

# Fargate

Aurelijus Banelis



VilniusPHP 0x63  
2021-02-04



# Aurelijus Banelis

## Senior Software Engineer

**aurelijus.banelis.lt**  
**aurelijus@banelis.lt**

PGP 0x320205E7**539B6203**  
130D C446 1F1A 2E50 D6E3  
3DA8 3202 05E7 539B 6203







# Aurelijus Banelis

## Senior Software Engineer

**aurelijus.banelis.lt**  
**aurelijus@banelis.lt**

PGP 0x320205E7**539B6203**  
130D C446 1F1A 2E50 D6E3  
3DA8 3202 05E7 539B 6203





# Development with infrastructure





**What**

**AWS Fargate intro**

**Why**

**Story about migrating to  
Fargate**

**How**

**Short Demo**





**What**

**AWS Fargate intro**

**Why**

**Story about migrating to  
Fargate**

**How**

**Short Demo**

# AWS Fargate

Serverless compute for containers

Get started with AWS Fargate

# Serverless compute for containers

## FEATURED LAUNCH

### AWS Proton

Define, manage, and update your infrastructure so your developers can focus on writing great code.

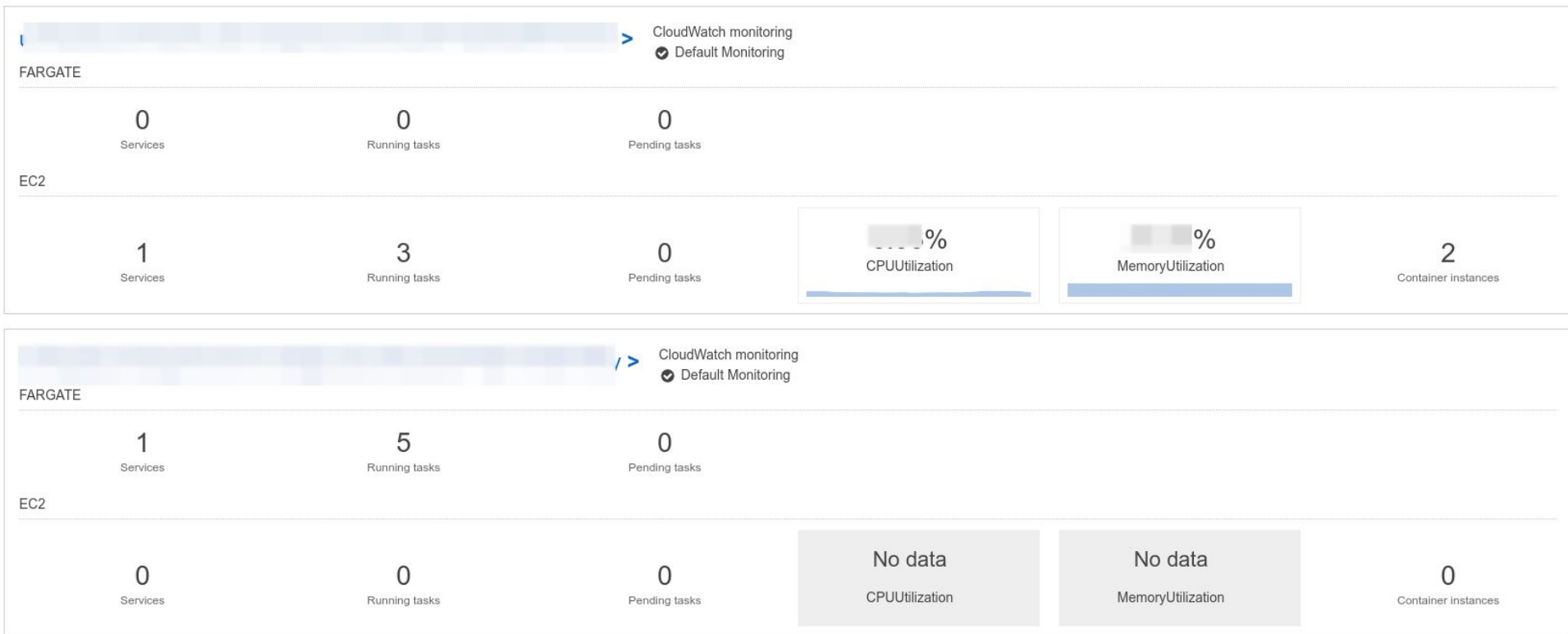
[Learn more >](#)

AWS Fargate is a serverless compute engine for containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS). Fargate makes it easy for you to focus on building your applications. Fargate removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design.

Fargate allocates the right amount of compute, eliminating the need to choose instances and scale cluster capacity. You only pay for the resources required to run your containers, so there is no over-provisioning and paying for additional servers. Fargate runs each task or pod in its own kernel providing the tasks and pods their own isolated compute environment. This enables your application to have workload isolation and improved security by design. This is why customers such as Vanguard, Accenture, Foursquare, and Ancestry have chosen to run their mission critical applications on Fargate.

