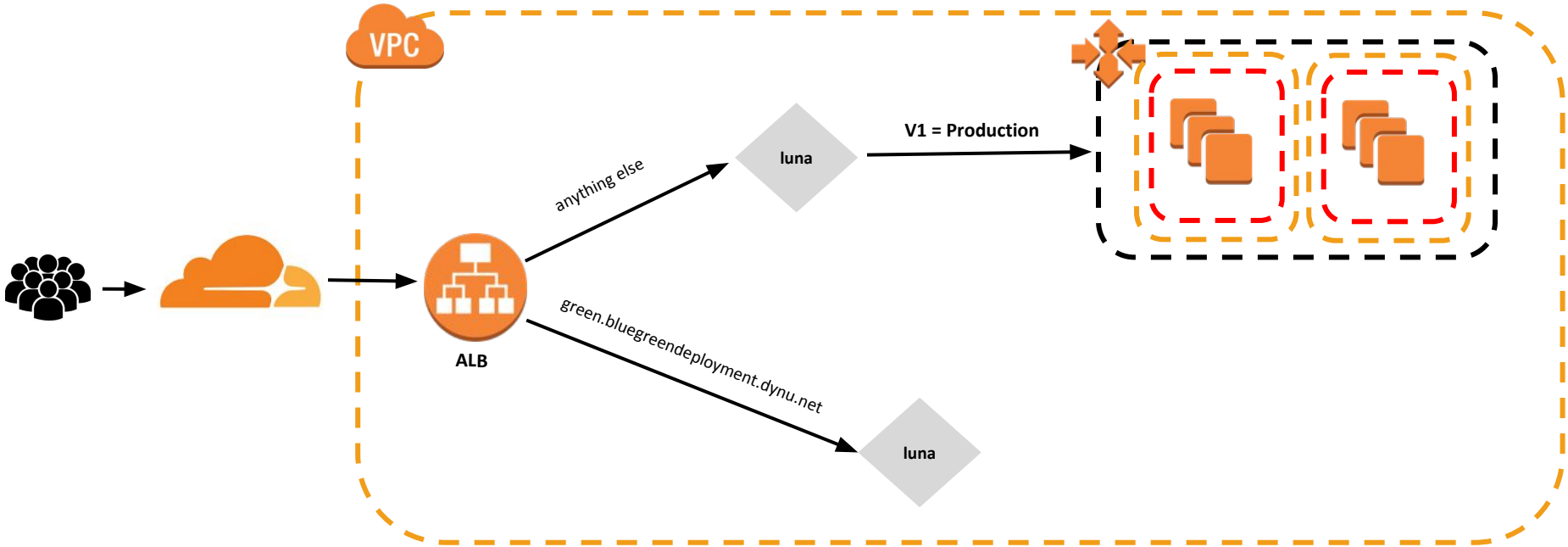
B/G @ Quali

Our implementation

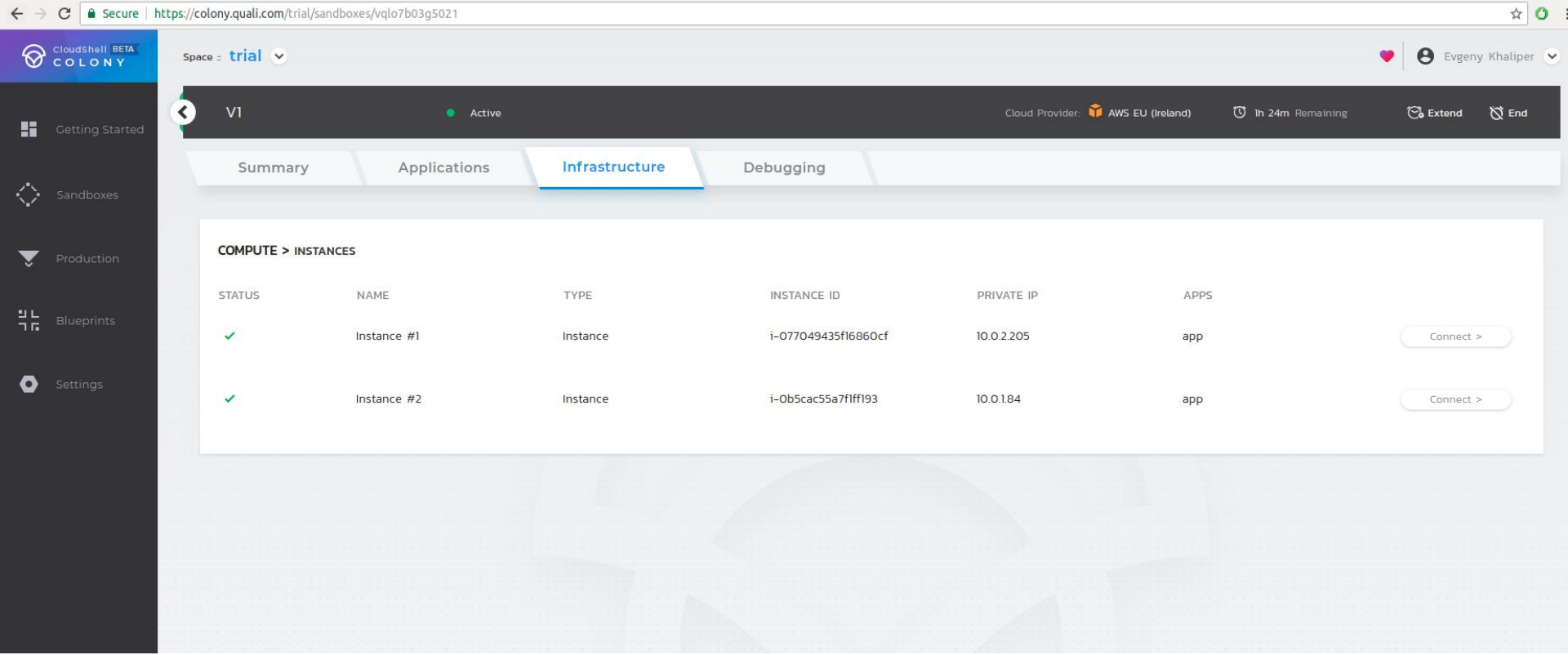


Quali

B/G @ Quali: Initial state

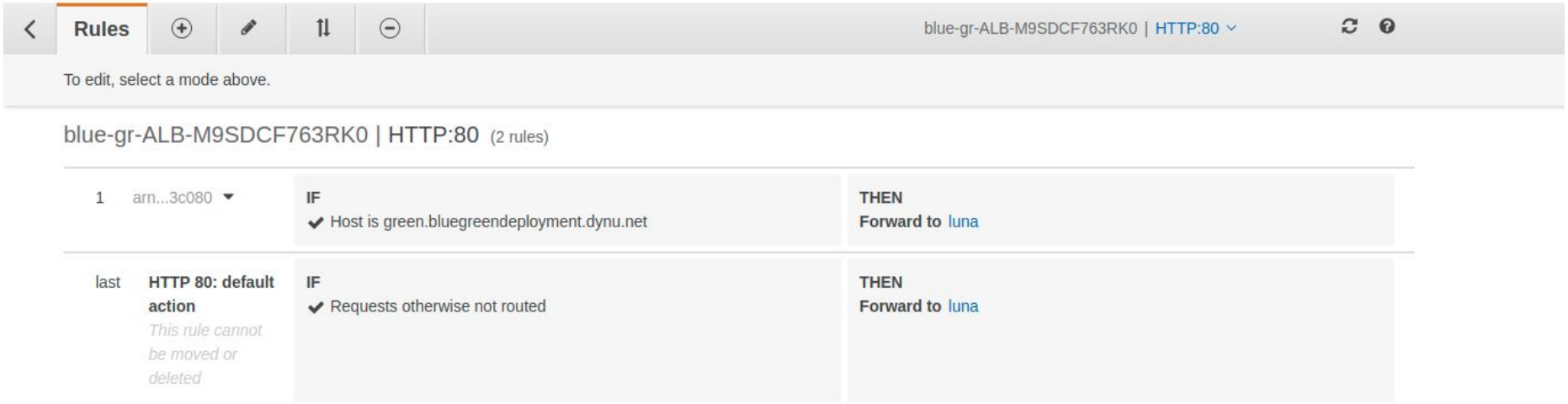


Quali


Secure | https://colony.quali.com/trial/sandboxes/vqlo7b03g5021
Cloudshell BETA COLONY
Space : trial
Evgeny Khaliper
Getting Started
Sandboxes
Production
Blueprints
Settings
V1
Active
Cloud Provider: AWS EU (Ireland)
1h 24m Remaining
Extend
End
Summary
Applications
Infrastructure
Debugging
COMPUTE > INSTANCES
STATUS
NAME
TYPE
INSTANCE ID
PRIVATE IP
APPS
Instance #1
Instance
i-077049435f16860cf
10.0.2.205
app
Connect >
Instance #2
Instance
i-0b5cac55a7f1ff193
10.0.1.84
app
Connect >

Quali

AWS Topology: ALB Rules before green



Quali

Demo V1

Quali

AWS Topology: Target Groups "luna"

