duolingo

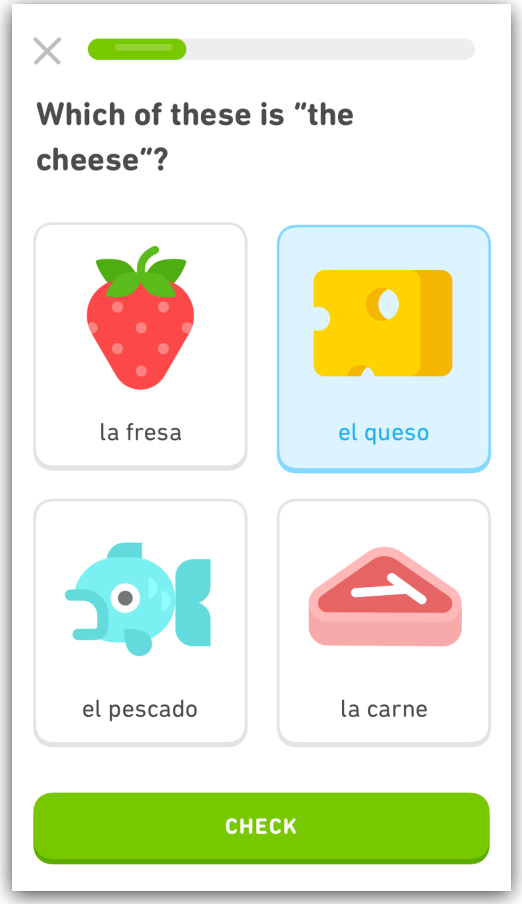
Microservice Journey

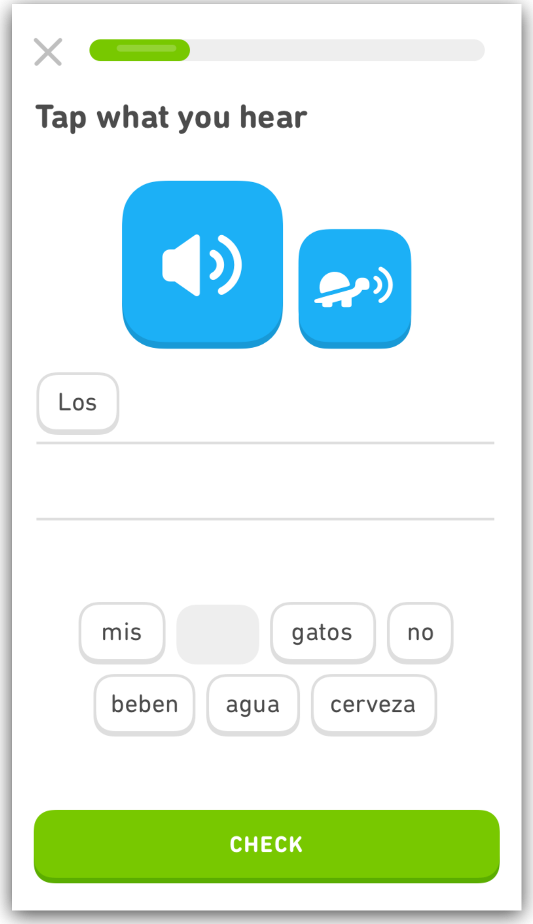
Ohio LinuxFest 2019

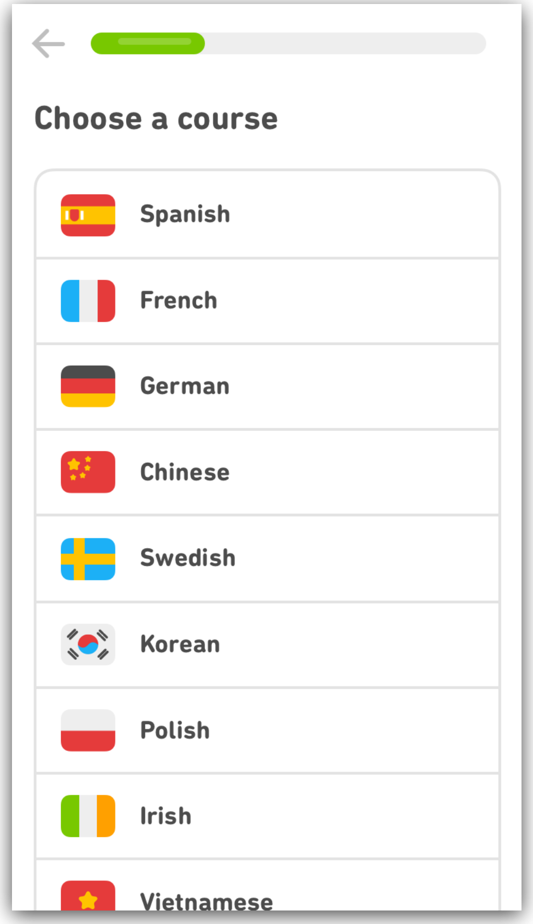
Max Blaze

Free and accessible language education for all

duolingo







The most downloaded education app in the world

duolingo

30+ languages / 80+ courses

duolingo

34

Hours of Duolingo

=

1

University Semester

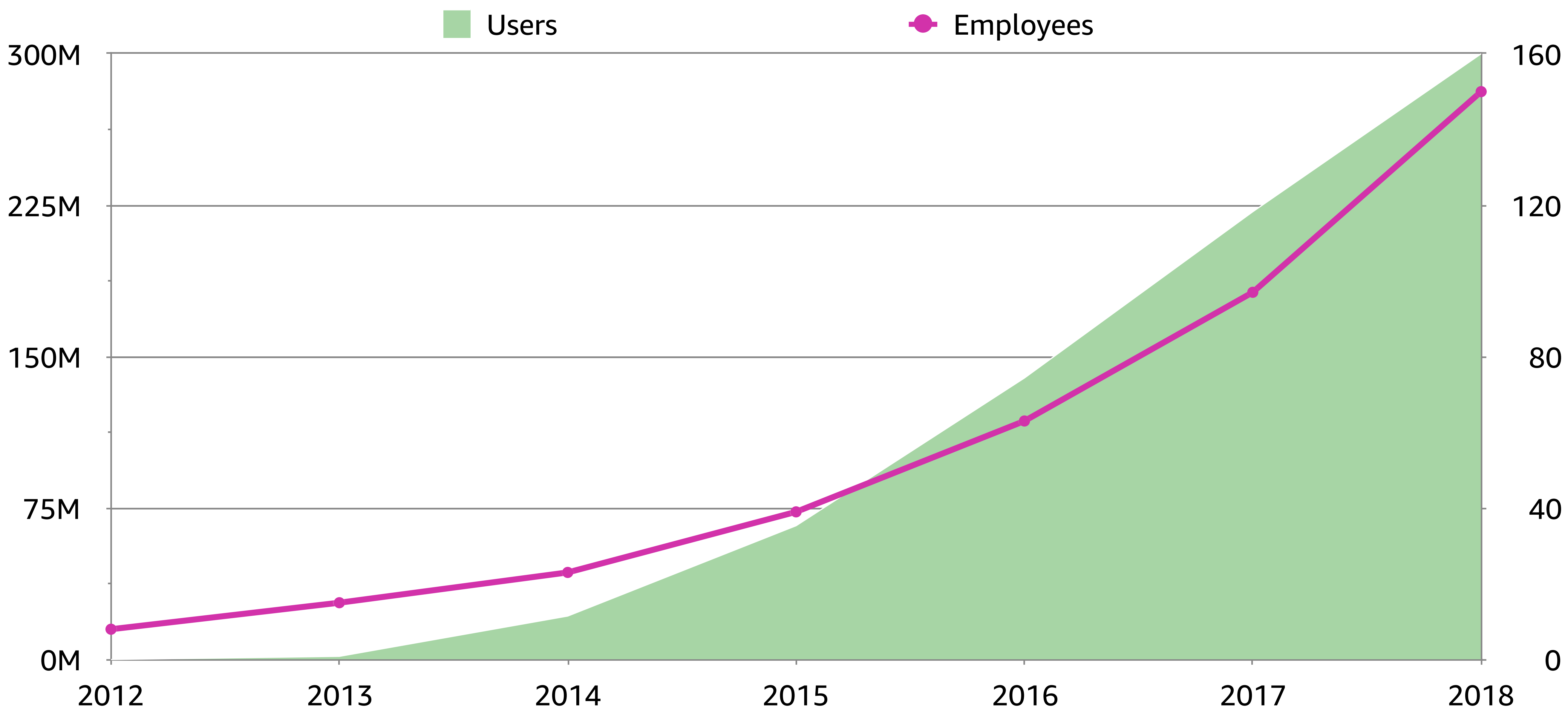