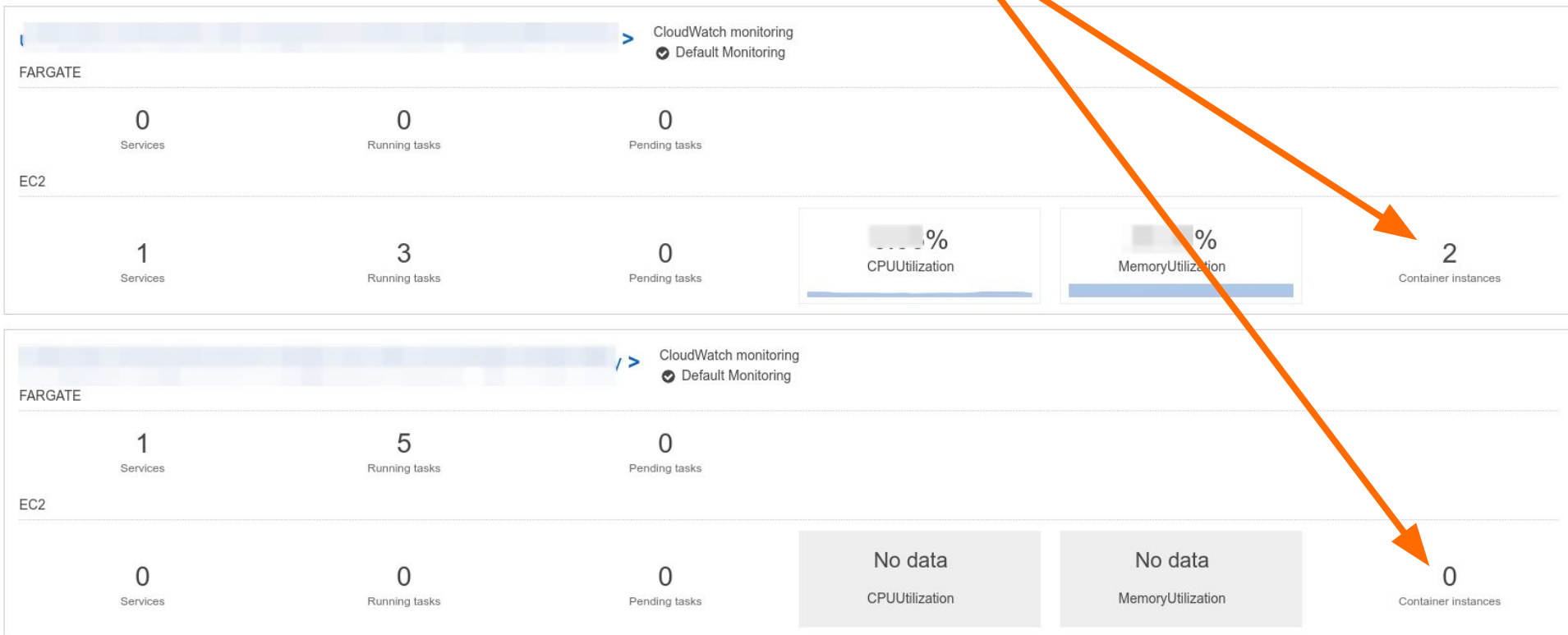
<https://aws.amazon.com/fargate>

# Serverless compute

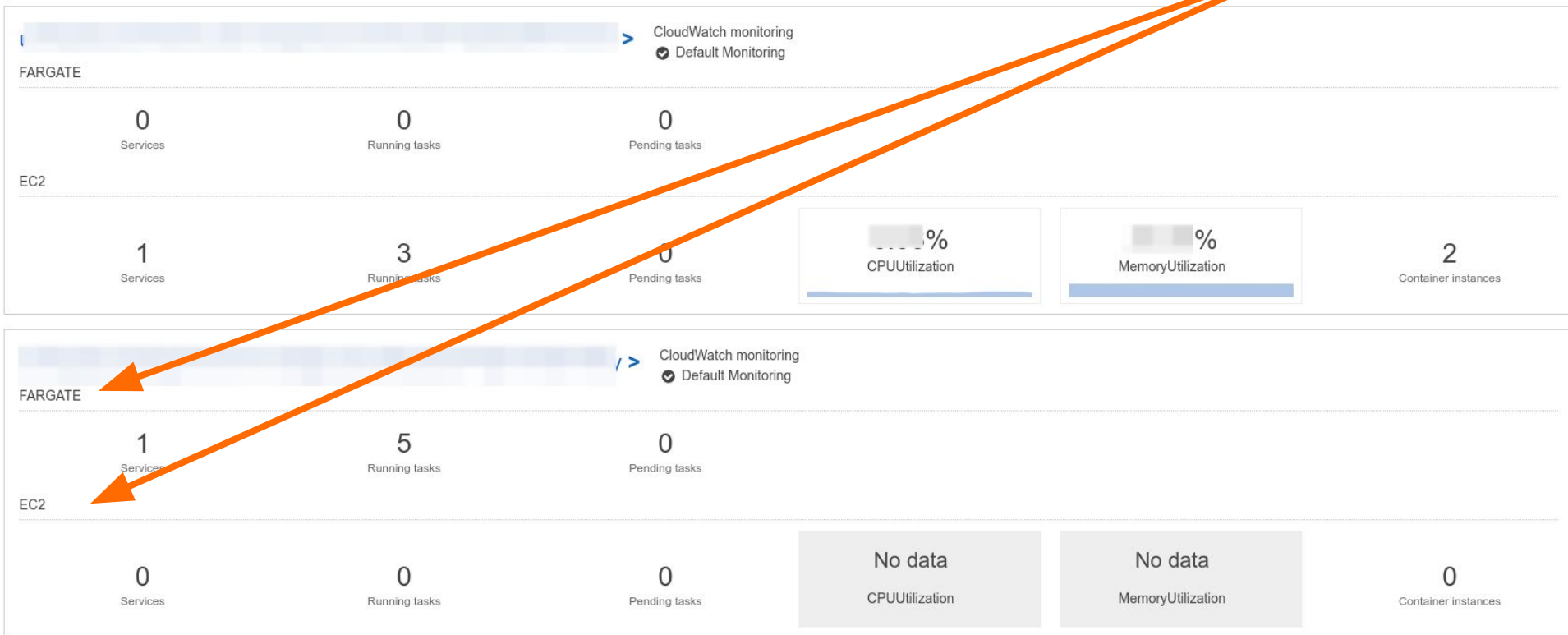




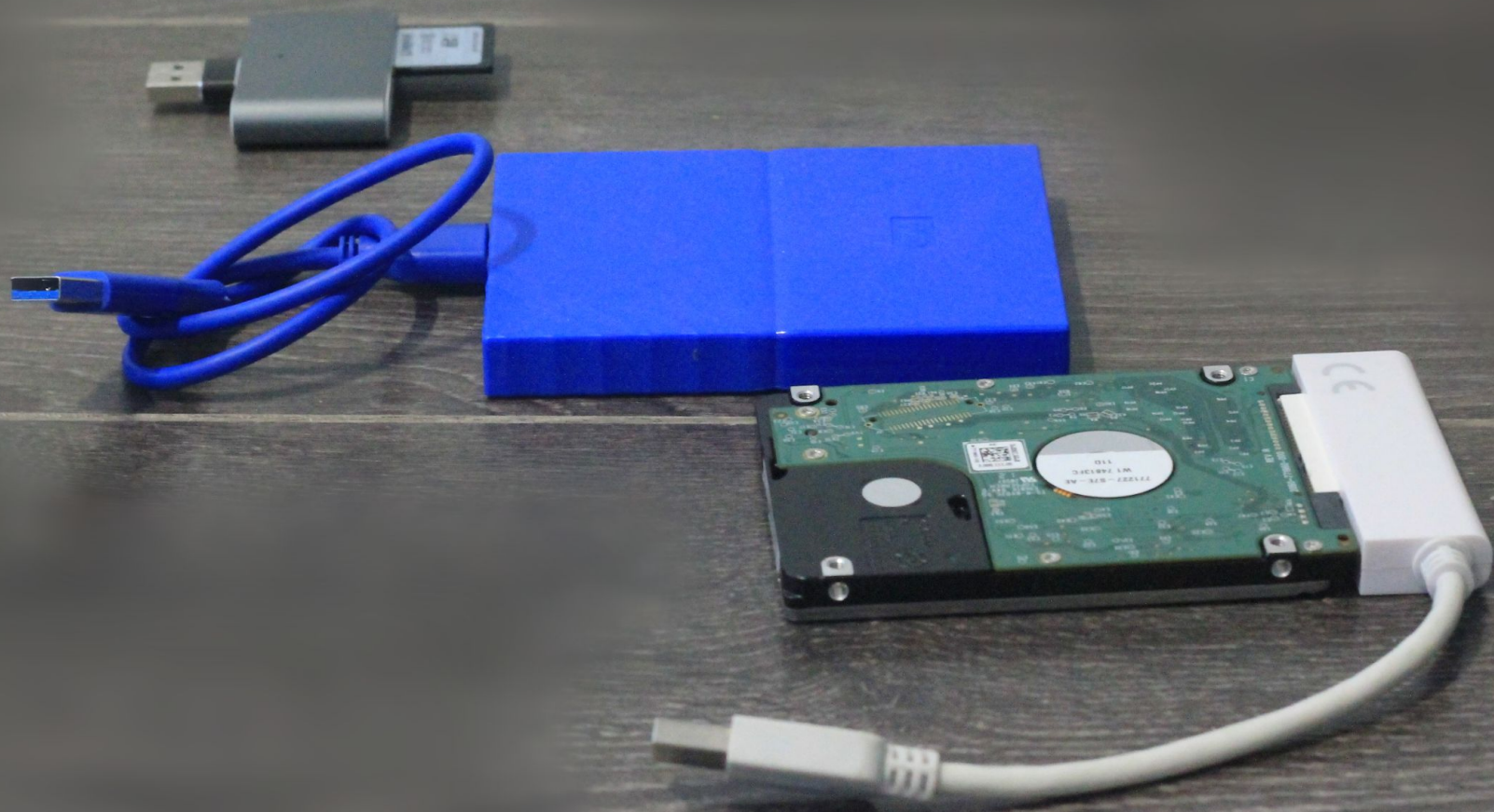
# Serverless compute



# Serverless compute









Lambda



Fargate

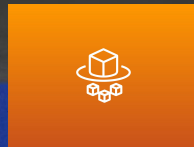


EC2





Lambda



Fargate



EC2



# For containers

```
FROM php:7.3.3-fpm

LABEL maintainer "Karelilijn Banella <karelilijn@banella.it>"

WORKDIR /tmp

# Get composer
RUN apt-get update
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
RUN php -r "if (hash_file('sha384', 'composer-setup.php') === '4dc595b24a9b147d40f5c950437e54998b69172e4608551369684f000884000') { echo 'Valid composer-setup.php'; } else { echo 'Invalid composer-setup.php'; exit 1; }"
RUN php composer-setup.php
RUN mv composer-setup.php /usr/bin/composer
RUN ln -s /usr/bin/composer.phar /usr/bin/composer

# Install dependencies and PHP extensions
RUN apt-get update \
&& apt-get install -y libzip-dev bash-completion gnu nano libicu-dev g++ libpq-dev libcurl4-gnutls-dev git wget zip unzip \
&& docker-php-ext-install -j$(nproc) zip mysql pdo_mysql calendar onig pcntl sockets upgrade \
&& rm -rf /usr/lib/apt/README"

# xdebug helpers (do not use this in real production)
ADD enable_xdebug.sh /enable_xdebug.sh
ADD disable_xdebug.sh /disable_xdebug.sh
RUN perl -i install_xdebug-2.7.0-88.phar \
  && chmod +x /enable_xdebug.sh && \
  && chmod +x /disable_xdebug.sh && \
  && touch /usr/local/etc/php/conf.d/custom-xdebug.ini && \
  && chmod 777 /usr/local/etc/php/conf.d/custom-xdebug.ini

# Add Synergy extension
RUN wget https://get.synergy.com/cli/installer -O - | bash
RUN mv -f synergy/bin/synergy /usr/local/bin/synergy

# Not root user
RUN useradd -s /bin/bash -m -d /home/php -s /bin/bash php
USER php
ENV HOME /home/php

# Configure git, so there would not be errors running "synergy new wj.project"
RUN git config --global user.email "Docker.Fake.user@example.com" && git config --global user.name "Docker.Fake.user"

# xdebug configuration
ENV PHP_IDE_CONFIG="serverName=localhost"

WORKDIR /code
VOLUME /code
```





# For containers

```
FROM php:7.3.3-fpm

LABEL maintainer "Burt"

WORKDIR /app

# Get composer
RUN apt-get update
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
RUN php composer-setup.php
RUN php -r "unlink('composer-setup.php');"
RUN ln -s /php/composer.phar /usr/local/bin/composer

