How containers have panned out

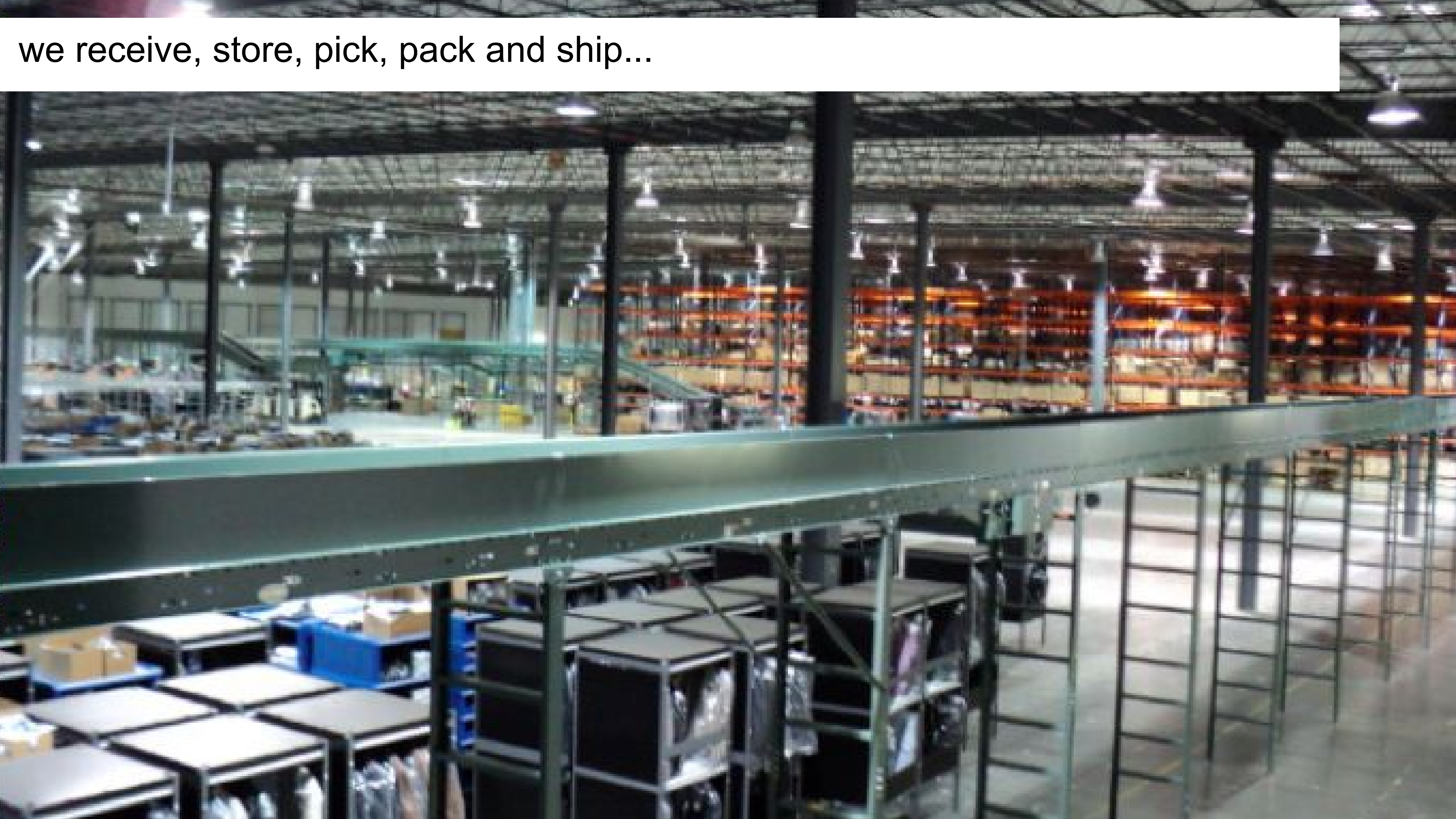
Adrian Trenaman, Raconteur & SVP Engineering, Gilt / HBC Digital
Q-Con, New York, June 2016
@gilttech @adrian_trenaman @hbc_tech

“What *competitive advantage* did containers give you?”

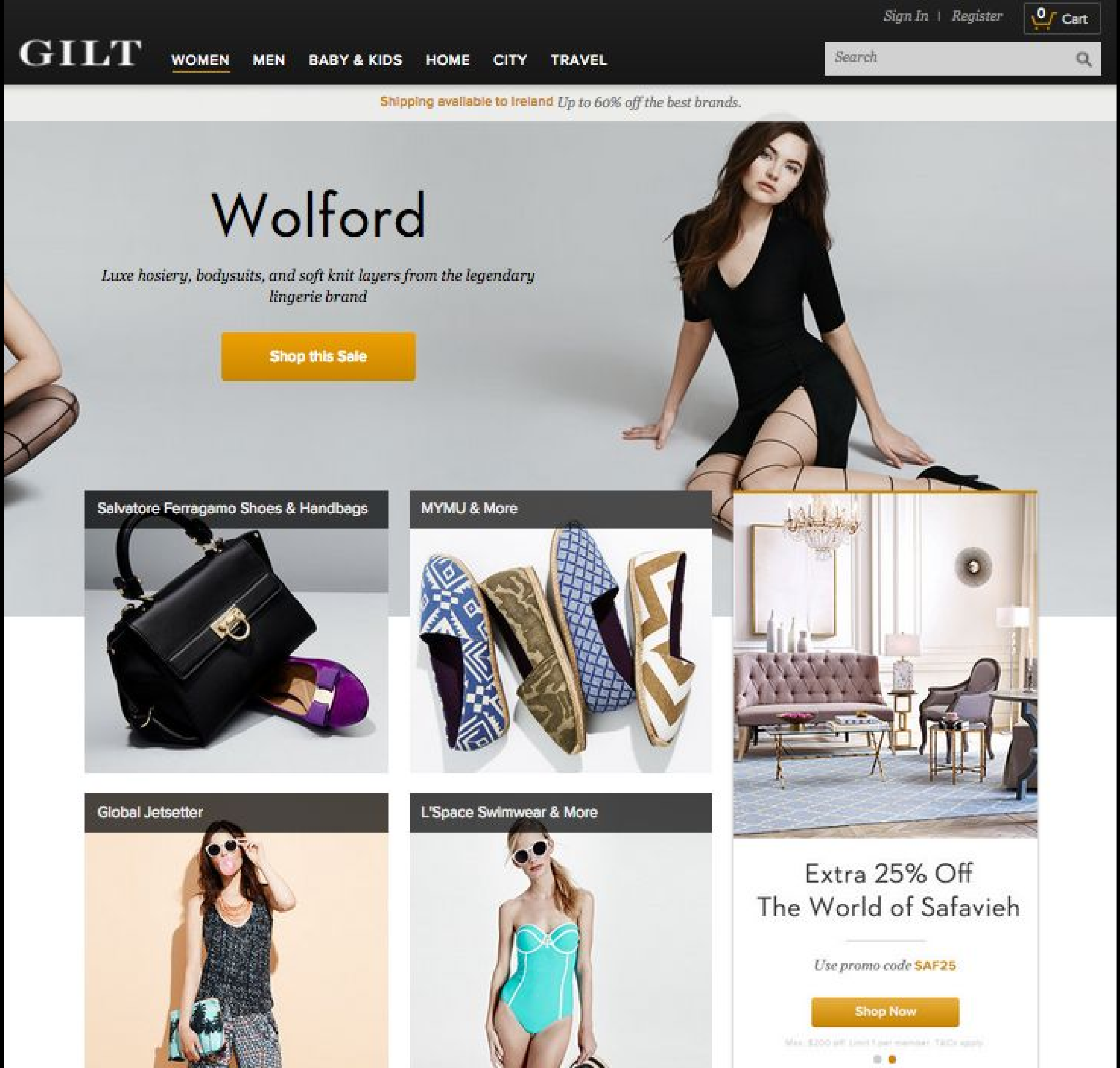
Gilt: luxury designer brands at discounted prices