duolingo

300M+ users worldwide

180 employees

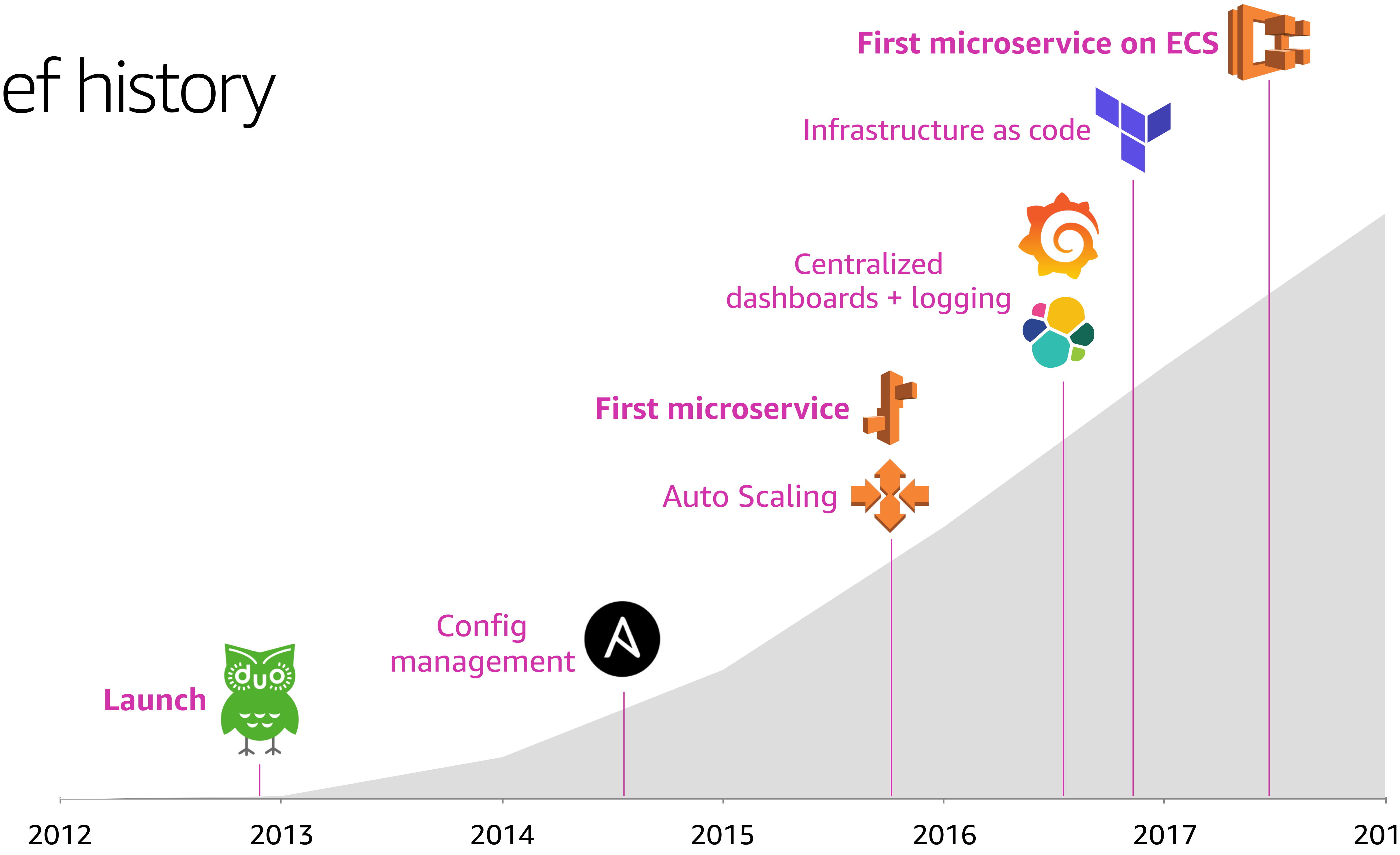
duolingo

Duolingo growth



duolingo

A brief history



duolingo

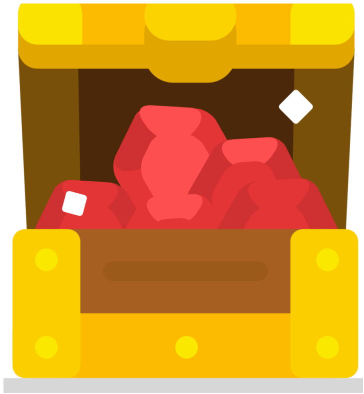
Why move to microservices?

Scalability

Velocity

Flexibility

Reliability

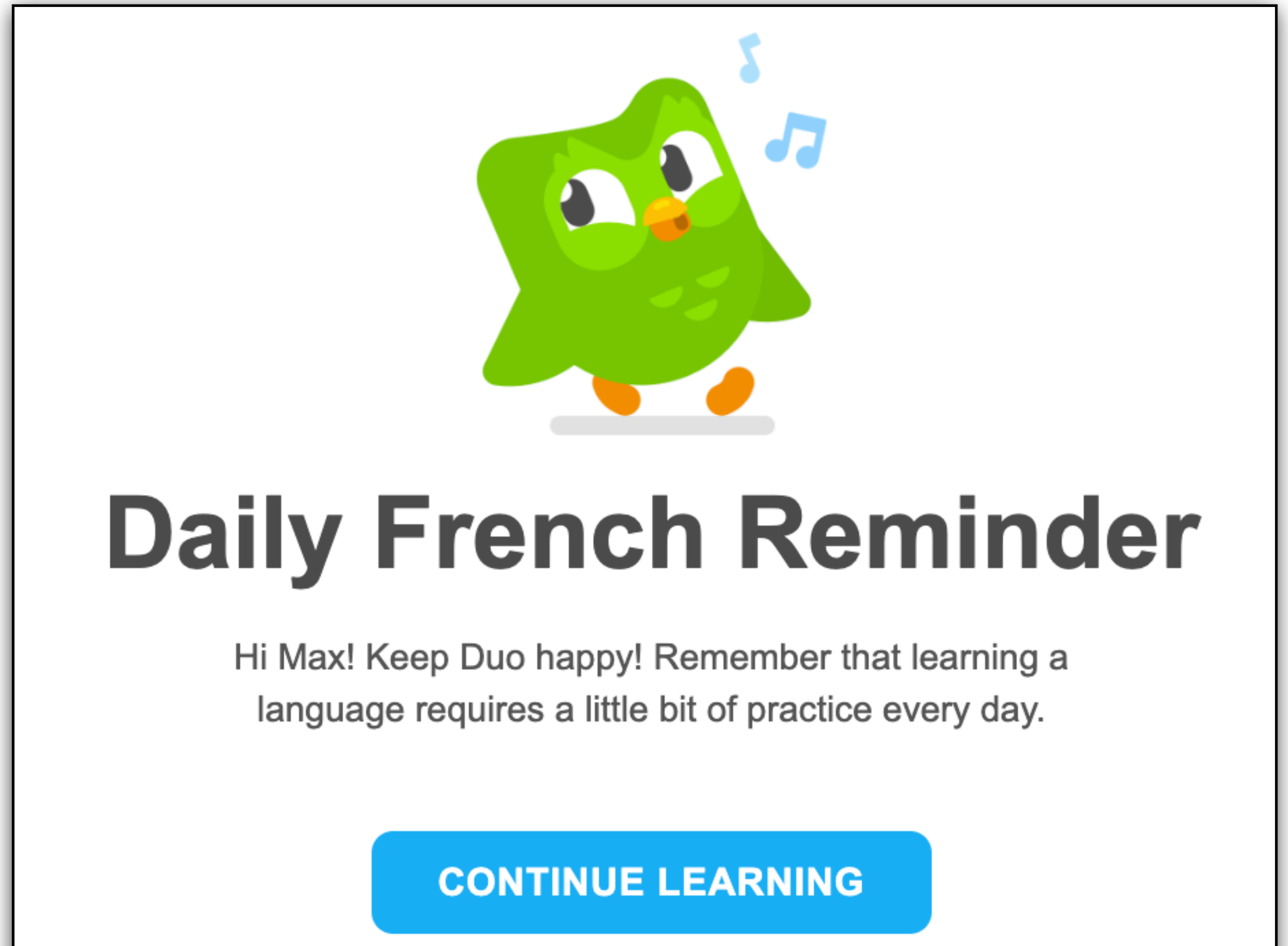


Cost savings

duolingo

How do you decide what to carve out of your monolith first?