Create target group | Actions

search : arn:aws:elasticloadbalancing:eu-west-1:... | Add filter | 1 to 1 of 1

Name	Port	Protocol	Target type	Load Balan	VPC ID	Monitoring
luna	80	HTTP	instance	blue-gr-AL...	vpc-45db8023	

Target group: luna

Description | Targets | Health checks | Monitoring | Tags

The load balancer starts routing requests to a newly registered target as soon as the registration process completes and the target passes the initial health checks. If demand on your targets increases, you can register additional targets. If demand on your targets decreases, you can deregister targets.

Edit

Registered targets

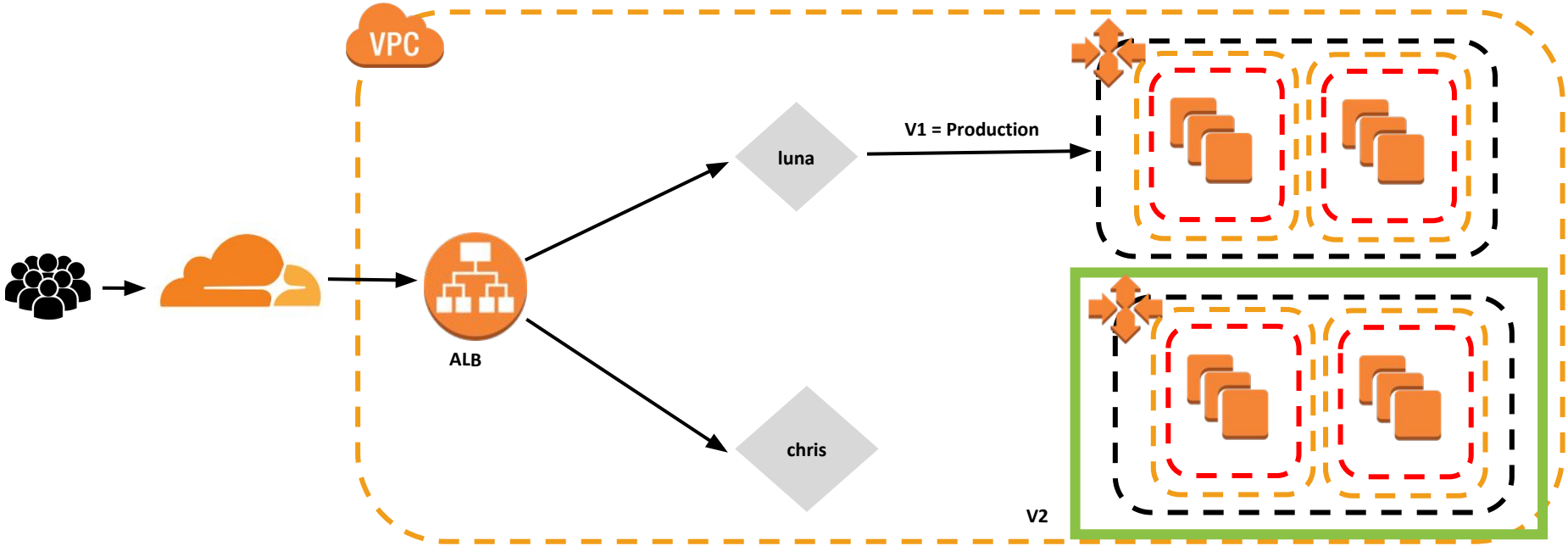
Instance ID	Name	Port	Availability Zone	Status
i-077049435f16860cf	app	80	eu-west-1b	healthy
i-0b5cac55a7f1ff193	app	80	eu-west-1a	healthy