# Install dependencies and PHP extensions
RUN apt-get update
RUN apt-get install \
    libicu-dev git wget unzip \
    libzip-dev git wget unzip \
    libzip-dev \
    # database credentials
    DB_USER=root
    DB_PASS=pass # this is the secret password

# xdebug helpers
ADD enable_xdebug.sh
RUN chmod +x enable_xdebug.sh
RUN ./enable_xdebug.sh
RUN apt-get install \
    libicu-dev git wget unzip \
    libzip-dev \
    # database credentials
    DB_USER=root
    DB_PASS=pass # this is the secret password

# Add Symfony extension
RUN wget https://get.symfony.com
RUN mv -f symfony.com/symfony.com

# Not root user
RUN useradd -s /bin/bash user
USER user
WORKDIR /home/user

# Configure git, so there is no global config
RUN git config --global user.name "Docker Fake user"

# xdebug configuration
ENV PHP_IDE_CONFIG="serverName=localhost"

WORKDIR /code
VOLUME /code
```

Code

```
1 <?php
2
3 use App\Kernel;
```

Configurations

```
1 # database credentials
2 DB_USER=root
3 DB_PASS=pass # this is the secret password
```

PHP 8 ChangeLog

[8.0](#)

Version 8.0.1

Runtime and extensions

# For containers

The image shows a Dockerfile for a PHP 8.0 container. It includes instructions for setting the base image, installing dependencies, and configuring the environment. Three sections are highlighted with callouts:

- Code:** A snippet of PHP code showing the start of a Laravel application kernel.
- Configurations:** A snippet of environment variables for database credentials.
- Runtime and extensions:** A snippet of commands to install PHP extensions and the Symfony extension.

```
FROM php:7.3.3-fpm

LABEL maintainer "Bartek"

WORKDIR /app

# Get composer
RUN apt-get update
RUN apt-get install -y curl
RUN curl -sS https://getcomposer.org/installer | php
RUN mv composer.phar /usr/bin/composer

# Install dependencies and PHP extensions
RUN apt-get update
RUN apt-get install -y git unzip curl unzip \
    docker-php-ext-install pdo_mysql

# xdebug helpers
ADD enable_xdebug.sh /
RUN chmod +x enable_xdebug.sh
RUN ./enable_xdebug.sh

# Add Symfony extension
RUN wget https://get.symfony.com
RUN mv -f symfony.com/symfony.com

# Not root user
RUN useradd -s /bin/bash -m -u 1000 php
USER php
ENV HOME /home/php

# Configure git, so there is no global config
RUN git config --global user.name "Docker Fake user"

# xdebug configuration
ENV PHP_IDE_CONFIG="serverName=localhost"

WORKDIR /code
VOLUME /code
```