we shoot the product in our studios

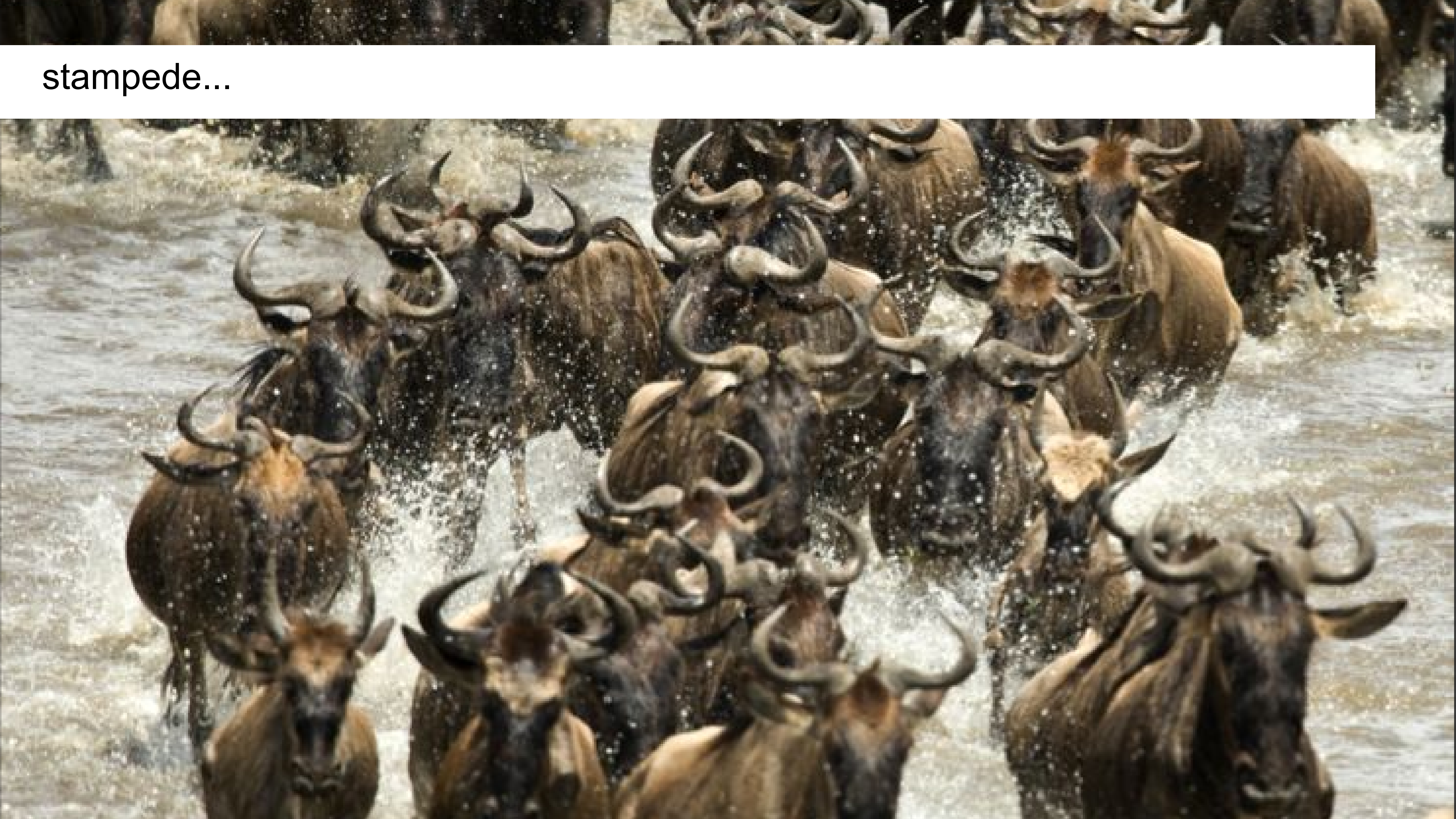
we receive, store, pick, pack and ship...

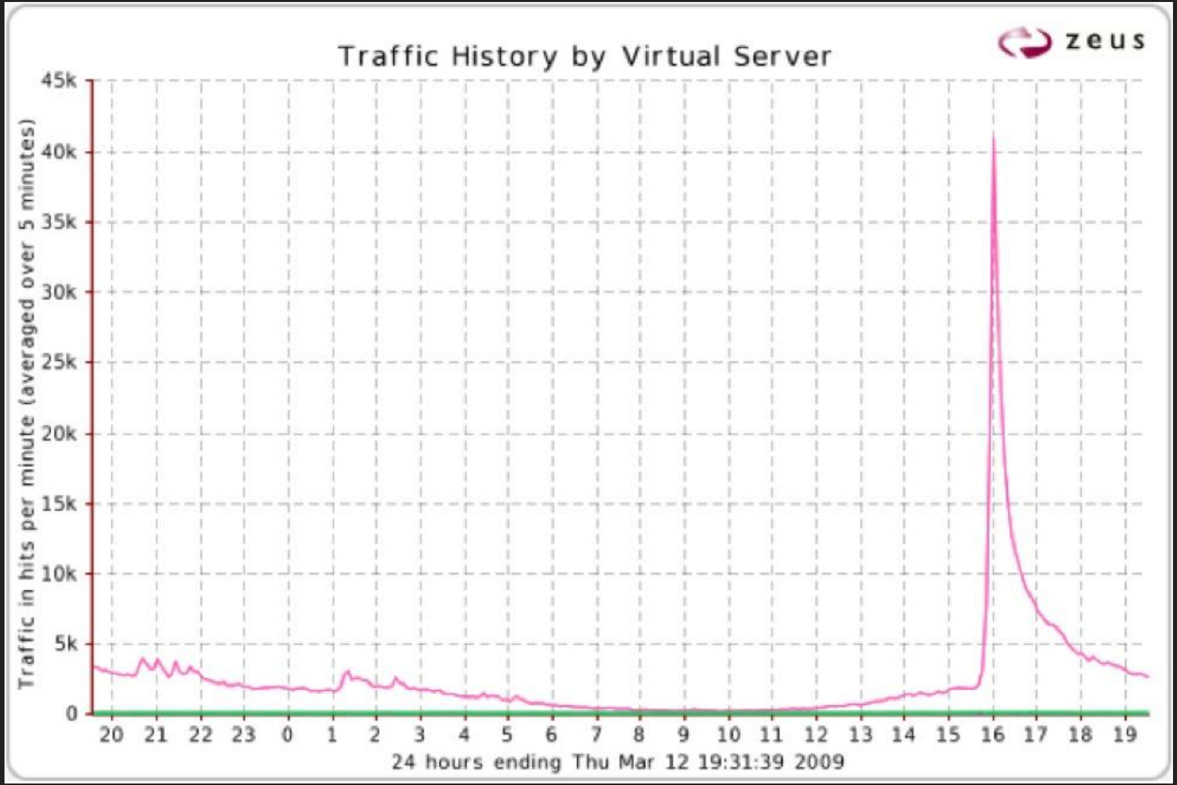


we sell every day at noon...



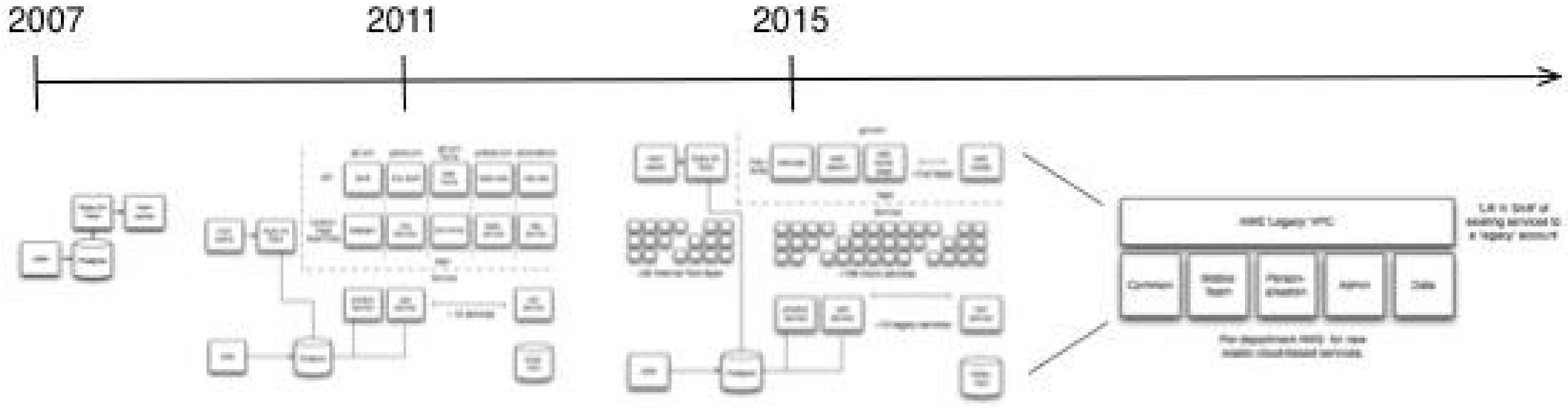
stampede...





this is what the stampede really looks like...