Availability Zones

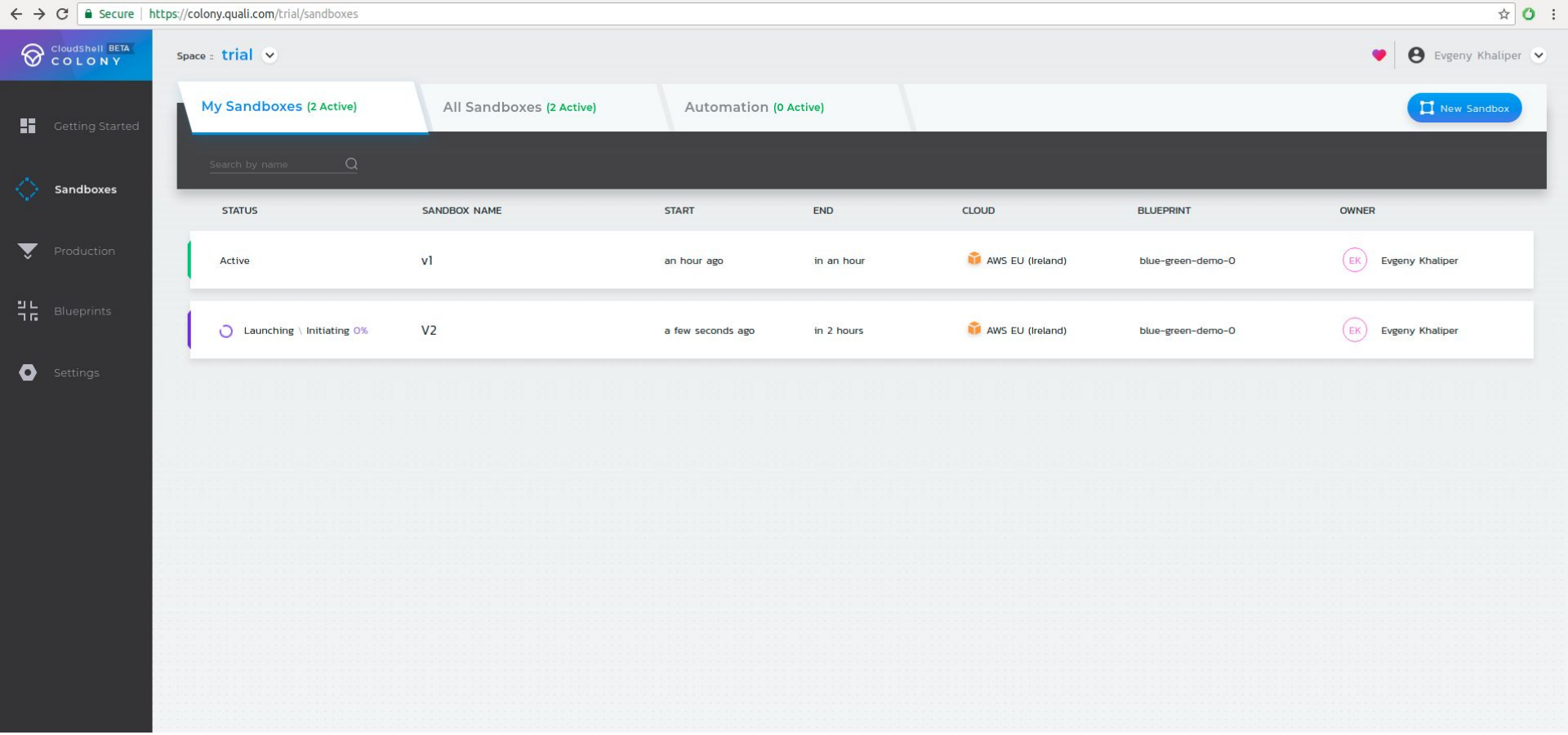
Availability Zone	Target count	Healthy?
eu-west-1b	1	Yes
eu-west-1a	1	Yes