**Code**

```
1 <?php
2
3 use App\Kernel;
```

**Configurations**

```
1 # database credentials
2 DB_USER=root
3 DB_PASS=pass # this is the secret password
```

**PHP 8 ChangeLog**

[8.0](#)

**Version 8.0.1**

**Runtime and extensions**





# For containers



## Azure VM



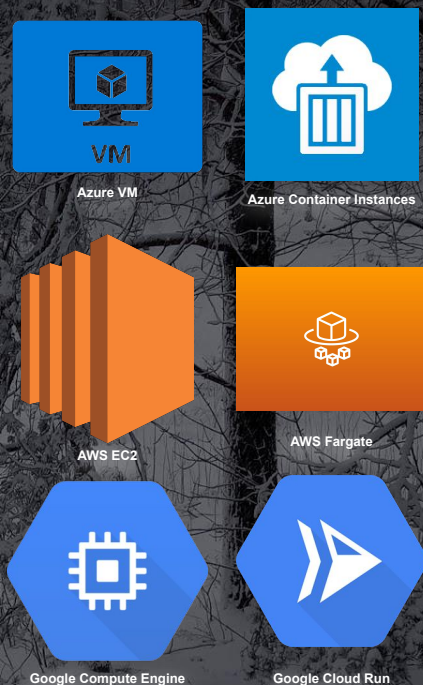
AWS EC2



Google Compute Engine



# For containers





# Serverless compute for containers

```
FROM php:7.3.3-fpm

LABEL maintainer "Hurikid"

WORKDIR /tmp

# Not composer
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
RUN php -r "if (hash_file('sha256', 'composer-setup.php') === '3fa8c0fb6a00373aa713411d8987267478573393638574260276136609e68d762') { echo 'installer hash is OK'; } else { echo 'installer hash is NOT OK. Aborting.'; exit 1; }" && rm -rf composer-setup.php
RUN php composer-setup.php
RUN mv composer.phar /usr/bin/composer
RUN ln -s /usr/bin/composer /usr/bin/composer

# Install dependencies and PHP extensions
RUN apt-get update
RUN apt-get install \
    ca-certificates curl \
    gnupg lsb-release \
    docker-php-ext-install pdo_mysql unzip \
    git -y

# xdebug helpers
RUN echo 'xdebug.enabled=1' >> /usr/local/etc/php/conf.d/xdebug.ini
RUN echo 'xdebug.remote_enable=1' >> /usr/local/etc/php/conf.d/xdebug.ini
RUN echo 'xdebug.remote_host=127.0.0.1' >> /usr/local/etc/php/conf.d/xdebug.ini
RUN echo 'xdebug.remote_port=8888' >> /usr/local/etc/php/conf.d/xdebug.ini
RUN echo 'xdebug.ideurl=php://dev' >> /usr/local/etc/php/conf.d/xdebug.ini
RUN echo 'xdebug.mode=debug' >> /usr/local/etc/php/conf.d/xdebug.ini
RUN echo 'xdebug.start_with_request=yes' >> /usr/local/etc/php/conf.d/xdebug.ini

# Add Symfony extension
RUN wget https://get.symfony.com/composer -O - | bash
RUN mv -f /usr/local/bin/symfony /usr/bin/symfony

# Not root user
RUN useradd -s /bin/bash -m -u 1000 symfony
RUN chown -R symfony:symfony /tmp
RUN git config --global user.name "Docker Fake user"

# xdebug configuration
ENV PHP_IDE_CONFIG="serverName=docker"

WORKDIR /code
VOLUME /code
```

Code

```
1 <?php
2
3 use App\Kernel;
```

Configurations

```
1 # database credentials
2 DB_USER=root
3 DB_PASS=pass # this is the secret password
```

PHP 8 ChangeLog

8.0

Version 8.0.1

Runtime and extensions



Via container orchestration





Lambda



Fargate



EC2

Think just as an  
alternative hardware





**What**

**Fargate – serverless  
compute for containers**

**Why**

**Story about migrating to  
Fargate**

**How**

**Short Demo**





**What**

**Fargate – serverless  
compute for containers**

**Why**

**Story about migrating to  
Fargate**

**How**

**Short Demo**



## Introducing AWS Fargate

Posted On: Nov 29, 2017

AWS Fargate is a compute engine for deploying and managing containers without having to manage any of the underlying infrastructure. Fargate makes it easy to scale your applications. You no longer have to worry about provisioning enough compute resources for your container applications. You can launch tens or tens of thousands of containers in seconds.

Previously, you needed to manage a cluster of Amazon EC2 instances, pick the instance types, manage the scheduling of the containers, and optimize cluster utilization. With Fargate, all of this goes away. Fargate seamlessly integrates with Amazon ECS. You just define your application as you do today for Amazon ECS. You package your application into task definitions, specify the CPU and memory needed, define the networking and IAM policies each container needs. Once everything is setup, Fargate launches and manages your containers for you.

With Fargate, billing is at a per second granularity and you only pay for what you use. You pay for the amount of vCPU and memory resources your containerized application requests. vCPU and memory resources are calculated from the time your container images are pulled until the Amazon ECS Task terminates, rounded up to the nearest second.

AWS Fargate is available with Amazon ECS and support for Amazon EKS will be available in 2018. Fargate is available in US East (N. Virginia) with other regions coming soon.

Visit the [detail page](#) to learn more.

## AWS Fargate launches platform version 1.4.0

by Massimo Re Ferre | on 08 APR 2020 | in [Amazon Elastic Container Service](#), [Amazon Elastic Kubernetes Service](#), [AWS Fargate](#), [Containers](#) | [Permalink](#) | [Comments](#) | [Share](#)

AWS Fargate is a managed service to run containers. Fargate allows customers to use Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS) to launch applications without the burden of having to deal with the undifferentiated heavy lifting of maintaining, patching, scaling, securing, life-cycling the infrastructure. While Amazon EC2 abstracts away hypervisors and physical servers from customers, AWS Fargate does the same for container runtimes and EC2 instances. If you want to read more about the role of Fargate in the container world, check out [this](#) blog post.

While Fargate makes the infrastructure disappear in the sense that the customer doesn't need to think about it, the infrastructure still exists and it's being managed by AWS. The way the infrastructure features surface to the end users today is through the notion of a Fargate *platform version*. You can read more about it in the [Fargate documentation](#) or you can read the [Fargate platform versions primer](#) blog post. The primer blog post goes into more detail about the philosophy behind why we introduced Fargate platform versions and, for example, the practical reasons why we are not tagging platform version `1.4.6` as LATEST just yet.