- Start with a small, but impactful feature
- Move up in size, complexity, and risk
- Consider dependencies



duolingo

Monolith

System availability

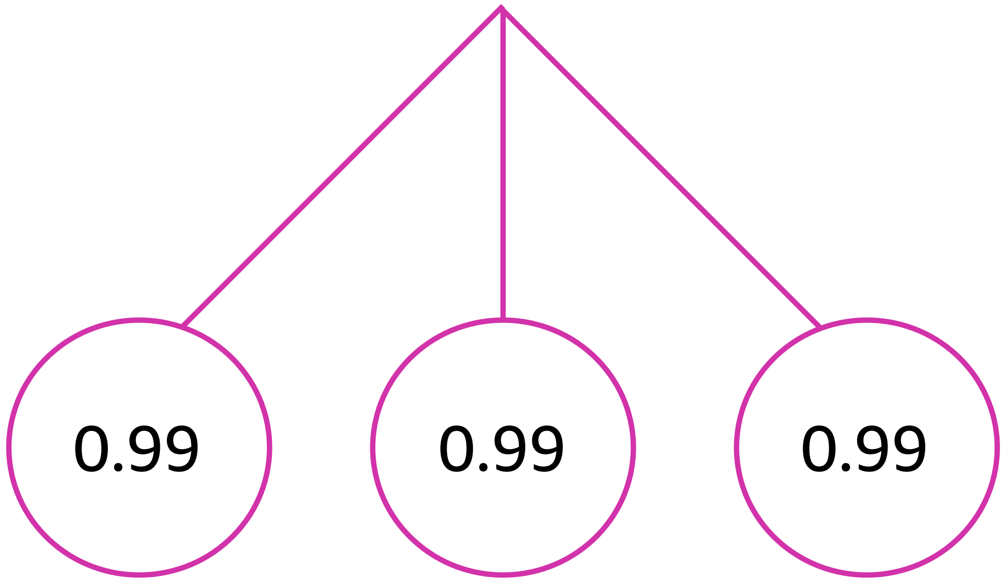
0.99

Chained microservices

0.99 * 0.99 * 0.99 = 0.97

duolingo

Independent microservices



$$1 - (1 - 0.99)^3 = 0.999999$$

Why use Docker for microservices?

- Standardizes the build process and encapsulates dependencies
- Local development environment similar to production
- Quick deployments and rollbacks
- Flexible resource allocation

duolingo

Simplifying local development setup (old way)

1. Clone this repository.
2. Set up and activate a virtualenv and install requirements using pip install -r requirements.txt.
3. Download and install Postgres: brew install postgresql
4. Run Postgres locally: postgres -D /usr/local/var/postgres
5. Download pgAdmin3 (not totally necessary, but will make life easier).
6. Using pgAdmin3, create a new login role under your local server with name "admin" and password "somepassword".
7. Create a DB called “db".
8. Run the migration script in the repo using python manage.py db upgrade.
9. Check that your DB is now populated with tables.
10. Set up and run memcached: brew install memcached
11. Set up and run redis: brew install redis-server
12. Set up and run elasticsearch: brew install elasticsearch
13. Finally, try to run the server using python application.py. You can test if it’s working by going here

duolingo

Simplifying local development setup (new way)

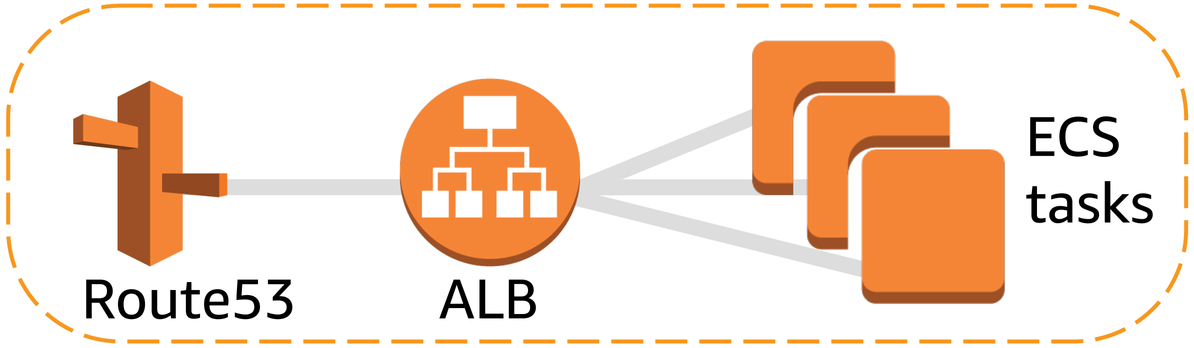
```
$ docker-compose build

$ docker-compose up
```

Why use Docker with ECS?

- Task Auto Scaling
- CloudWatch metrics
- Task-level IAM
- Dynamic ALB targets
- Manageability

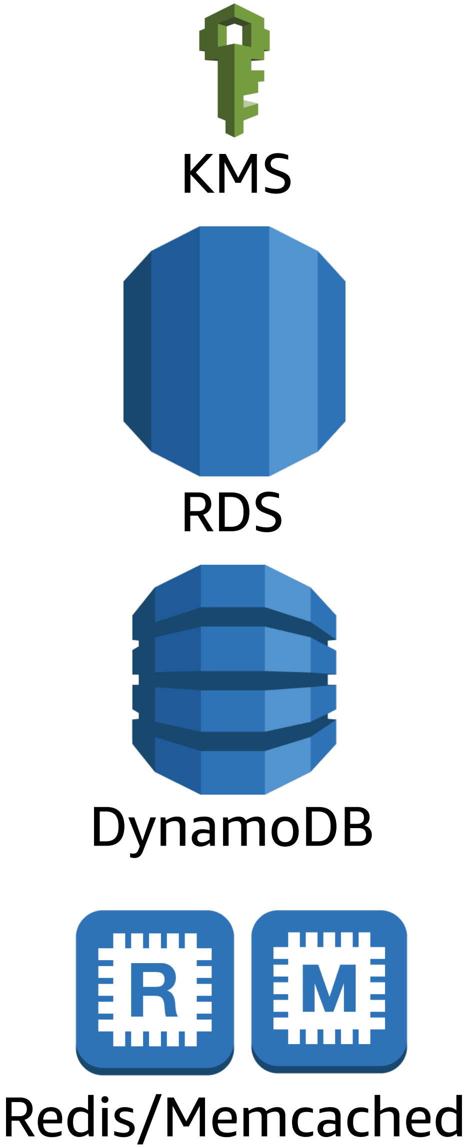
Microservice abstractions at Duolingo

Web service (internal or external)



Worker service (daemon or cron)

Data stores



Monitoring

duolingo

Microservice definition in Terraform

```
module “duolingo-api" {
  source        = “repo/terraform//modules/ecs_web_service"
  environment   = “prod"
  product       = “duolingo"
  service       = “api"
  owner         = “Max Blaze"
  min_count     = 2
  max_count     = 4
  cpu           = 512
  memory        = 256
  ecs_cluster   = "prod"
  internal      = "true"
  container_port = 5000
  version       = "${var.version}"
}
```

Billing tags

Auto Scaling

Resources

duolingo

Aurora database cluster definition in Terraform

```
module “duolingo-api–db“ {
  source                = “repo/terraform//modules/ecs_web_service"
  product               = “duolingo”
  service               = “api”
  subservice            = “db"
  owner                 = “Max Blaze”
  cluster_identifier    = “duolingo-api-db-cluster"
  identifier            = “duolingo-api"
  engine                = "aurora-postgresql"
  name                  = "duolingo"
  password              = "${data.aws_kms_secret.duolingo_api_db.duolingo_api_db}"
  instance_class        = "db.r4.large"
  num_cluster_instances = 2
}
```