Quali

AWS Topology: V2 deployment

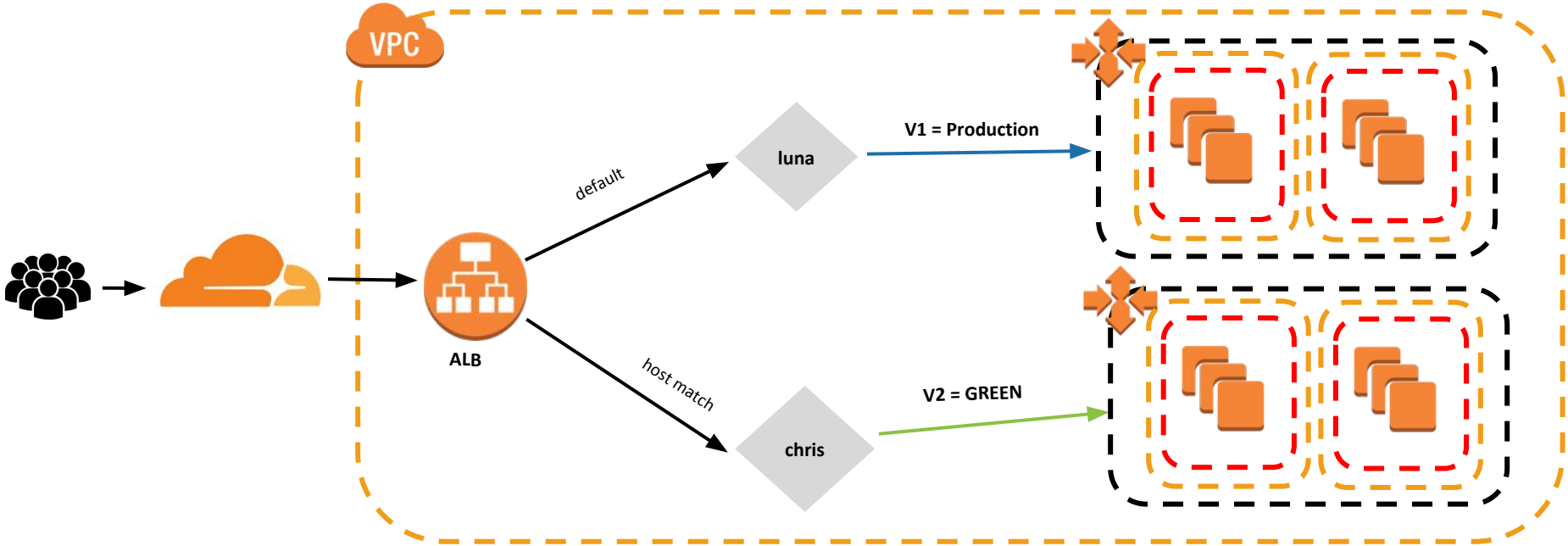


Quali


Secure | https://colony.quali.com/trial/sandboxes
CloudShell BETA COLONY
Space :: trial
Evgeny Khaliper
Getting Started
Sandboxes
Production
Blueprints
Settings
My Sandboxes (2 Active)
All Sandboxes (2 Active)
Automation (0 Active)
New Sandbox
Search by name
STATUS
SANDBOX NAME
START
END
CLOUD
BLUEPRINT
OWNER
Active
v1
an hour ago
in an hour
AWS EU (Ireland)
blue-green-demo-0
EK
Evgeny Khaliper
Launching \ Initiating 0%
V2
a few seconds ago
in 2 hours
AWS EU (Ireland)
blue-green-demo-0
EK
Evgeny Khaliper