Today we are launching [platform version 1.4.0](#) of AWS Fargate.

In this blog post, we are going to provide you with a summary of the Fargate features we are enabling with this release and some of the changes we are making underneath. These underlying changes don't necessarily have a direct relationship with the new customer-visible features but they are just as important.

What's new in Fargate platform version 1.4.0?

2017 → 2021



**Fargate**



**EC2**

**Features were similar**

**Price was no go 2017 → 2021**



# AWS Fargate Pricing

With AWS Fargate, there are no upfront payments and you only pay for the resources that you use. You pay for the amount of vCPU and memory resources consumed by your containerized applications.

## AWS Fargate Pricing

AWS Fargate pricing is calculated based on the vCPU and memory resources used from the time you start to download your container image until the Amazon ECS Task or Amazon EKS<sup>\*</sup> Pod terminates, rounded up to the nearest second.

<sup>\*</sup> See the regions where EKS/Fargate is available <https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/>

## Pricing Details

Pricing is based on requested vCPU and memory resources for the Task or Pod. The two dimensions are independent of each other.

Region: Europe (Ireland)

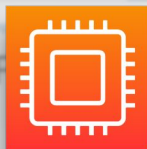
	Price
per vCPU per hour	\$0.04040
per GB per hour	\$0.004445

## Fargate Spot Pricing for Amazon ECS

Region: Europe (Ireland)

	Price
per vCPU per hour	\$0.01364719
per GB per hour	\$0.00152052

Is it cheaper?

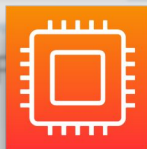


vCPU	Memory	Instance	EC2: On dema	EC2: Reserv	Fargate: Defa	On demand	Reserved
2	4	c5.large	0.096	0.048	0.09874	-97.23%	-91.72%
2	8	m4.large	0.111	0.0564	0.11652	-95.26%	-91.33%
4	16	m4.xlarge	0.222	0.1127	0.23304	-95.26%	-91.25%
2	8	t2.large	0.1008	0.053	0.11652	-86.51%	-85.82%
2	4	t2.medium	0.05	0.0265	0.09874	-50.64%	-50.64%
1	2	t2.small	0.025	0.0132	0.04937	-50.64%	-50.45%
4	16	t2.xlarge	0.2016	0.1059	0.23304	-86.51%	-85.74%

# Is it cheaper?

## Yes and no





vCPU	Memory	Instance	EC2: On dema	EC2: Reserv	Fargate: Defa	On demand	Reserved
2	4	c5.large	0.096	0.048	0.09874	-97.23%	-91.72%
2	8	m4.large	0.111	0.0564	0.11652	-95.26%	-91.33%
4	16	m4.xlarge	0.222	0.1127	0.23304	-95.26%	-91.25%
2	8	t2.large	0.1008	0.053	0.11652	-86.51%	-85.82%
2	4	t2.medium	0.05	0.0265	0.09874	-50.64%	-50.64%
1	2	t2.small	0.025	0.0132	0.04937	-50.64%	-50.45%
4	16	t2.xlarge	0.2016	0.1059	0.23304	-86.51%	-85.74%

# Is it cheaper?

## Yes and no

Amazon EBS pricing

**Free Tier**

AWS Free Tier includes 30GB of storage, 3 million IOPS, and 1GB of snapshot storage with Amazon Elastic Block Store (EBS).  
[View AWS Free Tier details >](#)

Region: Europe (Ireland) +

**Amazon EBS Volumes**

With Amazon EBS, you pay only for what you use. The pricing for Amazon EBS volumes is listed below.

Volume Type	Price
General Purpose SSD (gp2) - Storage	\$0.080GB-month
General Purpose SSD (gp2) - IOPS	3,000 IOPS free and \$0.0015/provisioned IOPS-month over 3,000
General Purpose SSD (gp2) - Throughput	125 MB/s free and \$0.040/provisioned MB/s-month over 125
General Purpose SSD (gp2) Volumes	\$0.11 per GB-month of provisioned storage
Provisioned IOPS SSD (io2) - Storage	\$0.150GB-month
Provisioned IOPS SSD (io2) - IOPS	\$0.072/provisioned IOPS-month up to 32,000 IOPS
	\$0.050/provisioned IOPS-month from 32,001 to 64,000 IOPS
	\$0.030/provisioned IOPS-month for greater than 64,000 IOPS
Provisioned IOPS SSD (io2) Volumes	\$0.118 per GB-month of provisioned storage AND \$0.072 per provisioned IOPS-month
Throughput Optimized HDD (st2) Volumes	\$0.08 per GB-month of provisioned storage



	vCPU	Memory	Instance	EC2: On dema	EC2: Reserv	Fargate: Defa	On demand	Reserved
	2		4 c5.large	0.096	0.048	0.09874	-97.23%	-91.72%
	2		8 m4.large	0.111	0.0564	0.11652	-95.26%	-91.33%
	4		16 m4.xlarge	0.222	0.1127	0.23304	-95.26%	-91.25%
	2		8 t2.large	0.1008	0.053	0.11652	-86.51%	-85.82%
	2		4 t2.medium	0.05	0.0265	0.09874	-50.64%	-50.64%
	1		2 t2.small	0.025	0.0132	0.04937	-50.64%	-50.45%
	4		16 t2.xlarge	0.2016	0.1059	0.23304	-86.51%	-85.74%

# Is it cheaper?

# Yes and no

Because overprovisioning

## AWS Fargate Pricing

AWS Fargate pricing is calculated based on the vCPU and memory resources used from the time you start to download your container image until the Amazon ECS Task or Amazon EKS Pod terminates, rounded up to the nearest second.

\* See the regions where Fargate is available: <https://docs.aws.amazon.com/deep-research/latest/developerguide/regions-product-availability.html>

## Pricing Details

Pricing is based on requested vCPU and memory resources for the Task or Pod. The base Amazon ECS Task or Amazon EKS Pod pricing is \$0.04048 per vCPU per hour.

Region: Europe (Ireland)

per vCPU per hour

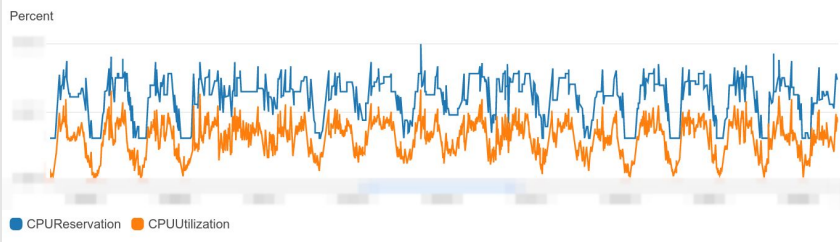
Price

\$0.04048

per GB per hour

\$0.00445

## Fargate Spot Pricing for Amazon ECS



## Amazon EBS pricing

### Free Tier

AWS Free Tier includes 30GB of storage, 3 million IOPS, and 1GB of snapshot storage with Amazon Elastic Block Store (EBS). See AWS Free Tier details.