GILT: Evolution of a Micro-Services Architecture in a $1B startup



RoR
Monolith

Java,Scala
Loosely Typed Services

Lots of Small Apps (LOSA)
Lots of Micro Services

Get thee to the cloud.

A few big teams

Team-per-business.

Team-per-initiative.
Lots of initiatives.
Team & Individual autonomy.
Individual autonomy.
KPIs

Teams of Teams (Depts.)
Self-service Infrastructure by Dept.
Lots of Initiatives.
Lots of Autonomy.

$$m > n$$

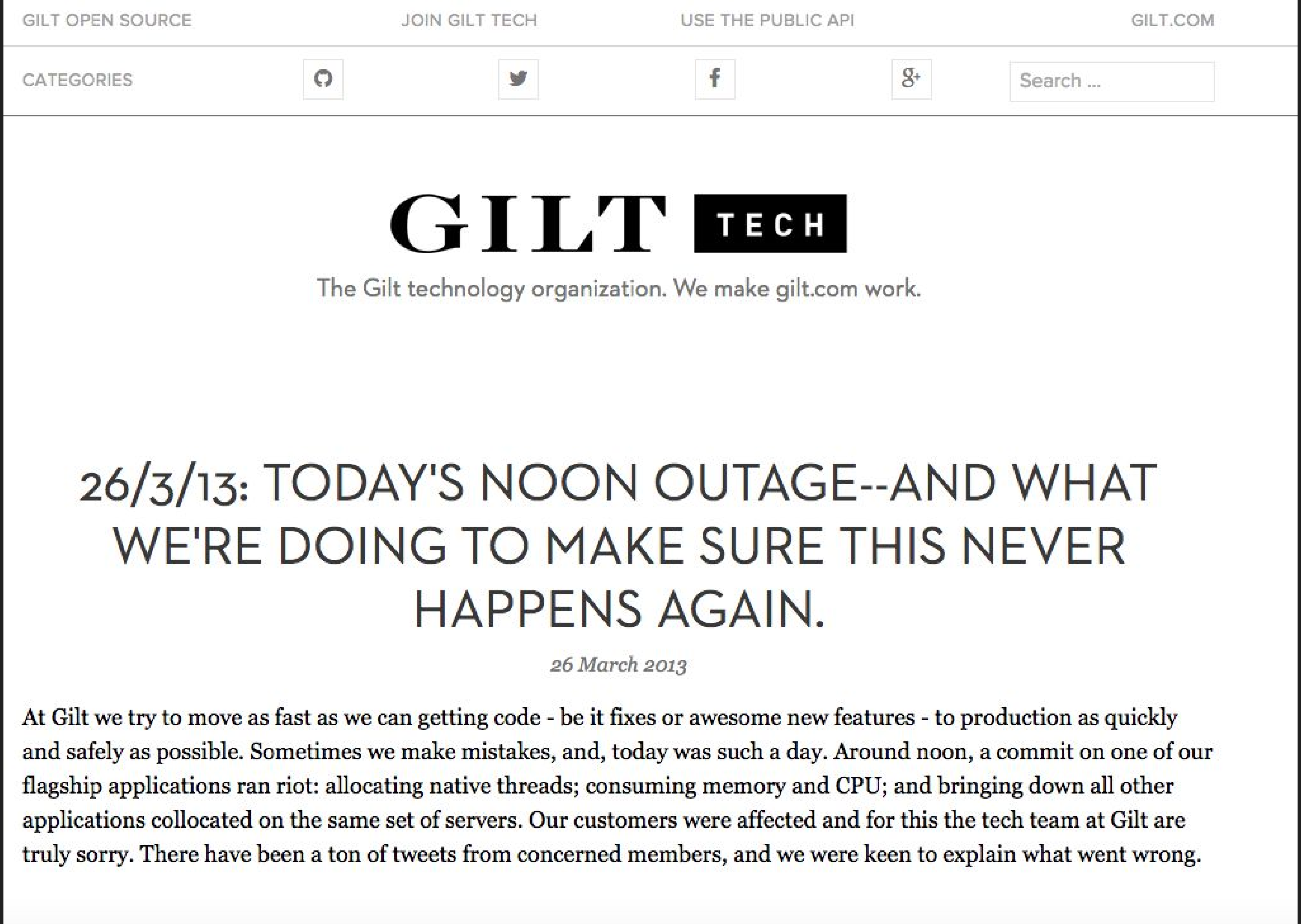
This is fundamentally a *packing* problem.

We have *n* machines, and we have *m* services to deploy.

1

It’s also an isolation problem

Any given service / team / engineer shouldn’t be able to take out someone else’s work in production.



GILT TECH

The Gilt technology organization. We make gilt.com work.

26/3/13: TODAY'S NOON OUTAGE--AND WHAT WE'RE DOING TO MAKE SURE THIS NEVER HAPPENS AGAIN.

26 March 2013

At Gilt we try to move as fast as we can getting code - be it fixes or awesome new features - to production as quickly and safely as possible. Sometimes we make mistakes, and, today was such a day. Around noon, a commit on one of our flagship applications ran riot: allocating native threads; consuming memory and CPU; and bringing down all other applications collocated on the same set of servers. Our customers were affected and for this the tech team at Gilt are truly sorry. There have been a ton of tweets from concerned members, and we were keen to explain what went wrong.

It’s also an *impedance mismatch* problem.

Developers often think of machines as something that’s all theirs, magically provided by the hardware fairy.