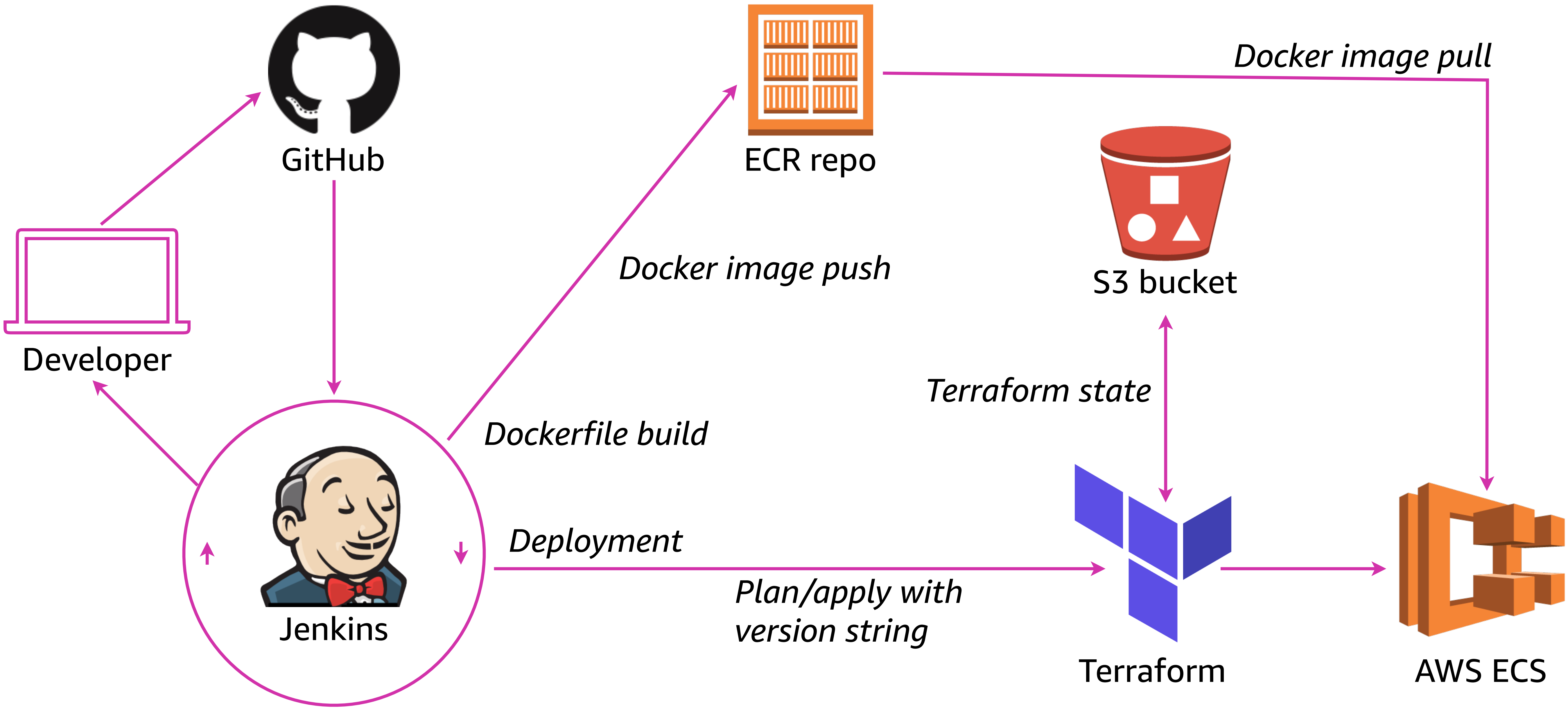
Billing tags

DB engine

Instance type

duolingo

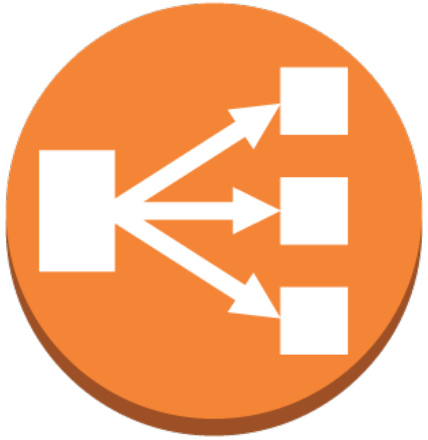
Continuous integration and deployment



duolingo

Load balancing

- ALBs and CLBs operate at different network layers
- ALBs are more strict when handling malformed requests
- ALBs default to HTTP/2
    - Headers are *always* passed as lowercase
- There are differences in CloudWatch metrics



ALB $\neq$ CLB

Task-level IAM role permissions

- Apply permissions at the service level
- Do not share permissions across microservices
- Needs to be supported by the AWS client library

Standardizing microservices

- Develop a common naming scheme for repos and services
- Autogenerate as much of the initial service as possible
- Move core functionality to shared base libraries
- Provide standard alarms and dashboards
- Periodically review microservices for consistency and quality

duolingo

Monitoring microservices

Web service dashboard

- Local time and UTC
- Healthy, unhealthy, and running tasks
- Latency average and percentiles
- Number of requests
- CPU and memory utilization (min/avg/max)
- Service errors by AZ
- ALB errors by AZ

duolingo

Monitoring microservices

Worker service dashboard

- Local time and UTC
- Running tasks
- CPU and memory utilization (min/avg/max)
- Visible messages
- Deleted messages

duolingo

Monitoring microservices

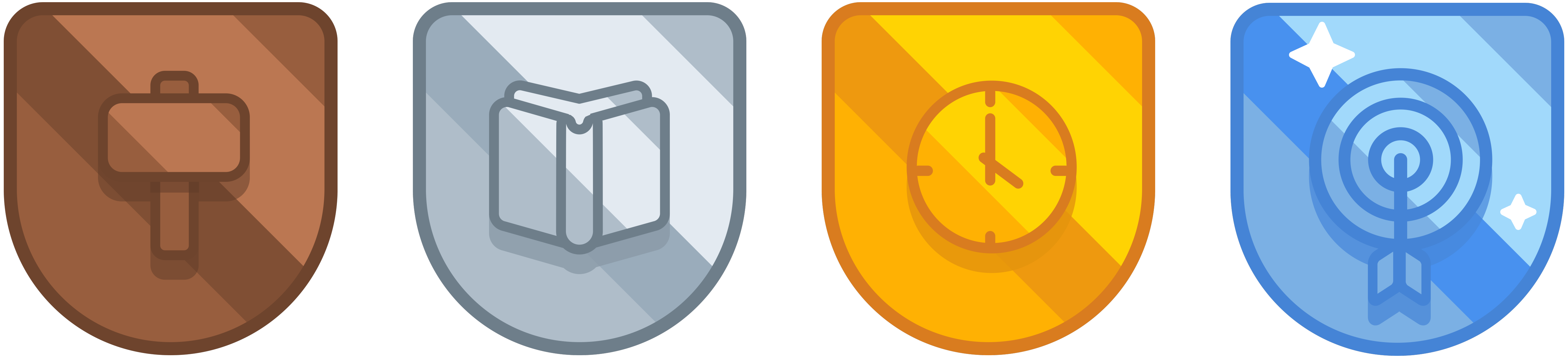
PagerDuty integration

- Schedules and rotations are defined in Terraform
- Emergency alarms page (high latency)
- Warning alarms go to e-mail (low memory)
- Include links to playbooks
- All pages are also visible in Slack