Region: Europe (Ireland)

### Amazon EBS Volumes

With Amazon EBS, you pay only for what you use. The pricing for Amazon EBS volumes is listed below.

Volume Type	Price
General Purpose SSD (gp2) - Storage	\$0.105GB-month
General Purpose SSD (gp2) - IOPS	3,000 IOPS free and \$0.00125 per provisioned IOPS-month over 3,000
General Purpose SSD (gp2) - Throughput	128 MB/s free and \$0.000125 per provisioned MB/s-month over 128
General Purpose SSD (gp2) Volumes	\$0.17 per GB-month of provisioned storage
Provisioned IOPS SSD (io1) - Storage	\$0.125GB-month
Provisioned IOPS SSD (io1) - IOPS	\$0.0125 per provisioned IOPS-month up to 32,000 IOPS
Provisioned IOPS SSD (io1) Volumes	\$0.00125 per provisioned IOPS-month over 32,000 IOPS
Provisioned IOPS SSD (io1) Volumes	\$0.00125 per provisioned IOPS-month over 32,000 IOPS
Provisioned IOPS SSD (io1) Volumes	\$1.10 per GB-month of provisioned storage over \$0.0125 per provisioned IOPS-month
Throughput Optimized HDD (st1) Volumes	\$0.08 per GB-month of provisioned storage