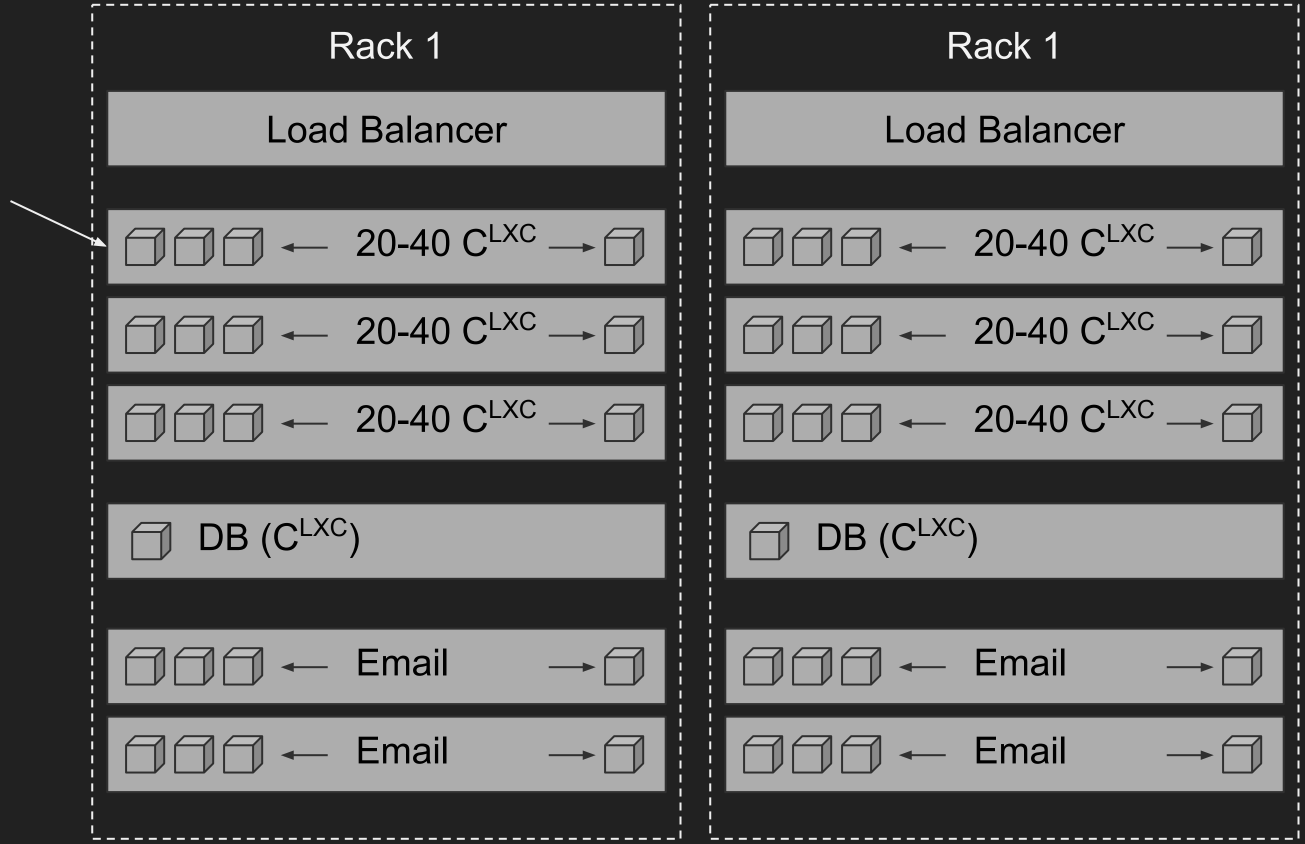
LXC

Leveraging LXC in Tokyo for Gilt Japan

16xCPU, 128GB RAM, 900GB Disk.

Ubuntu 12.04 (→ 16.04)

~220 C^{LXC} in total.

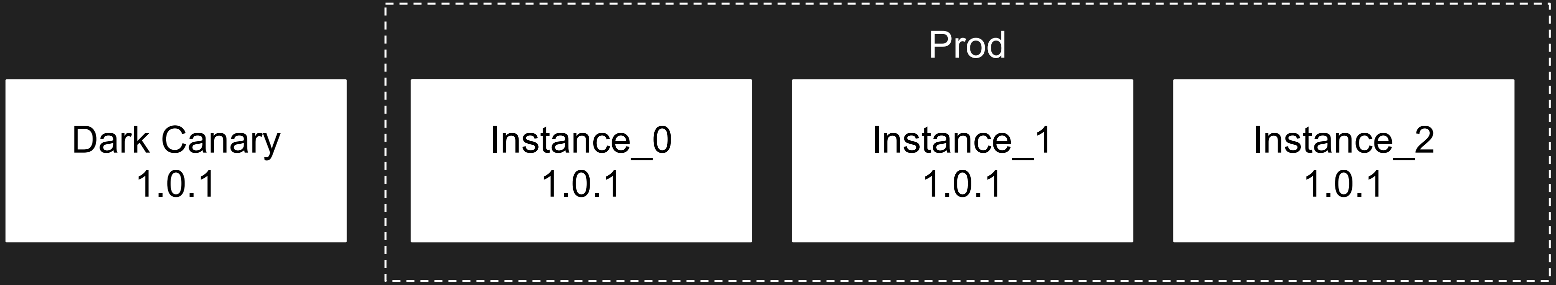


LXC @ Gilt Japan

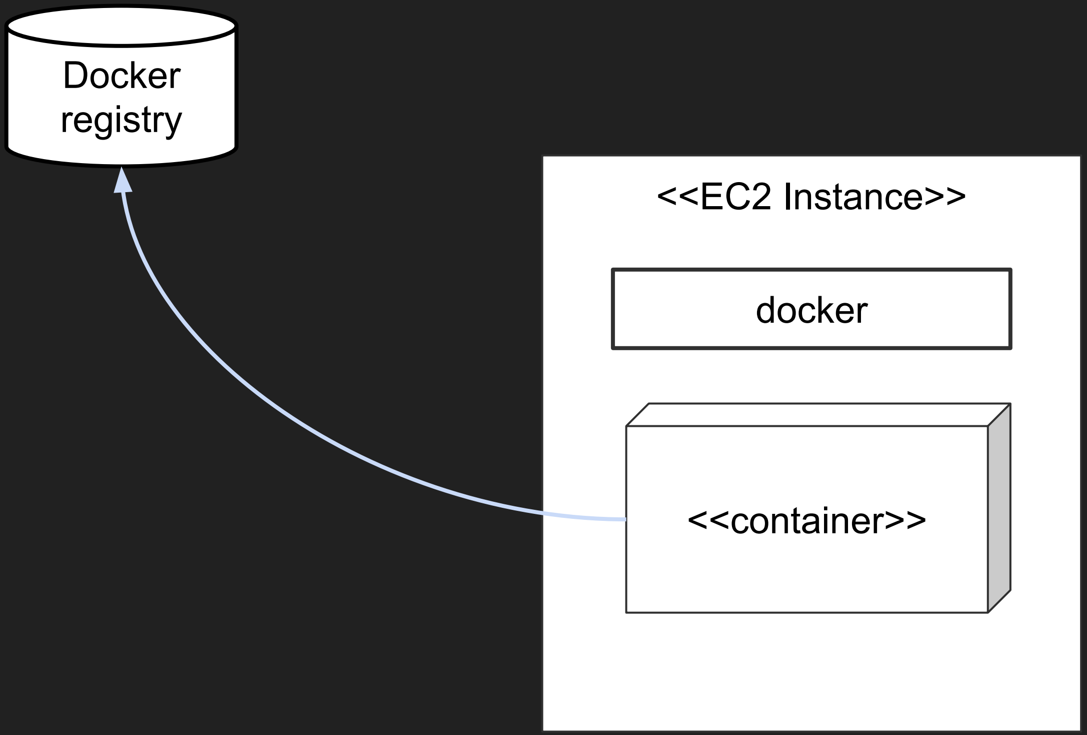
- ✔ Scalable, performant use of machine resources.
- ✔ Solves the impedance mismatch: developers see ‘a machine’
- ✔ Limits the damage a single engineer can do.
- ✔ Infra/Devops engineer embedded into a tightly knit engineering team
- ✕ Static infrastructure
- ✕ Potential for resource hogging