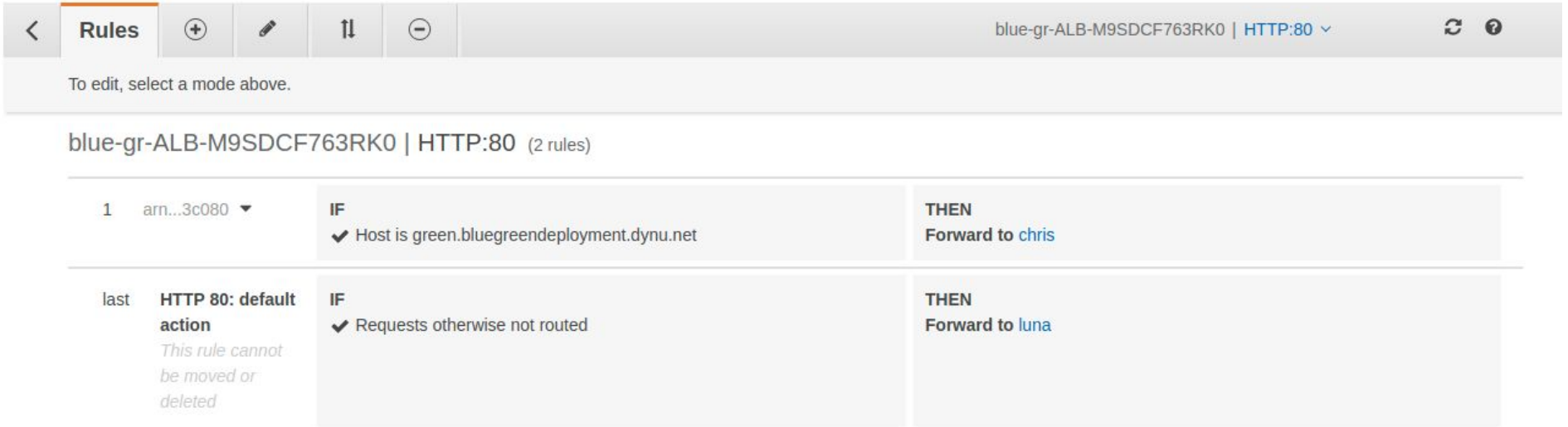
Quali

AWS Topology: Limited exposure of the GREEN



Quali

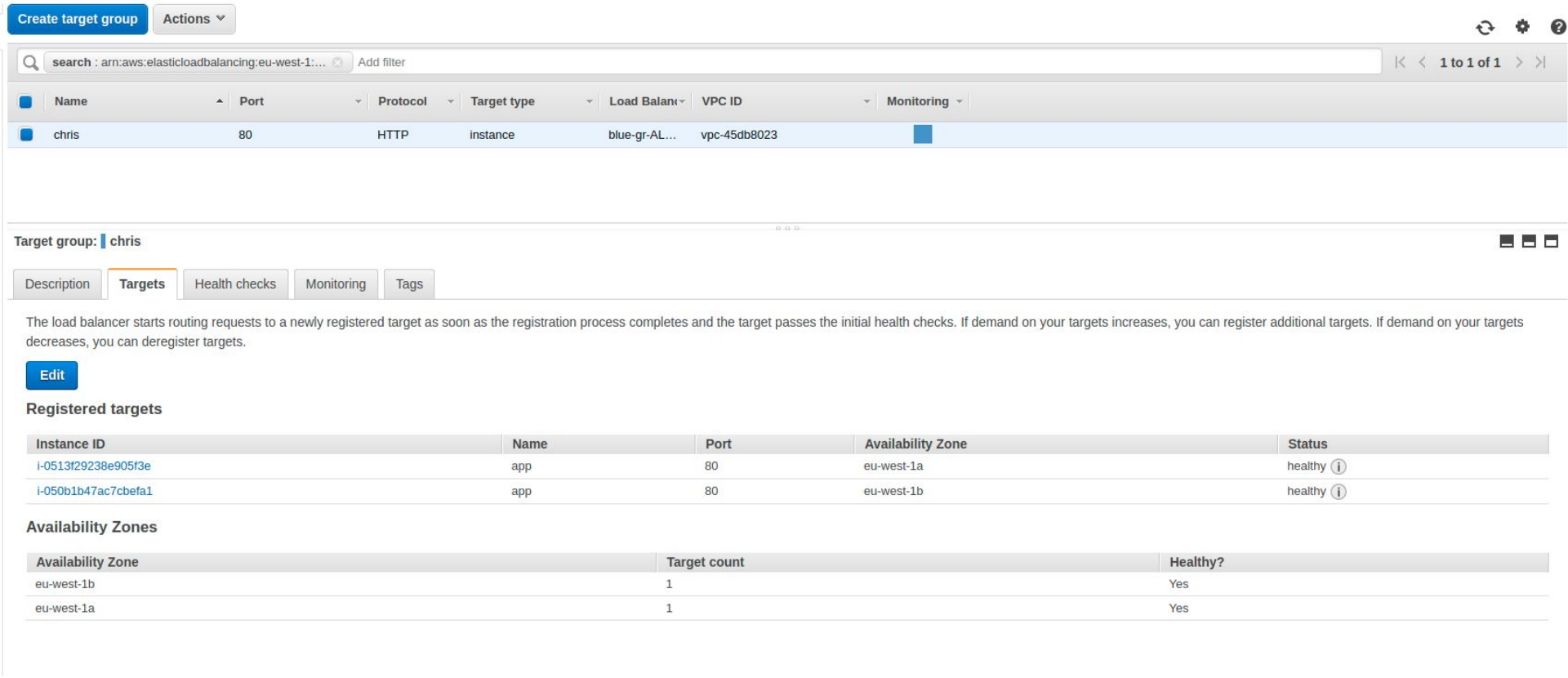
AWS Topology: ALB Rules during B/G



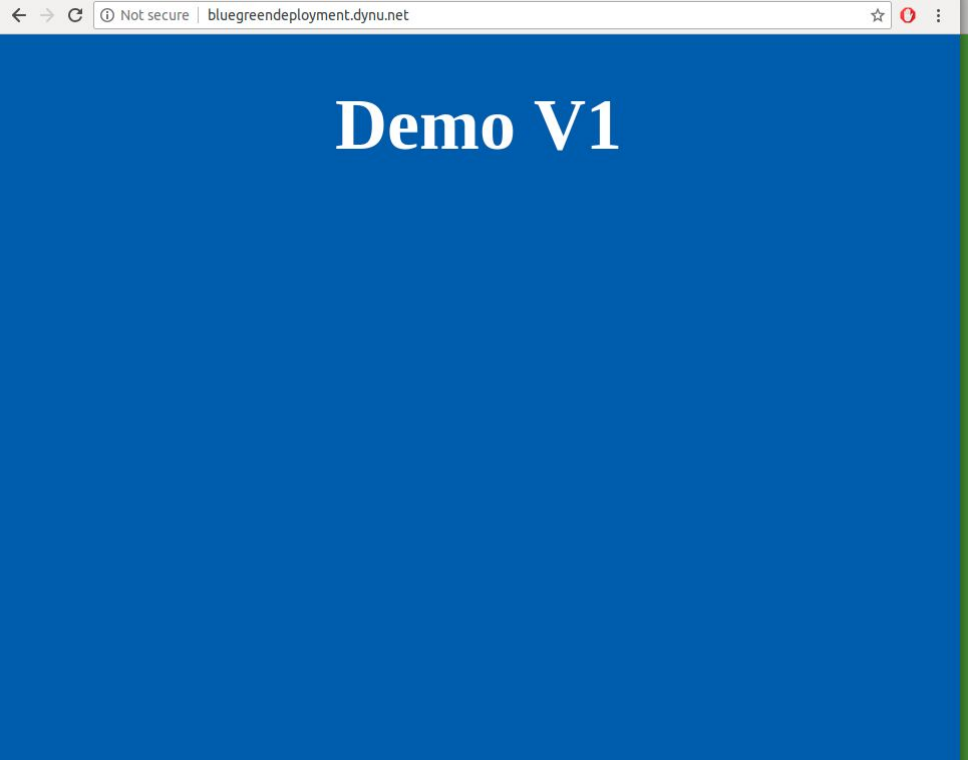
Quali

Quali

AWS Topology: Target Groups "chris" V2



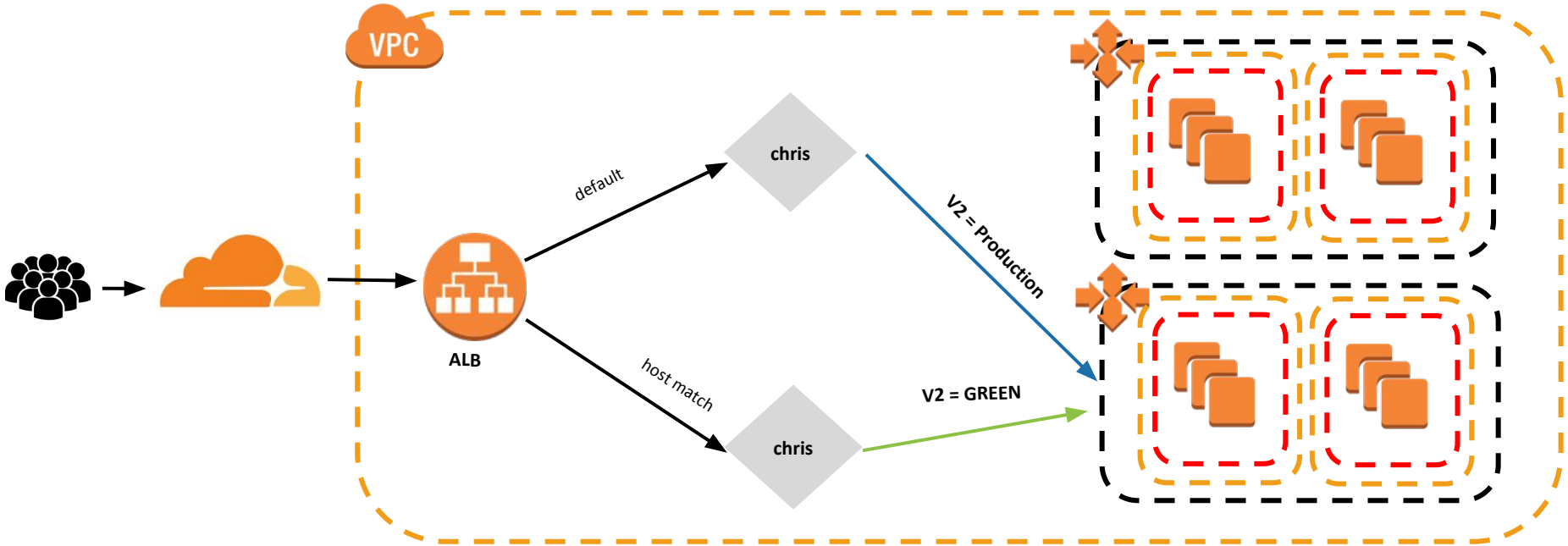
Quali


Not secure | bluegreendeployment.dynu.net
Demo V1


Not secure | green.bluegreendeployment.dynu.net
Demo V2

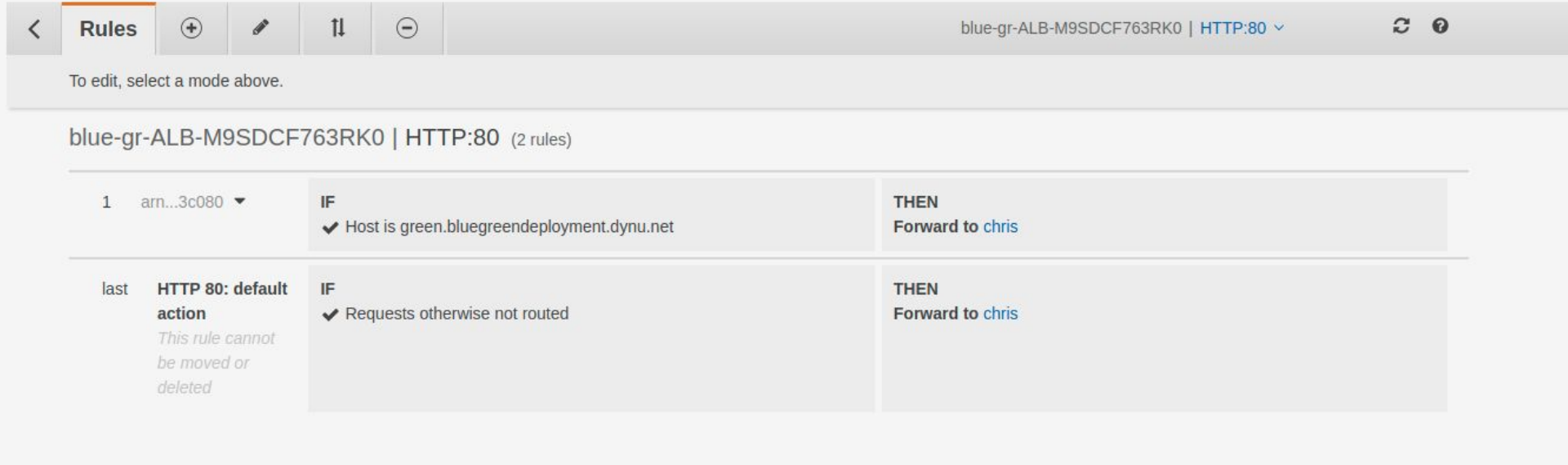