# Overprovisioning?

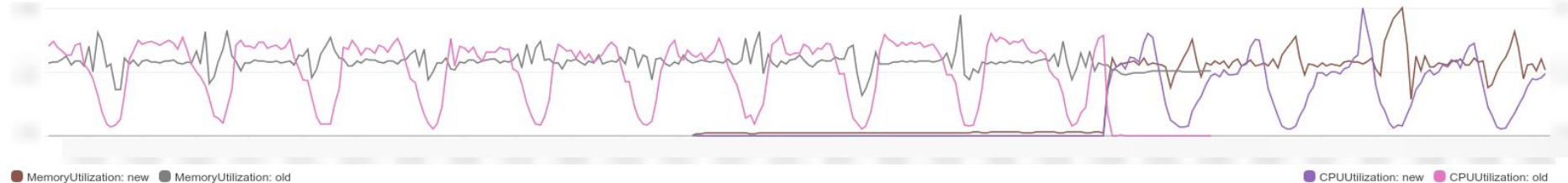
Count



Seconds



Percent



Percent

CPUUtilization: new CPUUtilization: old

# Overprovisioning?

Count

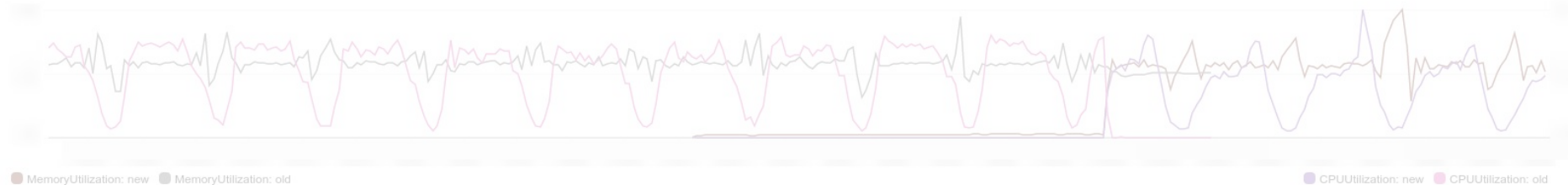


Seconds

Customers are happy when response time is low



Percent



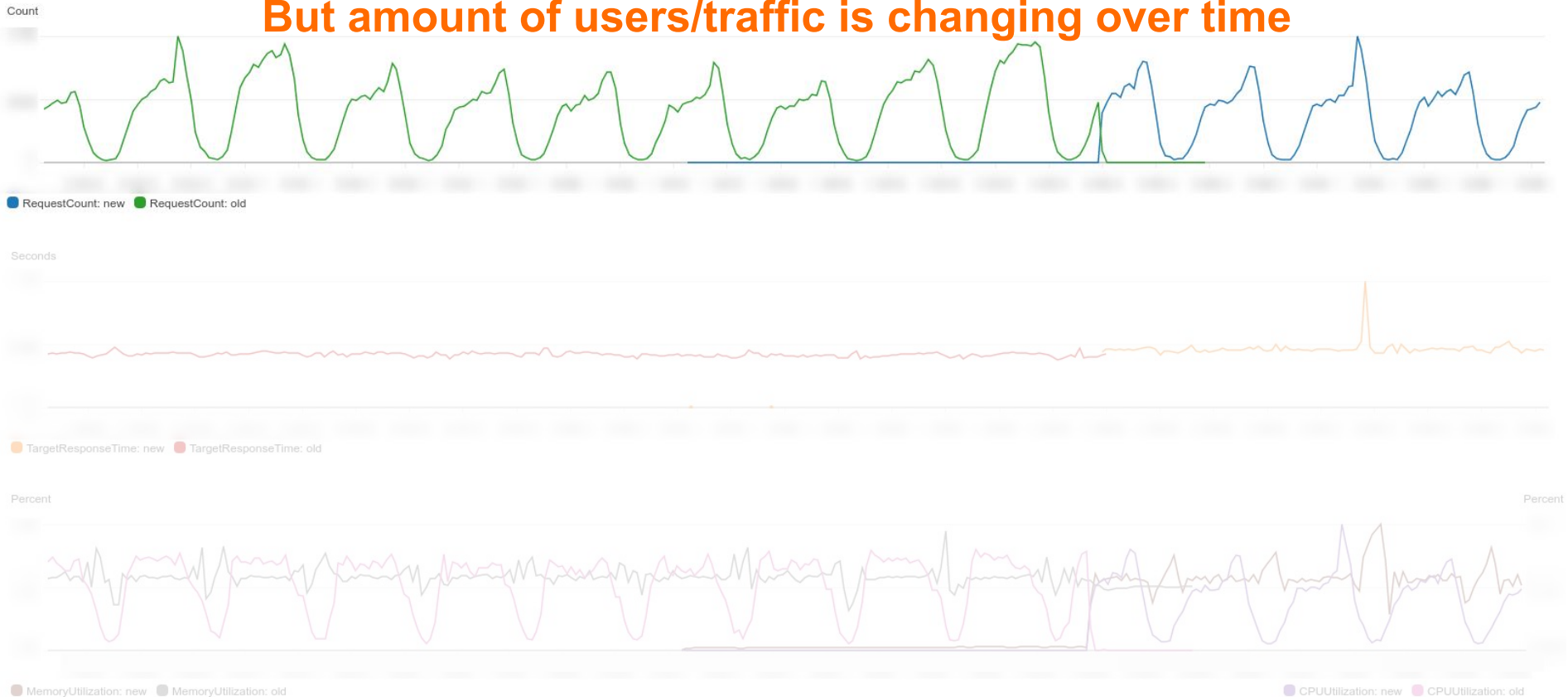
Percent

CPUUtilization: new CPUUtilization: old

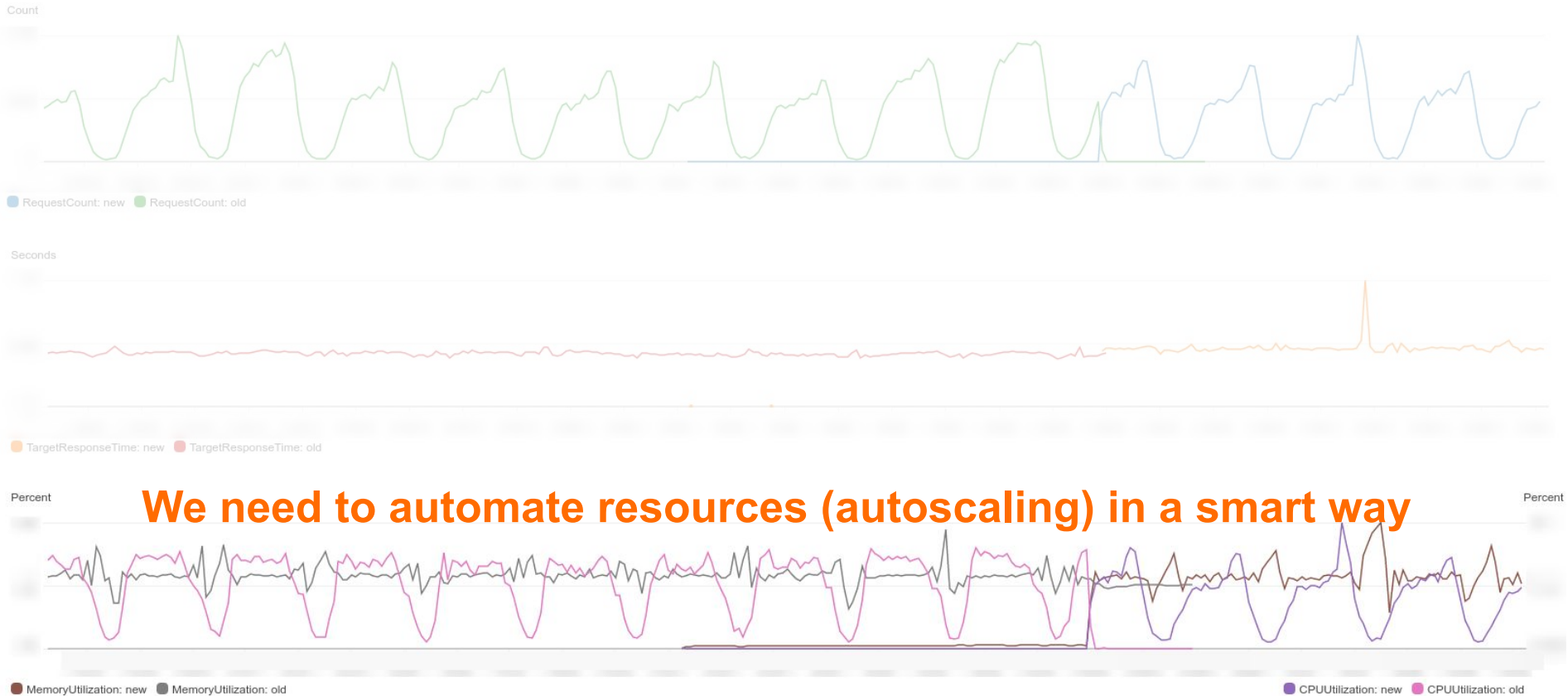


# Overprovisioning?

But amount of users/traffic is changing over time

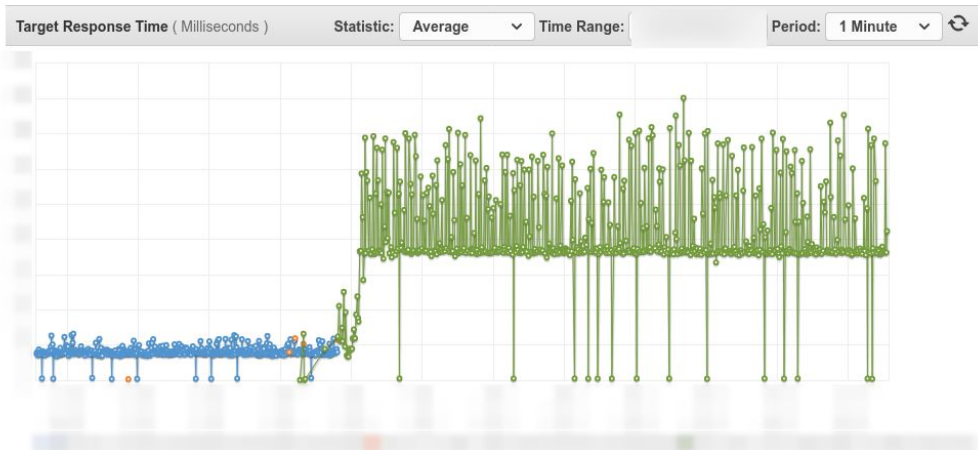


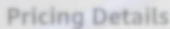
# Overprovisioning?





# Overprovisioning?





Pricing is based on requested vCPU and memory reserved for the Task and Job. The two dimensions are independently priced.

## Fargate Spot Pricing for Amazon ECS

## References

## Compute Savings Plan for Amazon ECS & Amazon EKS

# Is it cheaper?

# It can be



# AWS Fargate Pricing



## Pricing Details

Pricing is based on requested vCPU and memory resources for the Task or Pod. The Task or Pod is billed for the amount of vCPU and memory resources consumed by your containerized applications.

Region: Europe (Ireland) x

per vCPU per hour

per Task per hour

## Fargate Pricing for Amazon ECS

Region: Europe (Ireland) x

per vCPU per hour

per Task per hour

Compute Savings Plan for Amazon ECS & Amazon EKS

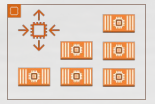
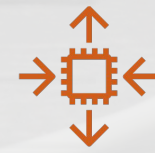
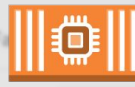
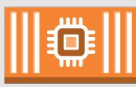
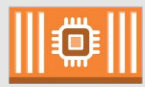
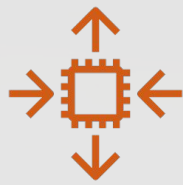
# Is it cheaper?