Immutable Deployment With Docker

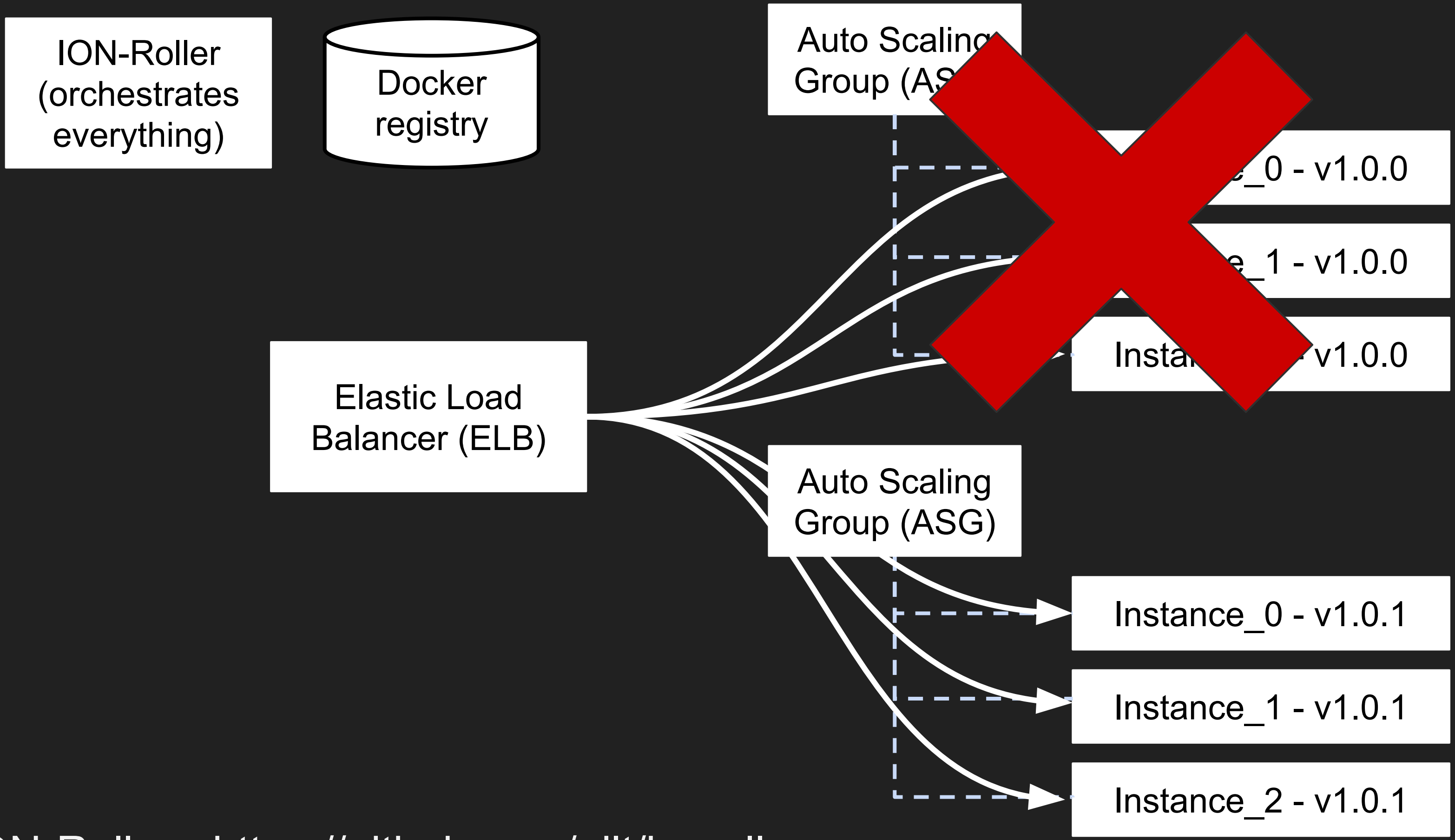
I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION
memegenerator.net



Core idea #1: dark canaries, canaries, release, roll-back.



Core idea #2: One container per host / EC2 instance



ION-Roller - https://github.com/gilt/ionroller

ION-Roller deployment:

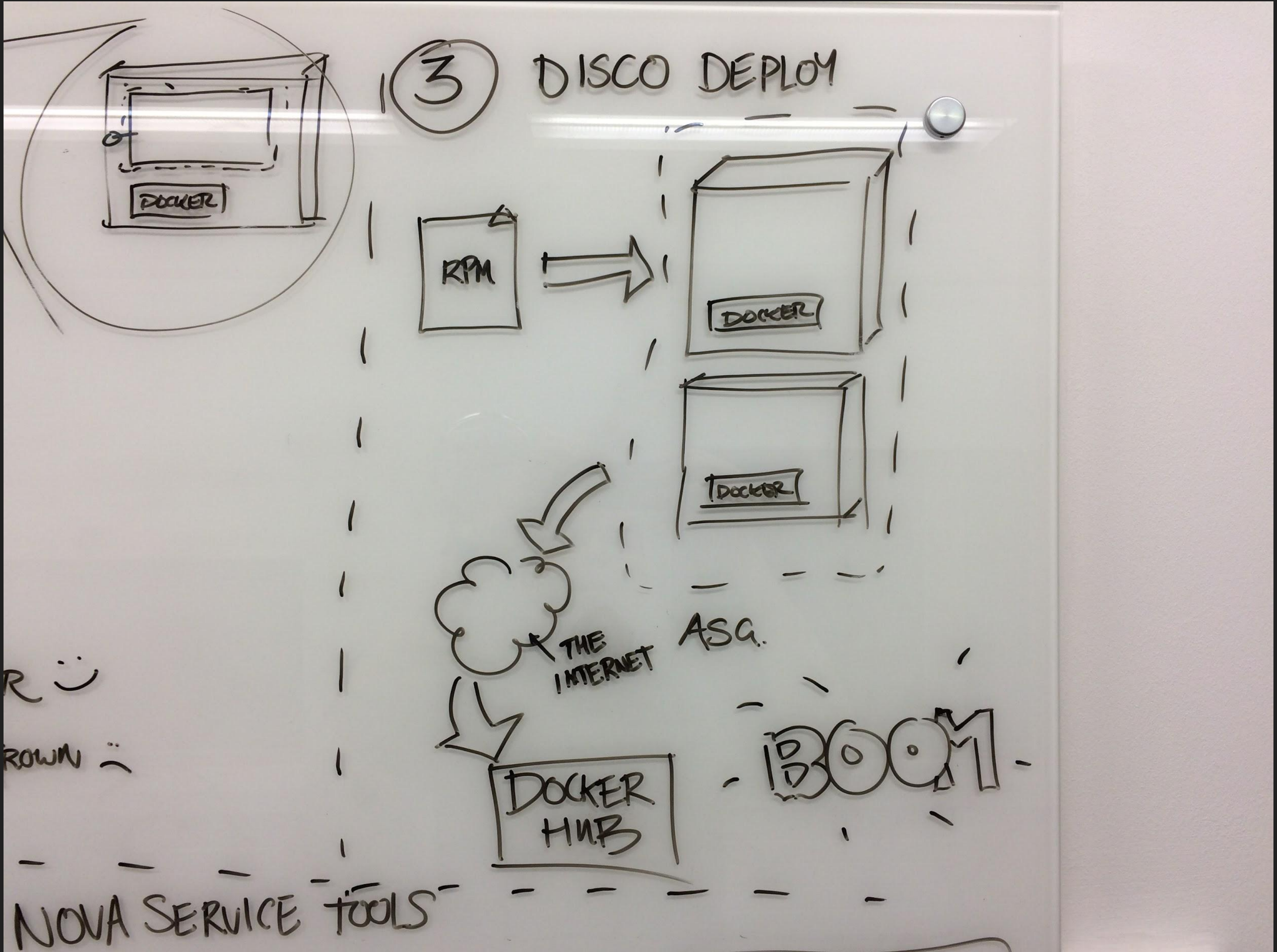
✔ Immutable deployment :)

✔ DNS + ELB traffic migration :)