duolingo

Grading microservices

Architecture | Documentation | Processes | Tests



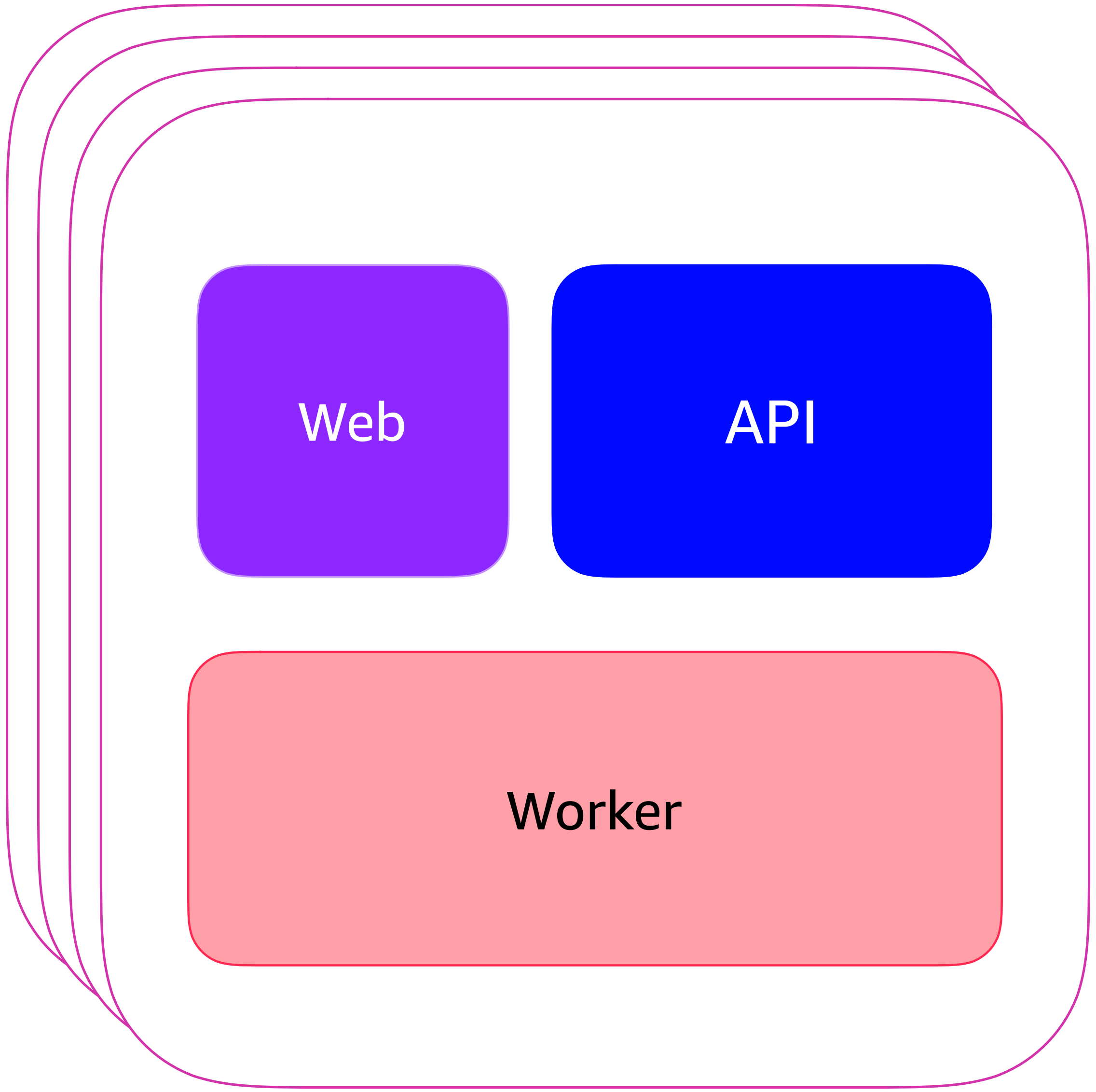
Grading microservices

Documentation

Item	Status
Is there a README file?	✅
Does the README file specify an owner?	✅
Is the documentation sufficient to install and run the microservice locally?	✔
Does the README state its dependencies on other microservices?	✔
Does the README state its clients?	✔
Is the API documented?	✔
Is the architecture explained? (e.g. architecture diagram)	✔
Are operational processes explained? (e.g. deployment, DB schema changes, data loaders)	✔

Cost reduction options

- Cluster
  - Instance type
  - Pricing options
  - Auto Scale
  - Add/remove AZs
- Task
  - Resource allocation
  - Auto Scale



duolingo

Cluster starting point

c3.2xlarge

Reserved Instances

On-Demand

High-CPU Instance Generations

	Speed	$/hour	Disk
c3.large	-	0.105	SSD
c4.large	+20% of c3	0.100	None (EBS-only)
c5d.large	+25% of c4	0.096	NVMe

c5 is 50% faster than c3!

duolingo

Moving to a new EC2 generation

Latest instances are generally *faster* and *cheaper but…*

- "cpu" units in ECS *will not* be equivalent
- Auto Scaling may not work properly between generations

c5.large		c4.large		c3.large
cpu = 1024	>	cpu = 1024	>	cpu = 1024

(1 vCPU core = 1024 units)