Quali

AWS Topology: Full exposure aka promotion to **BLUE**



Quali

AWS Topology: ALB Rules after B/G complete



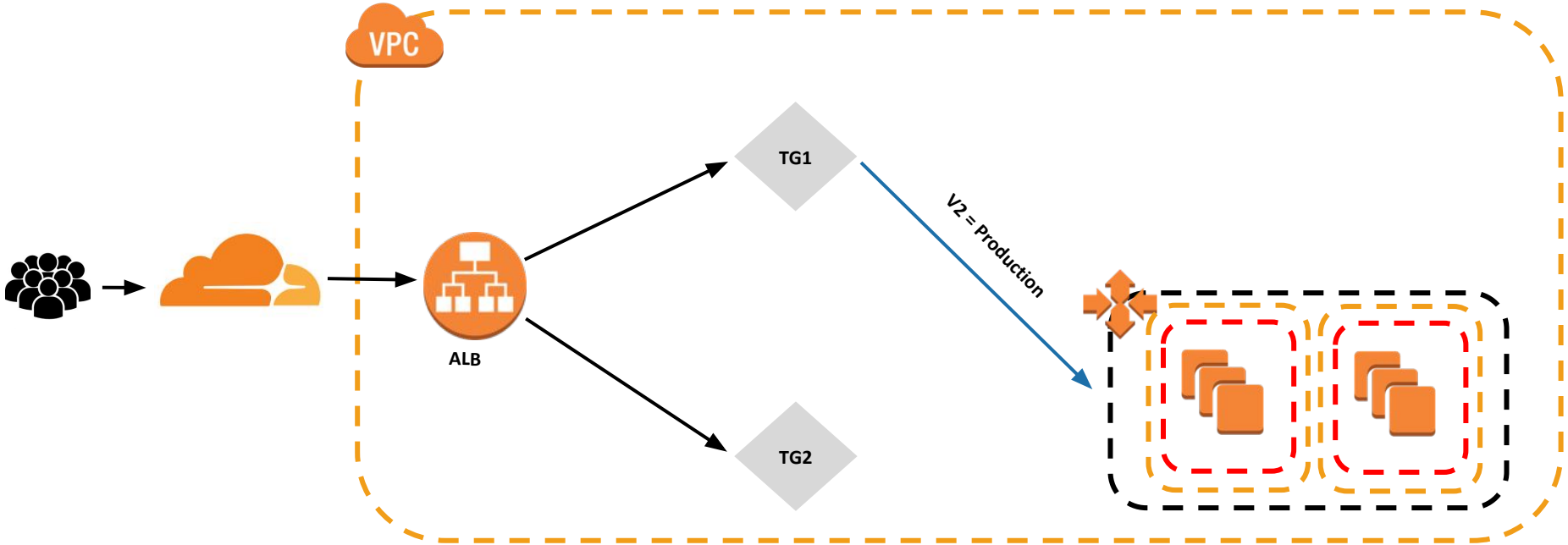
Quali


Not secure
bluegreendeployment.dynu.net
Demo V2


Not secure
green.bluegreendeployment.dynu.net
Demo V2

Quali

AWS Topology: V1 cleanups

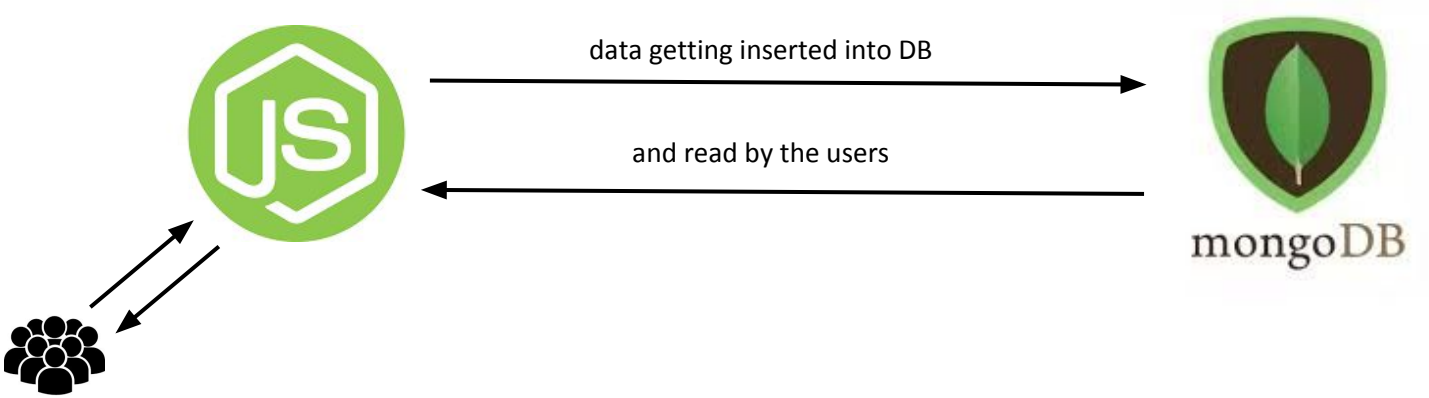


Quali

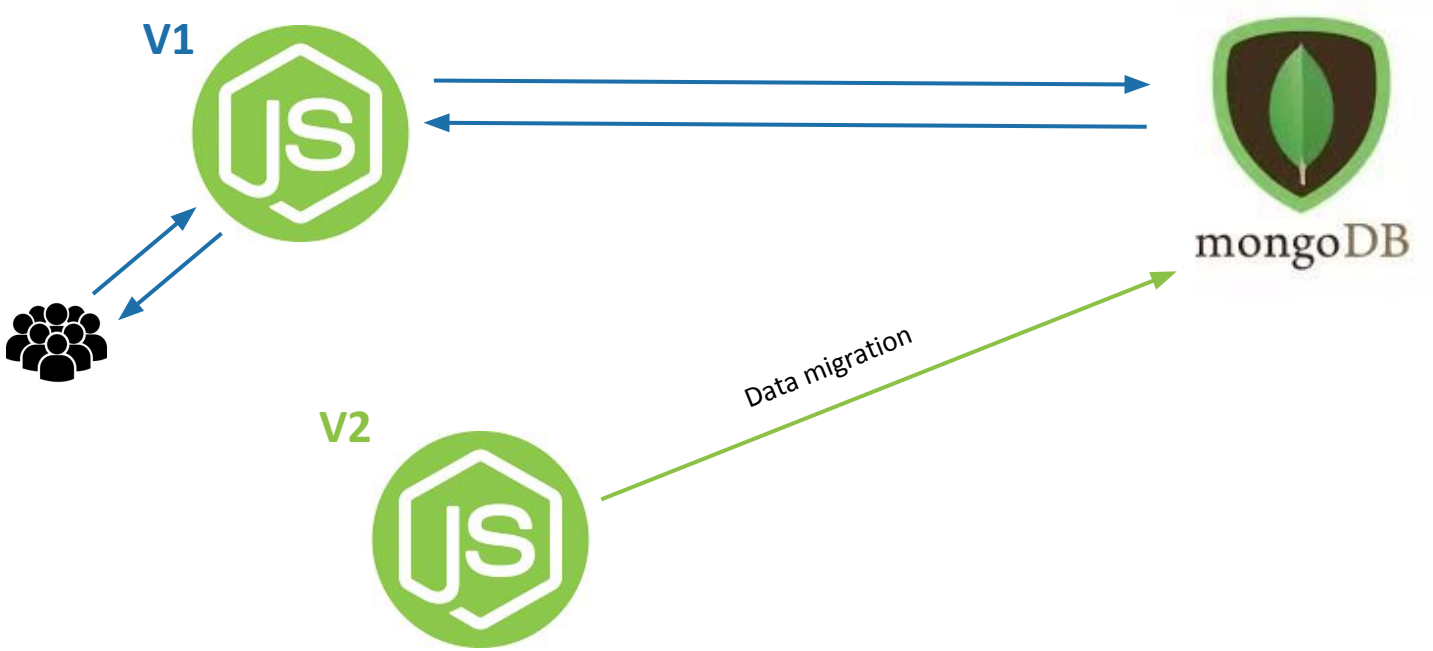
DB Backward/Forward compatibility

Quali

DB B/G: Initial state