× Slow to set up / tear down environments :(

× Potentially expensive under continuous deployment :(

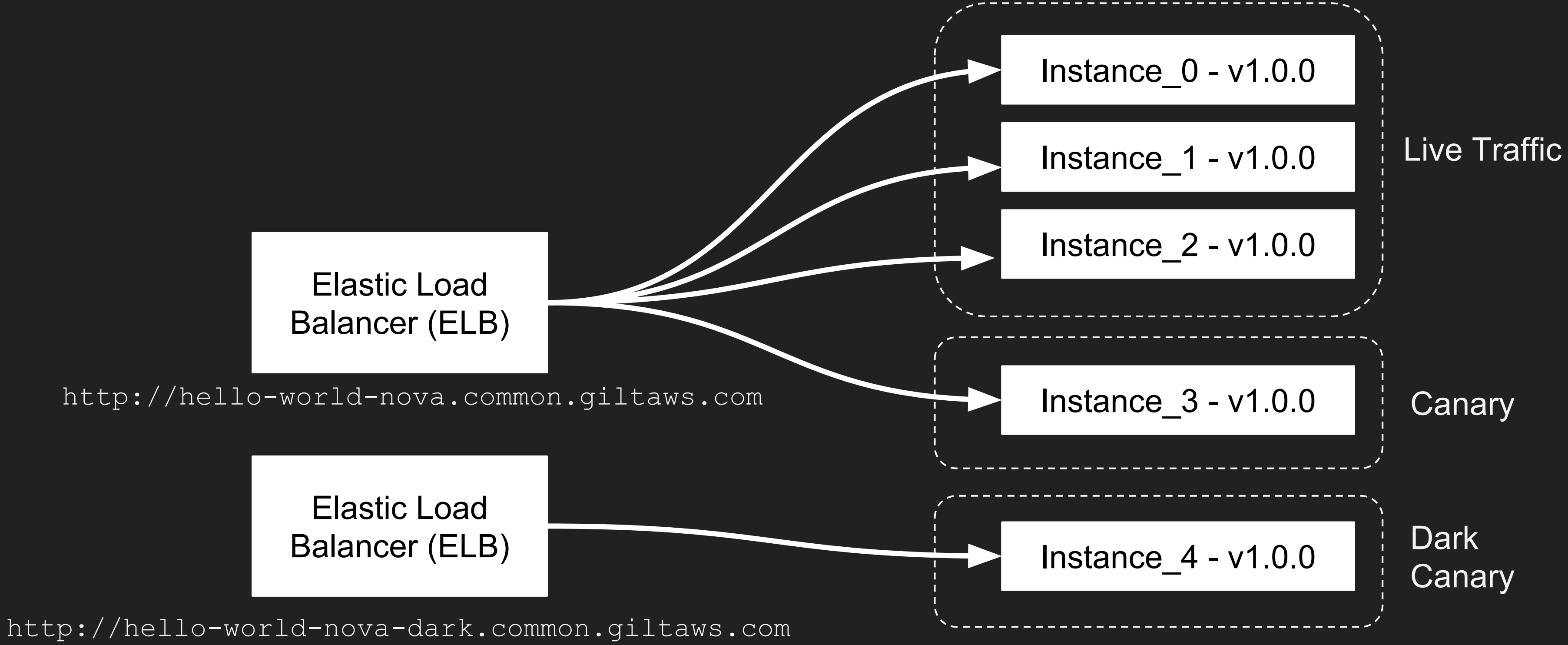
× Open-source, but in-house. ‘A snowflake in the making’ ❄


DOCKER
③ DISCO DEPLOY
RPM
DOCKER
DOCKER
THE INTERNET
ASG.
DOCKER HUB
BOOM
R
ROWN
NOVA SERVICE TOOLS

6

“We could solve this now, or, just wait six months, and Amazon will provide a solution”

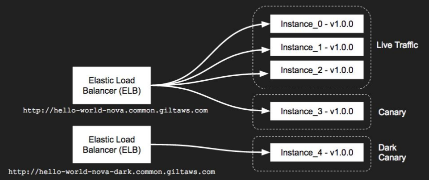
Andrey Kartashov, Distinguished Engineer, Gilt.


Elastic Load Balancer (ELB)
http://hello-world-nova.common.giltaws.com
Elastic Load Balancer (ELB)
http://hello-world-nova-dark.common.giltaws.com
Instance_0 - v1.0.0
Instance_1 - v1.0.0
Instance_2 - v1.0.0
Live Traffic
Instance_3 - v1.0.0
Canary
Instance_4 - v1.0.0
Dark Canary

github.com/gilt/nova- deployment patterns nova.yml

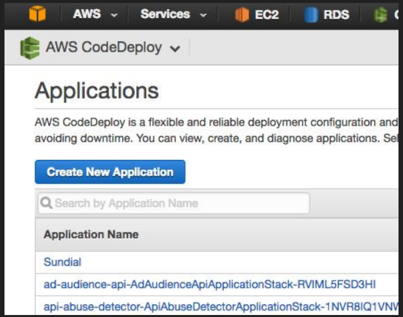
templates

```
$> nova stack create production
```


Elastic Load Balancer (ELB)
http://hello-world-nova.common.giltaws.com
Elastic Load Balancer (ELB)
http://hello-world-nova-dark.common.giltaws.com
Instance_0 - v1.0.0
Instance_1 - v1.0.0
Instance_2 - v1.0.0
Live Traffic
Instance_3 - v1.0.0
Canary
Instance_4 - v1.0.0
Dark Canary