duolingo

Fixed number of instances

c5d.2xlarge

Reserved Instances

On-Demand

Auto Scaling

c5d.large…c5d.18xlarge
m5d.large...m5d.24xlarge

Reserved Instances

On-Demand

Spot

duolingo

Fixed number of instances

spotinst

c5d.2xlarge

Reserved Instances

On-Demand

c5d.large…c5d.18xlarge
m5d.large...m5d.24xlarge

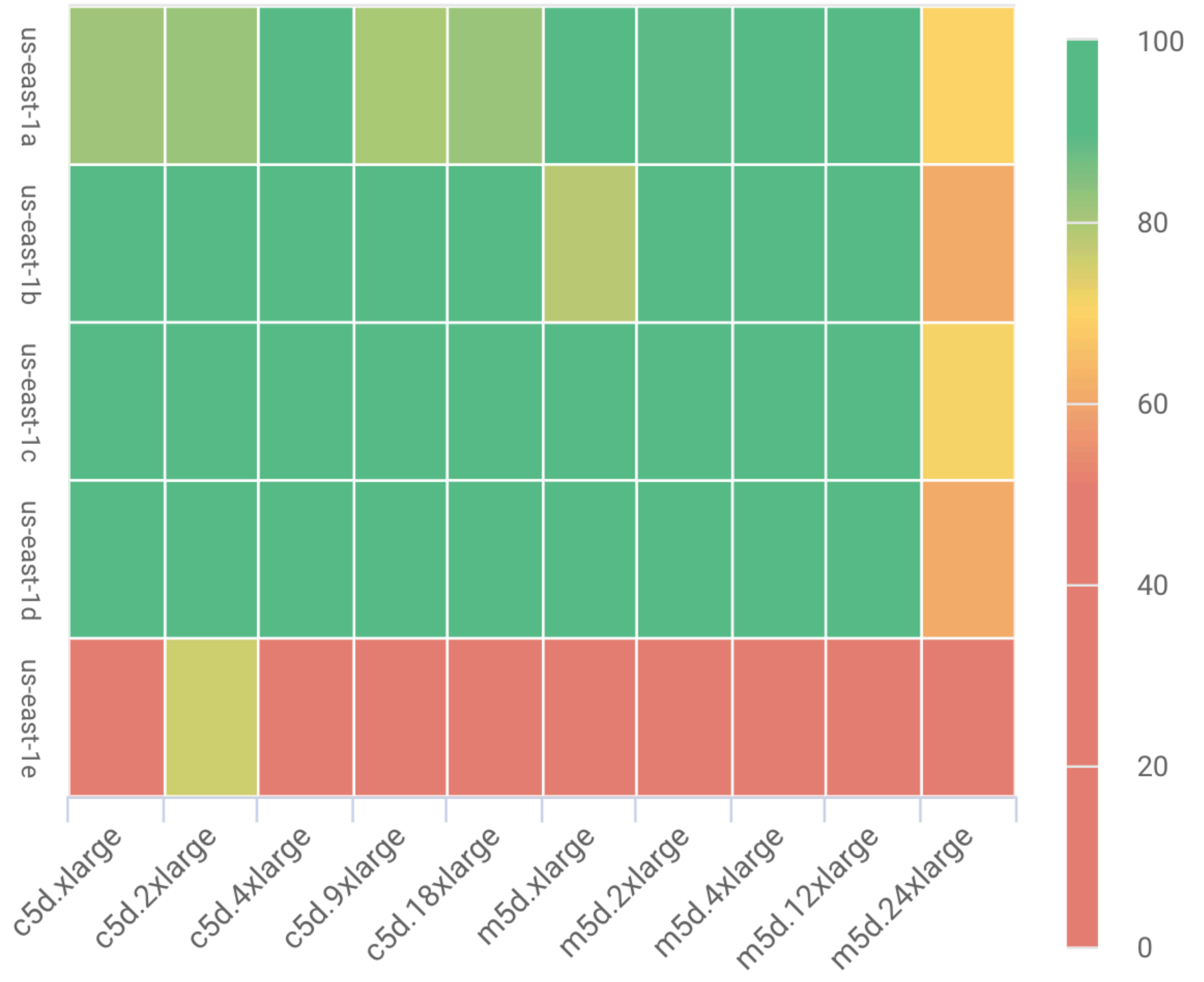
Reserved Instances

On-Demand

Spot

Spotinst cluster features

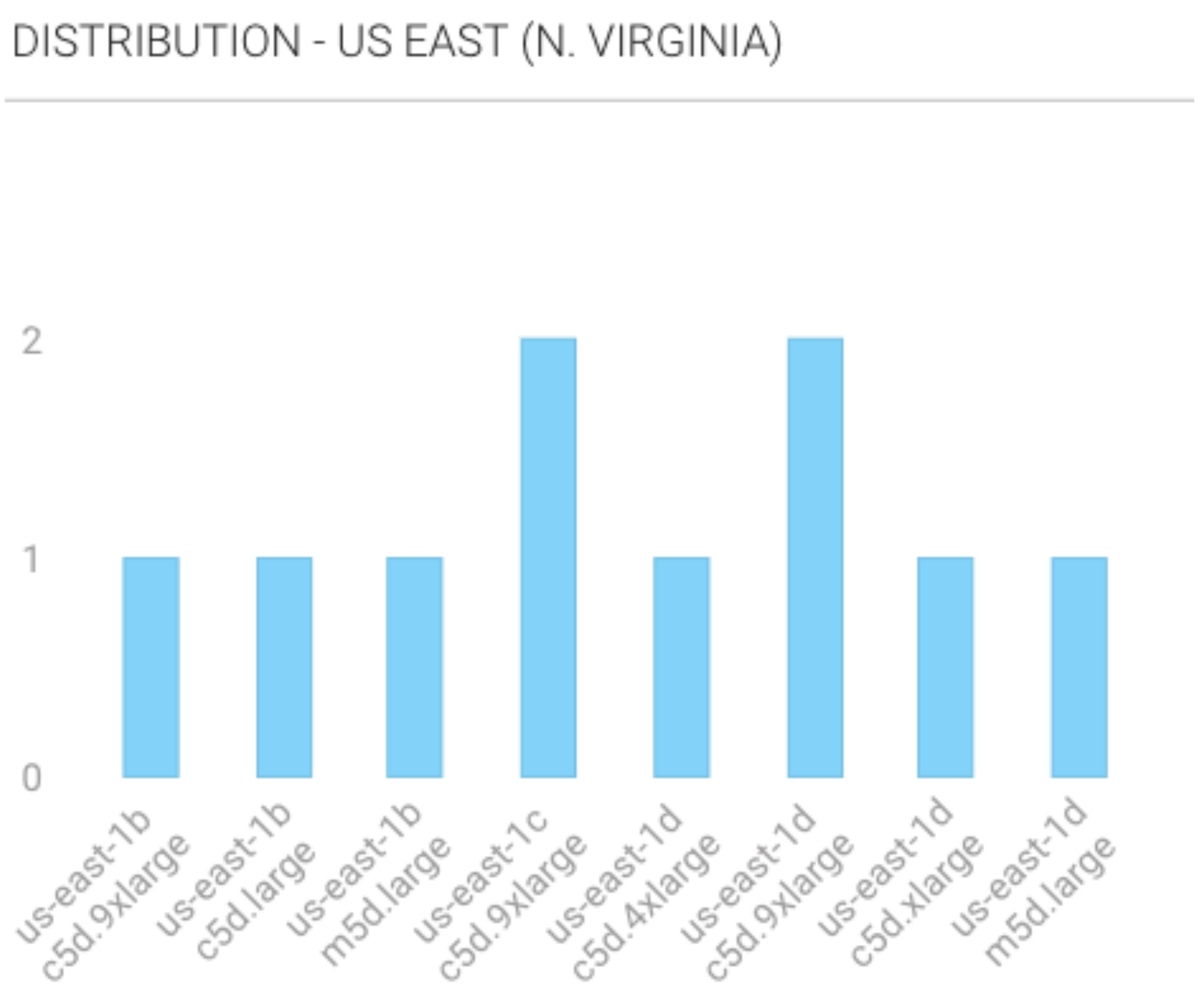
- Mixes families + sizes
- Uses RIs before spot
- 15 minute spot notice
- Fits capacity to ECS tasks
- AZ capacity heat map



duolingo

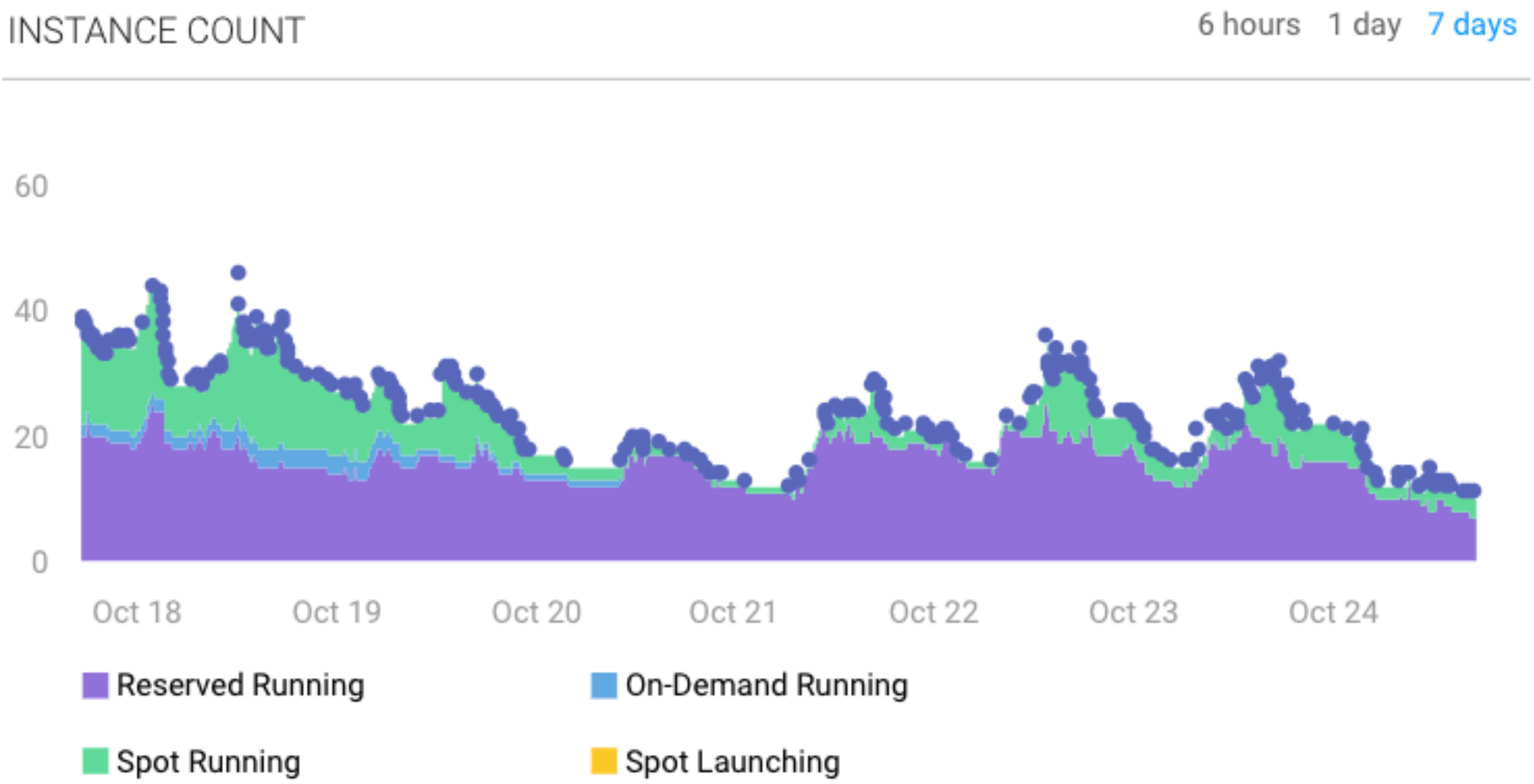
Spotinst cluster features

- Drains ECS tasks
- Cluster "headroom"
- Spreads capacity across AZs
- Bills on % of *savings*
- Terraform support



duolingo

Auto Scaling with Spotinst



duolingo

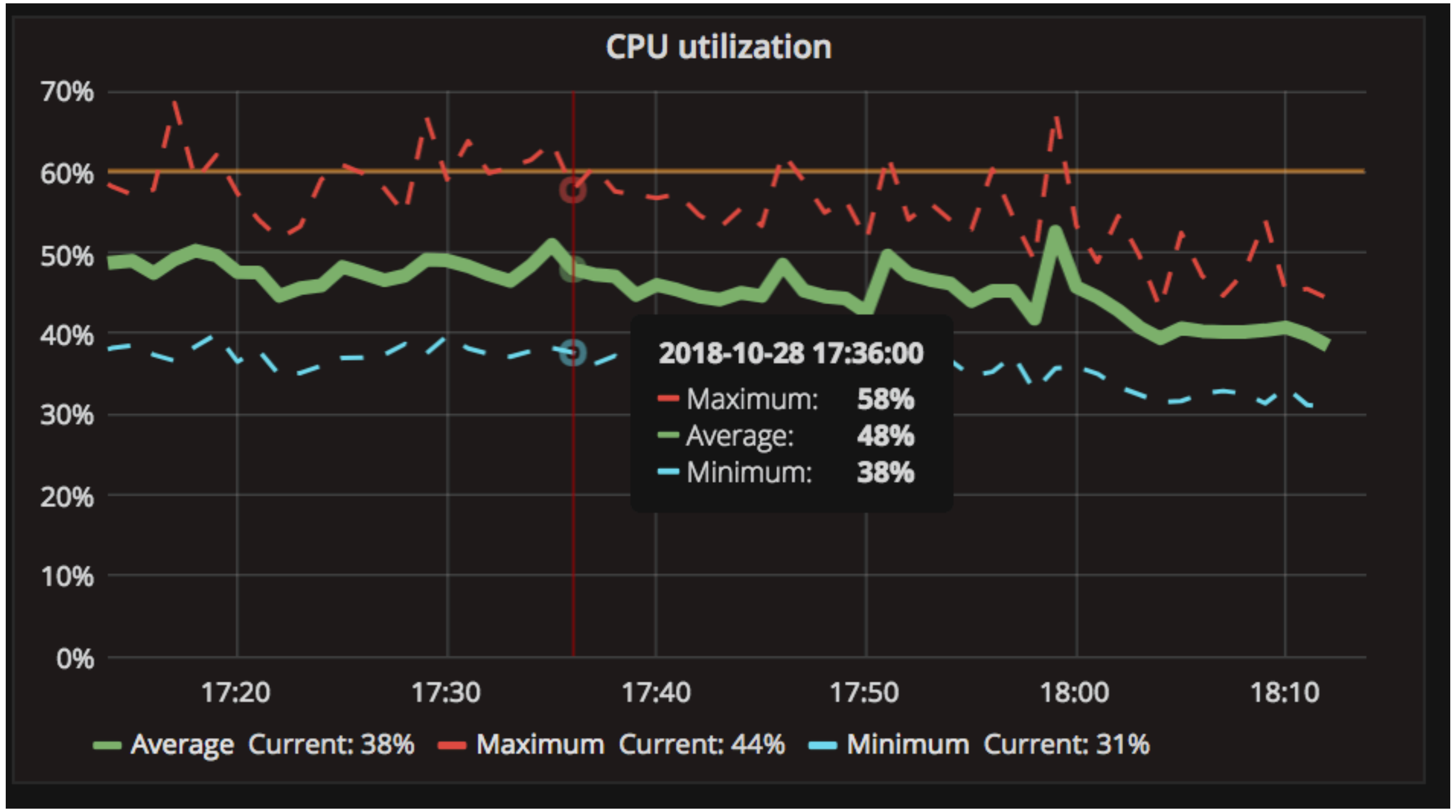
What about per-microservice costs?