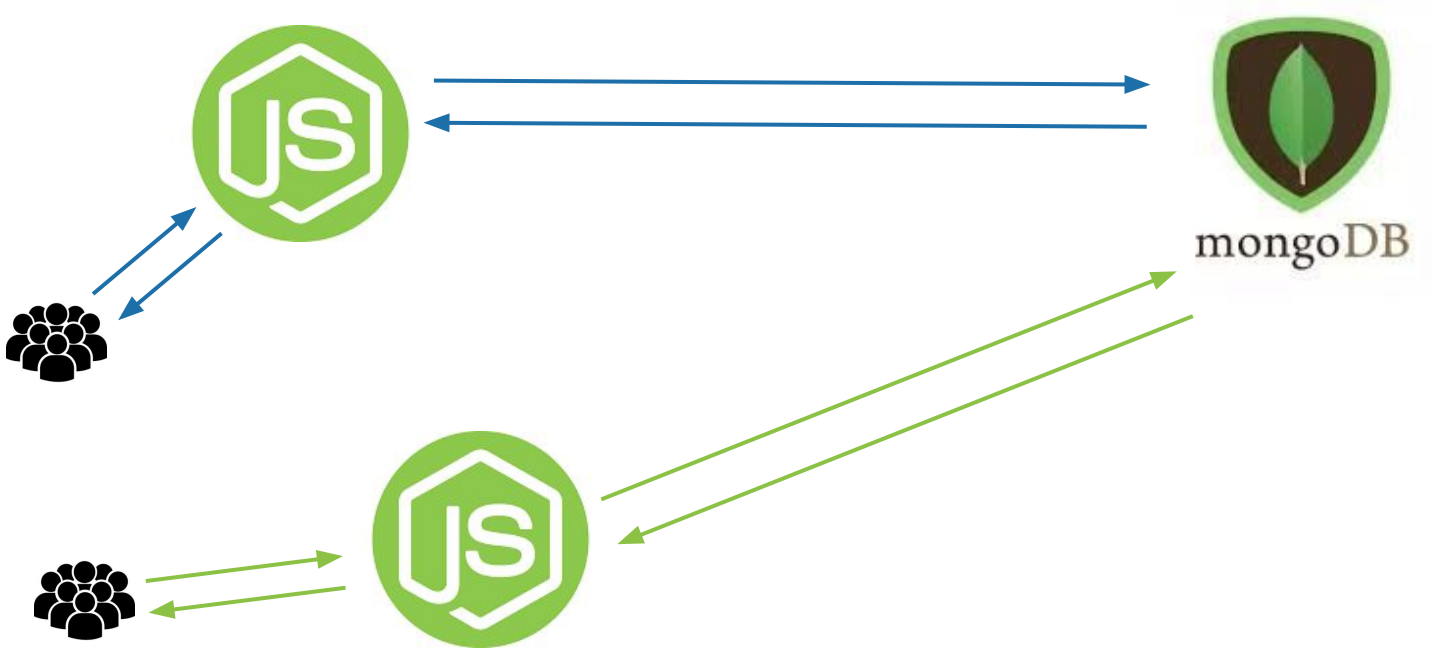


DB B/G: GREEN is up



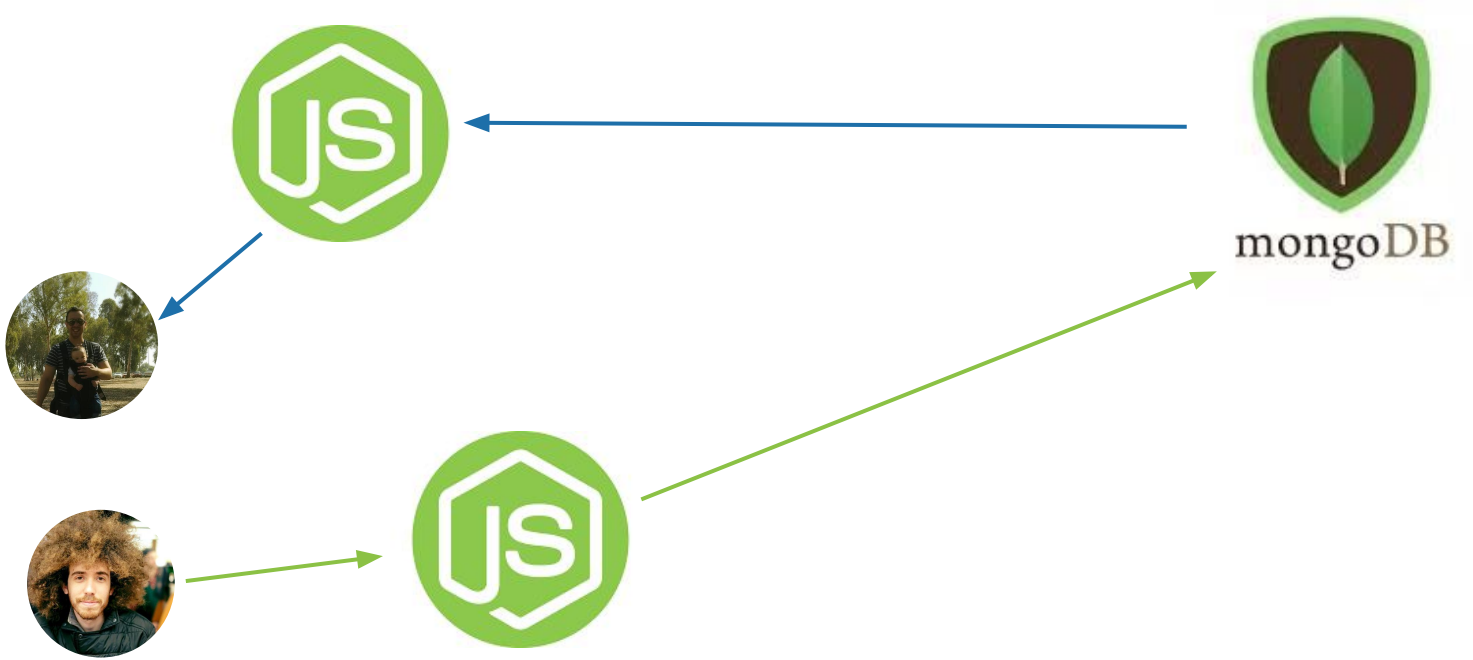
Quali

DB B/G: Traffic is getting redirected to **GREEN**



Quali

DB B/G: **BLUE** user reads **GREEN** data



Quali

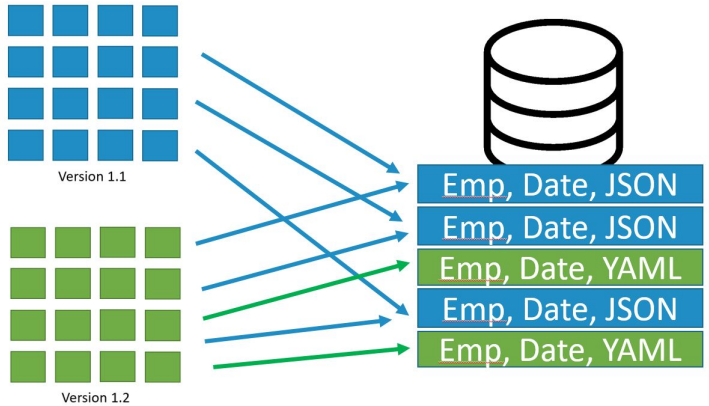
DB B/G: No comprende, amigo

mongoDB

Quali

DB B/G: Ways to handle

- Parallel change (just like in software development)
  - Both versions of the data coexists
  - Obsolete version is deleted once all the dependencies are updated
    - Easier with 'per environment B/G' since all services updated to new version at once



Quali

DB B/G: Ways to handle

- In between stage (When the data itself less important than the crash)
  - Add a way not to fall when expected data arrives without adding the feature itself
  - B/G
  - Add the feature itself and B/G again

Initial version

Initial version

Mock
for a change which "doesn't crush" the system

Mock

Real Feature

Quali

Thank you

Quali
systems