CloudFormation

CodeDeploy

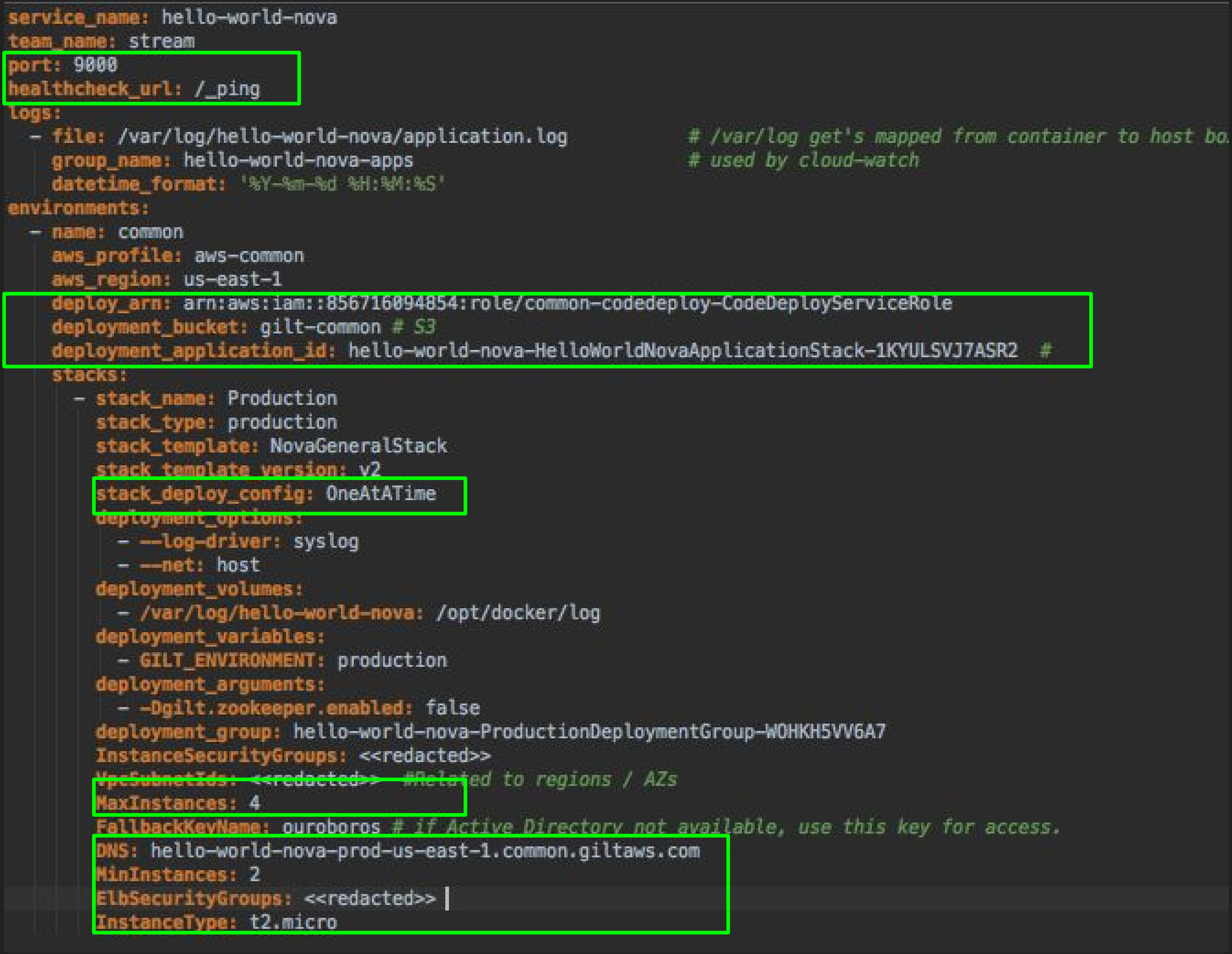


Applications

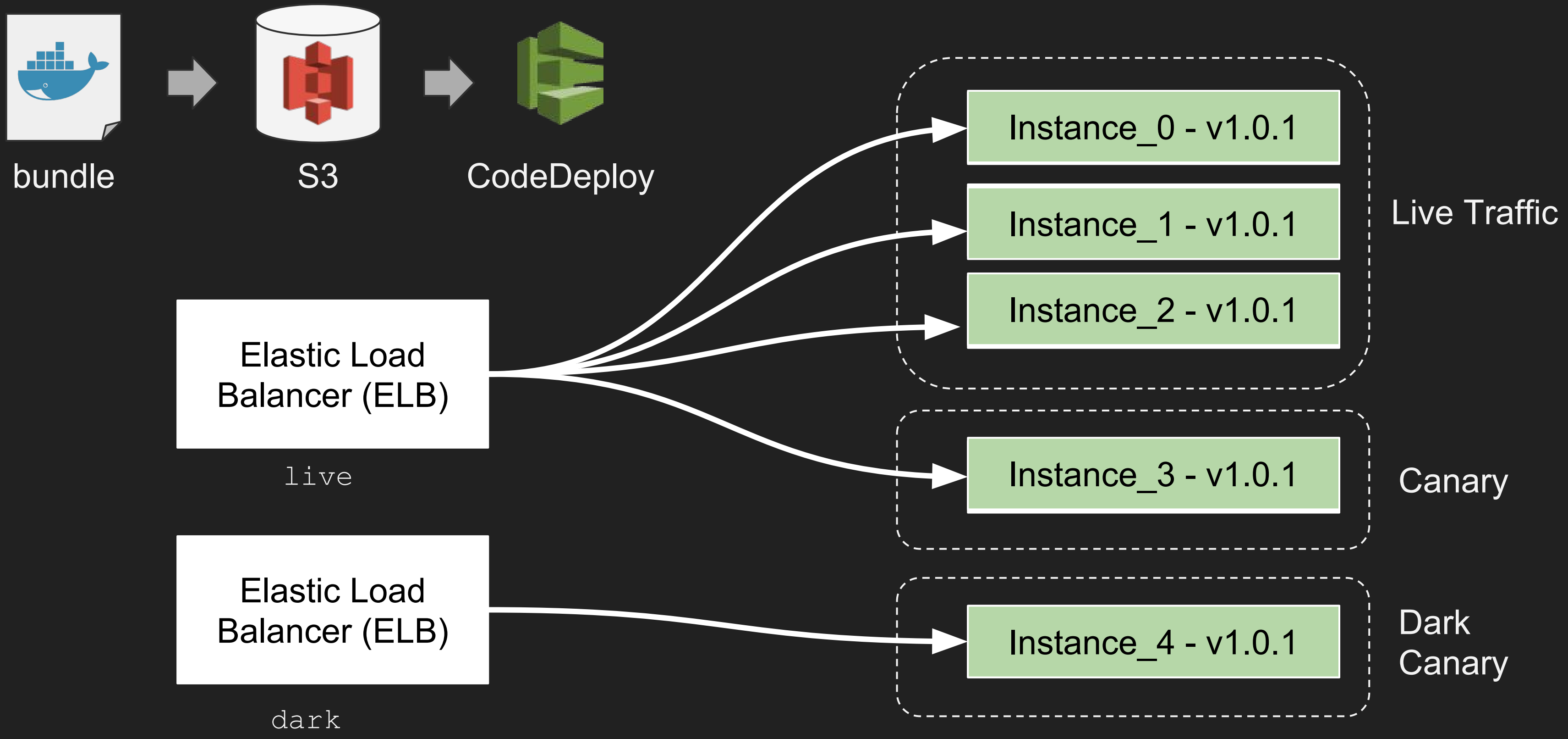
AWS CodeDeploy is a flexible and reliable deployment configuration and
avoiding downtime. You can view, create, and diagnose applications. Se

Application Name
Sundial
ad-audience-api-AdAudienceApiApplicationStack-RVIML5FSD3HI
api-abuse-detector-ApiAbuseDetectorApplicationStack-1NVR8IQ1VNV

github.com/gilt/nova - creating environments



```yaml
service_name: hello-world-nova
team_name: stream
port: 9000
healthcheck_url: /_ping
logs:
  - file: /var/log/hello-world-nova/application.log          # /var/log get's mapped from container to host bo
    group_name: hello-world-nova-apps                         # used by cloud-watch
    datetime_format: '%Y-%m-%d %H:%M:%S'
environments:
  - name: common
    aws_profile: aws-common
    aws_region: us-east-1
    deploy_arn: arn:aws:iam::856716094854:role/common-codedeploy-CodeDeployServiceRole
    deployment_bucket: gilt-common # S3
    deployment_application_id: hello-world-nova-HelloWorldNovaApplicationStack-1KYULSVJ7ASR2  #
    stacks:
      - stack_name: Production
        stack_type: production
        stack_template: NovaGeneralStack
        stack_template_version: v2
        stack_deploy_config: OneAtATime
        deployment_options:
          - --log-driver: syslog
          - --net: host
        deployment_volumes:
          - /var/log/hello-world-nova: /opt/docker/log
        deployment_variables:
          - GILT_ENVIRONMENT: production
        deployment_arguments:
          - -Dgilt.zookeeper.enabled: false
        deployment_group: hello-world-nova-ProductionDeploymentGroup-WOHKH5VV6A7
        InstanceSecurityGroups: <<redacted>>
        VpcSubnetIds: <<redacted>>  #Related to regions / AZs
        MaxInstances: 4
        FallbackKeyName: ouroboros # if Active Directory not available, use this key for access.
        DNS: hello-world-nova-prod-us-east-1.common.giltaws.com
        MinInstances: 2
        ElbSecurityGroups: <<redacted>>
        InstanceType: t2.micro
```



```
$> nova deploy common Production
1.0.1
```

github.com/gilt/nova- deployment

Nova deployment:

✓ No docker registry (shock! gasp!) :)

✓ Less boilerplate code :)

✓ Immutable deployment (on mutable infrastructure) :)

✓ Leverage AWS tooling :)

? Next up? Integrate with Code Pipeline :?

Fighting bit rot, *chaos-monkey* style

With long running mutable AMIs, it’s possible for bit-rot to creep in.

Think *security vulnerability*.

Novel approach: every day, kill and restart your oldest AMI randomly.

✔ Pick up latest AMI with fixes

✔ Fail early, noisily and loudly if there’s a problem *without* a production outage.

Vulnerability in container? Cut a new release against a fixed base-image.

Explorations in ECS

Sundial - running batch jobs with Docker & ECS

✔ Job dependencies (allows us to break large jobs into smaller jobs)

✔ Ease of viewing logs and debugging failures

✔ Automatic rescheduling of failed tasks within a job

✔ Isolation between jobs

✔ Low cost of setup and maintenance, as few moving parts as possible for Infra teams to manage

http://github.com/gilt/sundial

Sundial: processes

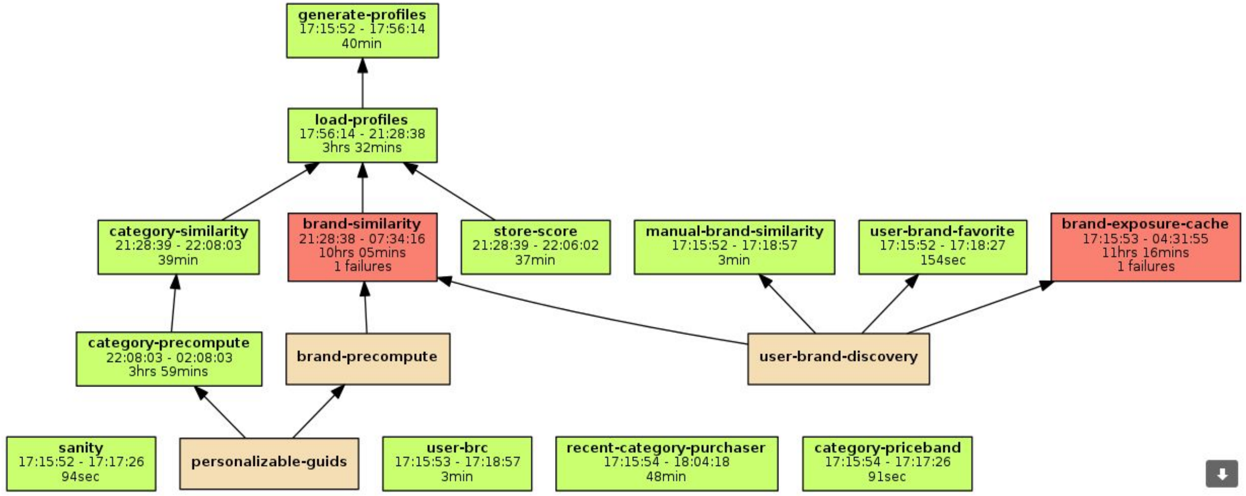
A *process* in Sundial is a grouping of tasks (jobs) with dependencies between them.