- Audit CPU/memory allocations for each service
- Update Auto Scaling and/or CPU allocations as needed

Goals

60% CPU

60–80% Memory



duolingo

Adjusting allocated CPU for scaling

*allocatedCPU * currentUtilization = actualCPU*
actualCPU / desiredUtilization = Units to set

Example:
Current utilization: 40%
Desired utilization: 60%

1024 * 40% = 409.6
409.6 / 60% = 682.67

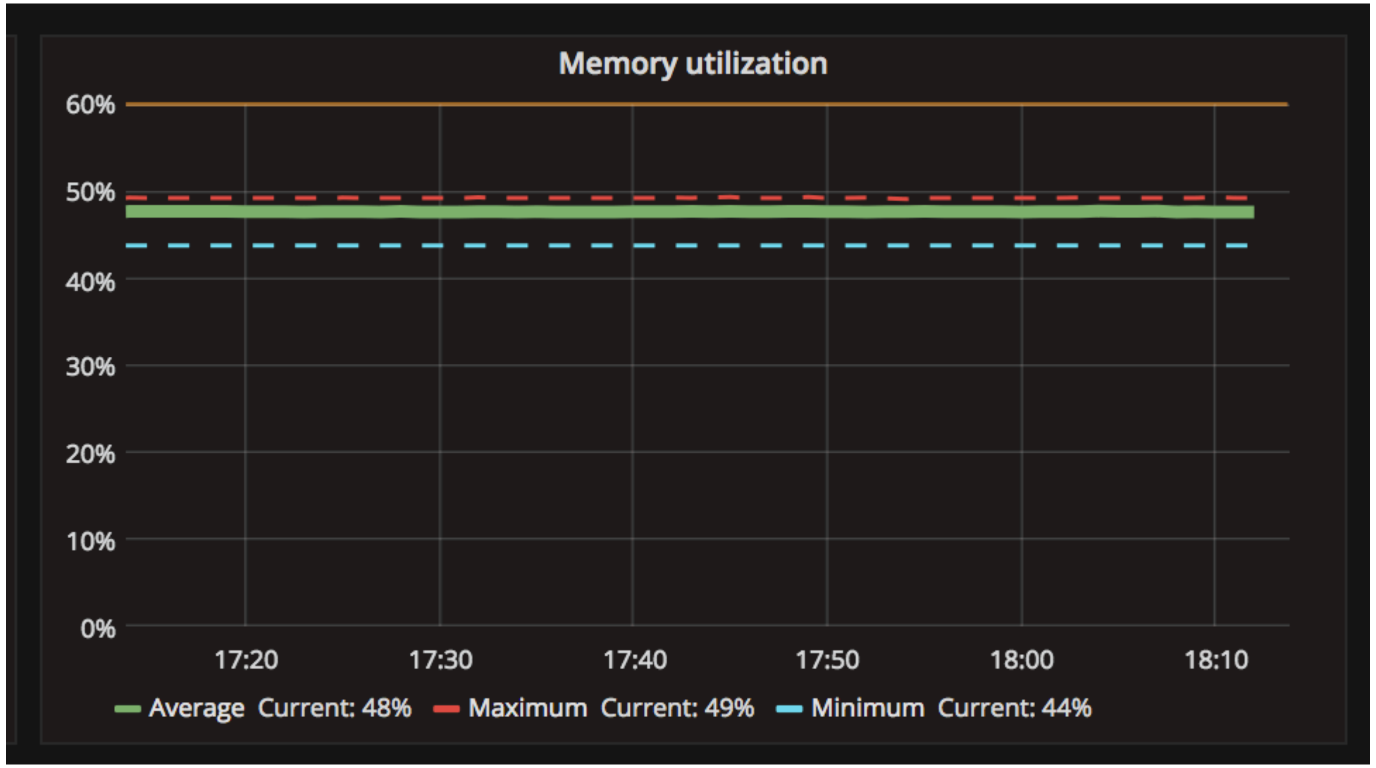
Set ECS “cpu” allocation to 683

(1 vCPU core = 1024 units)

duolingo

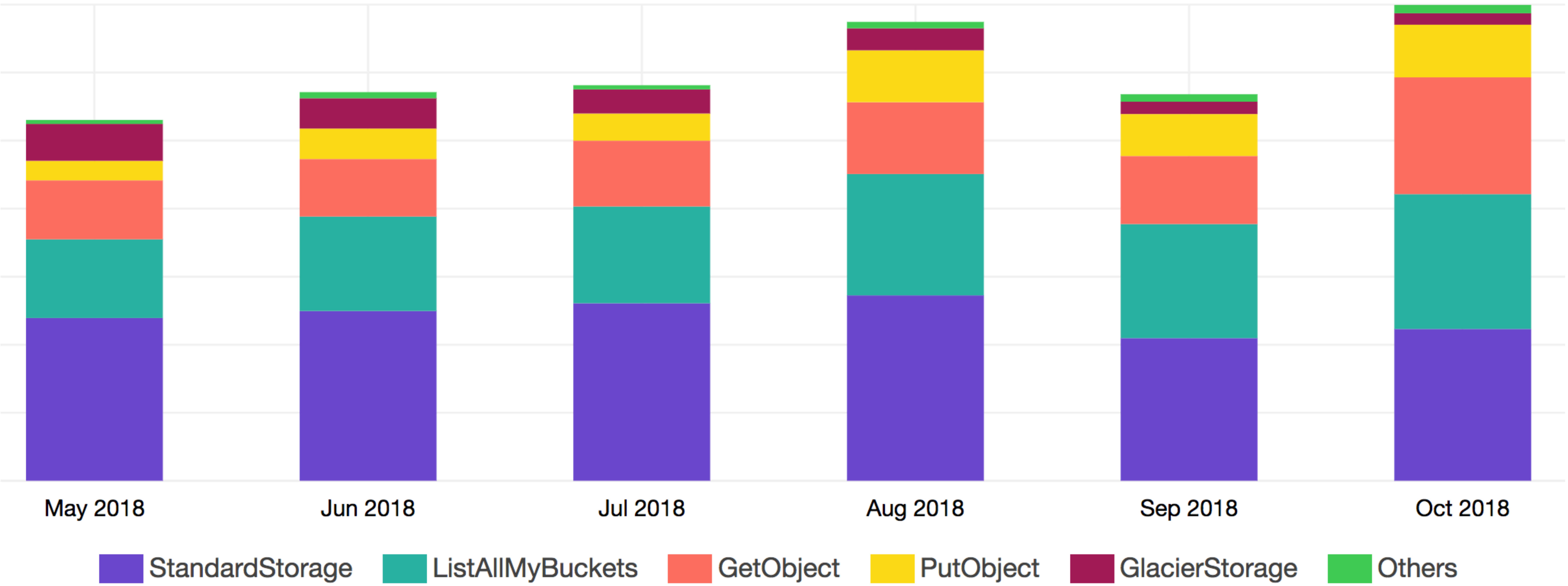
Adjusting allocated memory

- Track memory usage between deployments
- Constantly increasing memory usage points to memory leaks
- Set containers to restart if memory exceeds 100%



duolingo

API costs



ListAllMyBuckets + GetObject > 50% of S3 cost!

duolingo

Limits

"Each Amazon EC2 instance limits the number of packets that can be sent to the Amazon-provided DNS server to a maximum of 1024 packets per second per network interface. This limit cannot be increased."

s_maj
"Nitro based instance types are running fine nowadays. Just be aware that they might be not available in all AZs within region. And I think Nitro is not caching DNS requests where xen based instance were doing that."

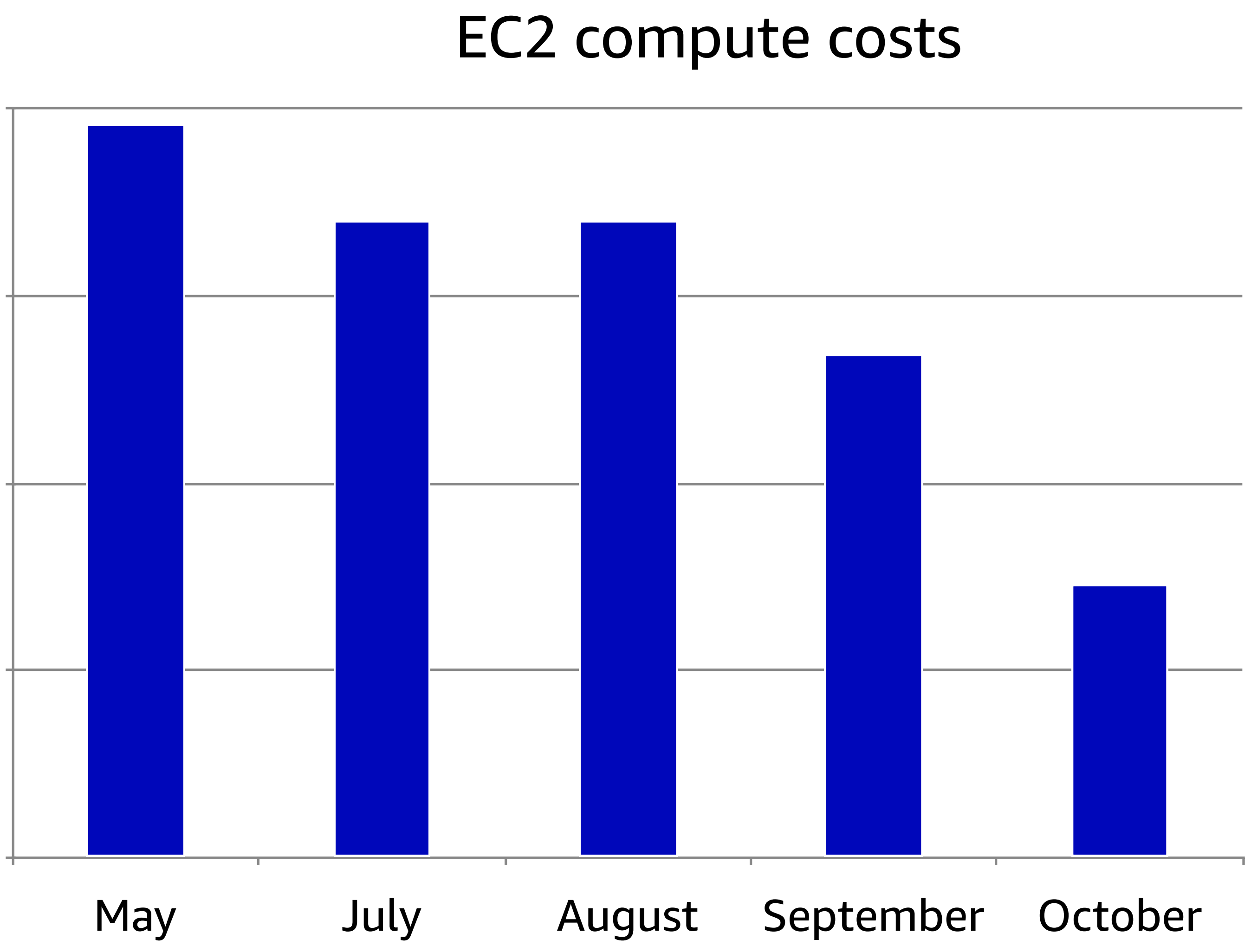
https://docs.aws.amazon.com/vpc/latest/userguide/vpc-dns.html#vpc-dns-limits
https://www.reddit.com/r/aws/comments/9bu4x4/how_are_nitro_instances_treating_everyone/

Cost savings

> 60% reduction in compute costs

> 30% reduction in costs per monthly active user (MAU)

> 25% reduction total AWS bill

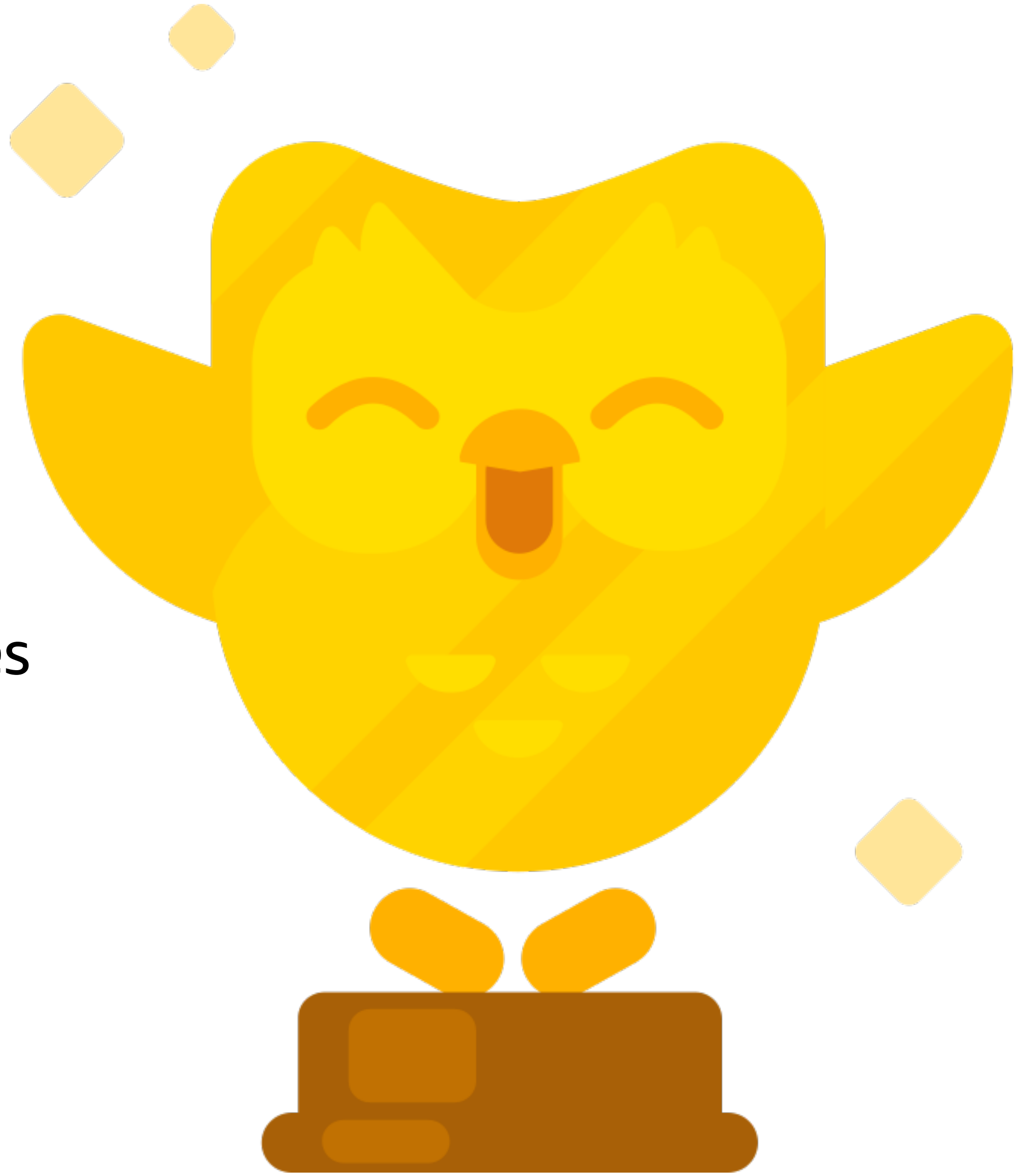


> 60% reduction from May to October

duolingo

Key results

- Scalability
  - Manage ~100 microservices
- Velocity
  - Teams deploy to their own services
- Flexibility
  - Officially support 3 different programming languages
- Reliability
  - 99.99% availability achieved after implementation
- Cost
  - 60% reduction in compute costs



duolingo

duolingo.com/careers

Resources

- Books
  - Building Microservices: Designing Fine-Grained Systems (Sam Newman)
  - Microservices in Production (Susan J. Fowler)
- References
  - ec2instances.info
  - github.com/open-guides/og-aws
- Tools and services
  - ansible.com
  - docker.com
  - elastic.io
  - github.com
  - grafana.com
  - jenkins.io
  - pagerduty.com
  - runatlantis.io
  - spotinst.com
  - terraform.io

duolingo