> **Schedule**: Either manually triggered, continuous schedule, or cron schedule
>
> **Overlap strategy**: if previous iteration hasn't completed, do we
>
> > Wait
> > Terminate previous iteration
> > Run in parallel

When a process kicks off, all tasks with no dependencies kick off.

When a task finishes, any tasks blocked by that task will kick off.

Process Summary: cerebro-batch


generate-profiles 17:15:52 - 17:56:14 40min
load-profiles 17:56:14 - 21:28:38 3hrs 32mins
category-similarity 21:28:39 - 22:08:03 39min
brand-similarity 21:28:38 - 07:34:16 10hrs 05mins 1 failures
store-score 21:28:39 - 22:06:02 37min
manual-brand-similarity 17:15:52 - 17:18:57 3min
user-brand-favorite 17:15:52 - 17:18:27 154sec
brand-exposure-cache 17:15:53 - 04:31:55 11hrs 16mins 1 failures
category-precompute 22:08:03 - 02:08:03 3hrs 59mins
brand-precompute
user-brand-discovery
sanity 17:15:52 - 17:17:26 94sec
personalizable-guids
user-brc 17:15:53 - 17:18:57 3min
recent-category-purchaser 17:15:54 - 18:04:18 48min
category-priceband 17:15:54 - 17:17:26 91sec

Process for cerebro-batch has failed after 14 hours with process ID 0fcf4de3-da67-4909-b7a3-16c3013faadb.

Task Summary

brand-exposure-cache	Failed after 1 attempt	11 hours
brand-precompute	Did Not Run	
brand-similarity	Failed after 1 attempt	10 hours
category-precompute	Suceeded after 1 attempt	3 hours
category-priceband	Suceeded after 1 attempt	91 seconds
category-similarity	Suceeded after 1 attempt	39 minutes
generate-profiles	Suceeded after 1 attempt	40 minutes
load-profiles	Suceeded after 1 attempt	3 hours
manual-brand-similarity	Suceeded after 1 attempt	3 minutes
personalizable-guids	Did Not Run	
recent-category-purchaser	Suceeded after 1 attempt	48 minutes
sanity	Suceeded after 1 attempt	94 seconds
store-score	Suceeded after 1 attempt	37 minutes
user-brand-discovery	Did Not Run	
user-brand-favorite	Suceeded after 1 attempt	2 minutes

ECS is getting really attractive...

We're prototyping using for customer-facing services on our mobile team:

- ✔ Less configuration / moving parts than MST/Nova
- ✔ Automatic rollout
- ✔ Easy integration with IAM, CloudWatch, ECR

But:

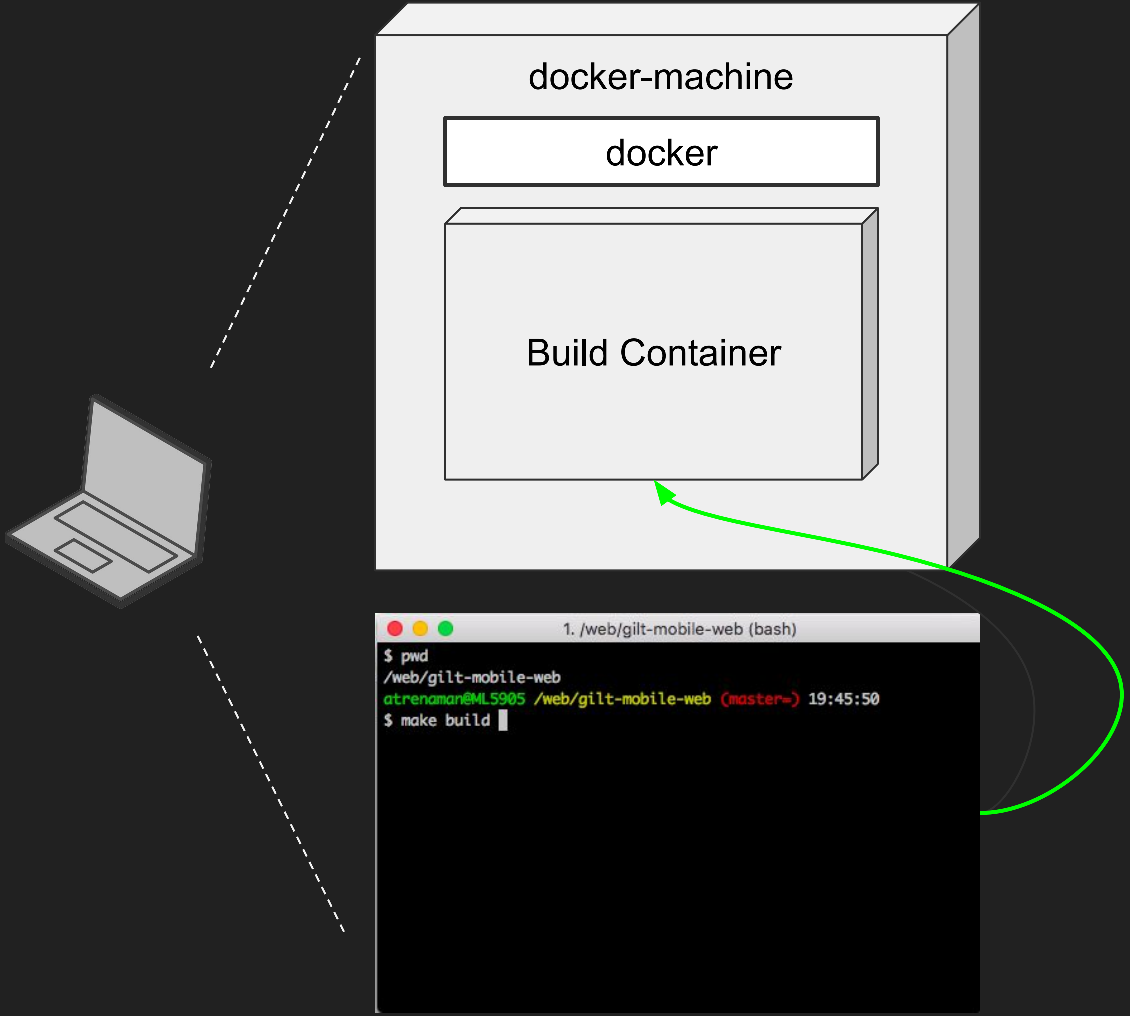
- ✕ IAM roles at instance level *not* container level
- ✕ Tension between CF stack templates and deployment updates
- ✕ ELBs require fixed ports: we want to define the listening port.

Docker as Build Platform

Using docker as a local build platform

The problem: keeping up with different versions / combinations of build tools is crazy hard.

Why not use Docker for build, using a versioned build container?



```
$ pwd
/web/gilt-mobile-web
atrenaman@ML5905 /web/gilt-mobile-web (master=) 19:45:50
$ make build
```

Lesson #1

Containers have let us separate *what* we deploy (JVM, RoR, …)
from *how* and *where* we deploy it (mst, nova, EC2, Triton)
and This Is Good.

Lesson #2

It’s still a wild-west in terms of how containers are deployed.
Different teams have different needs - be sensitive to that.

Lesson #3

Seek immutability in the container, not in the stack.

Lesson #4

The competitive advantage: containers let us deploy quickly, frequently and safely to production, which help us innovate faster.

That’s it.

#thanks @adrian_trenaman
@gilttech @hbc_tech