## It can be

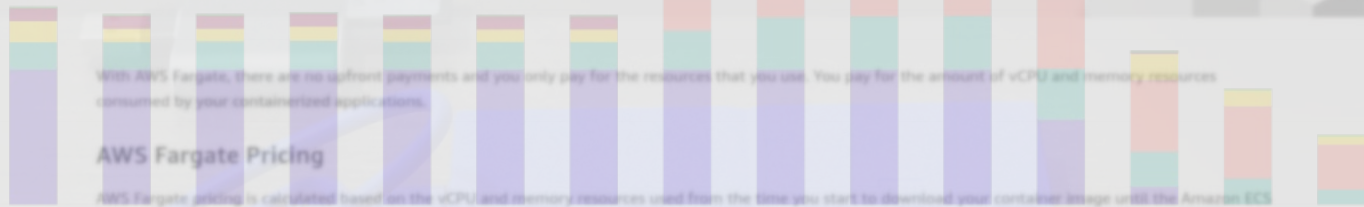
### ← Automating scaling of ⇒

## Containers

## Containers & instance



# AWS Fargate Pricing



## Pricing Details

Pricing is based on requested vCPU and memory resources for the Task or Pod. The Task or Pod is billed for the amount of vCPU and memory resources consumed by your containerized applications.

Region: Europe (Ireland) X

Price

Price

\$0.04040

\$0.00445

## Pricing for Amazon ECS

Region: Europe (Ireland) X



per vCPU per hour



per vCPU per hour

Compute Savings Plan for Amazon ECS & Amazon EKS

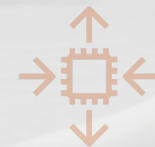
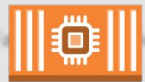
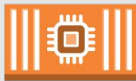
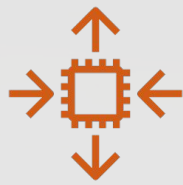
# Is it cheaper?

## It can be

### Containers

← Automating scaling of ⇒

# Less overprovisioning







**What about bad things?**

# What about bad things?

Costly for small services (<80€ per month)

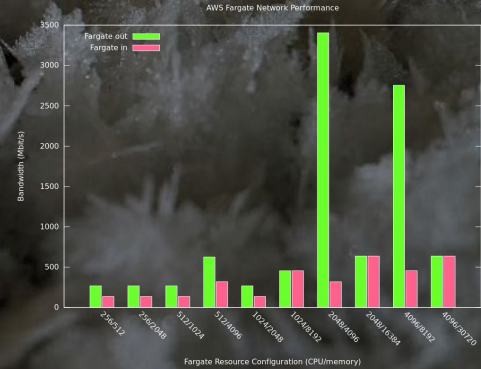




# What about bad things?

Costly for small services (<80€ per month)

Network speed not configurable



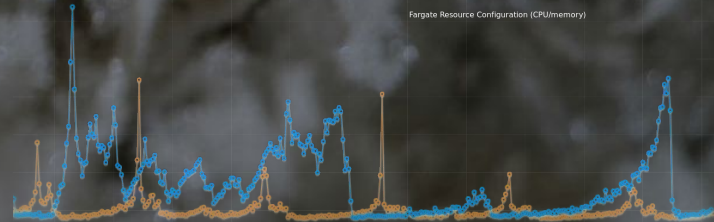
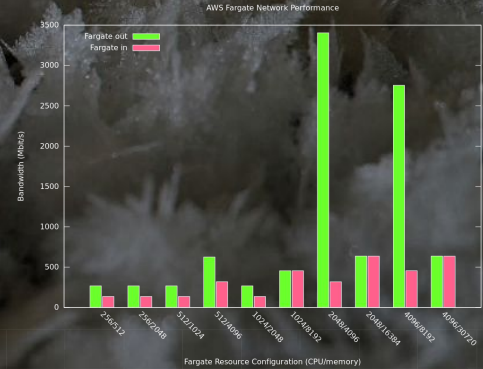
# What about bad things?

Costly for small services (<80€ per month)



Network speed not configurable

Starts at the same time (for cache invalidation)





# What about bad things?

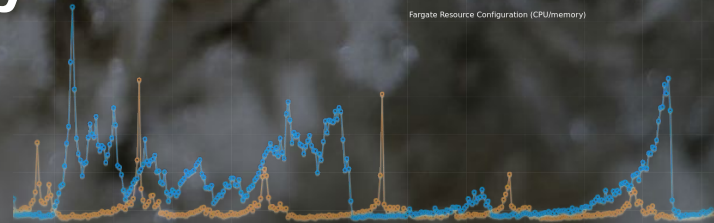
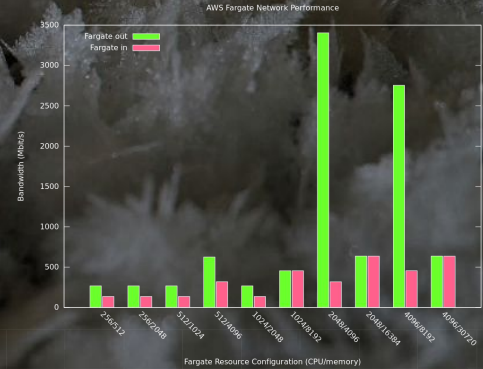
## Costly for small services (<80€ per month)



## Network speed not configurable

**Starts at the same time (for cache invalidation)**

## No capacity reservation for Black Friday





# What about bad things?

Costly for small services (<80€ per month)

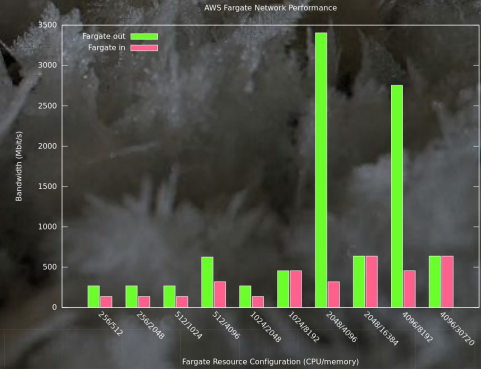


Network speed not configurable

Starts at the same time (for cache invalidation)

No capacity reservation for Black Friday

Other limitations and not battle tested





**What**

**Fargate – serverless  
compute for containers**

**Why**

**Autoscaling  
(overprovisioning)  
for spiky traffic**

**How**

**Short Demo**





**What**

**Fargate – serverless  
compute for containers**

**Why**

**Autoscaling  
(overprovisioning)  
for spiky traffic**

**How**

**Short Demo**



index.php

<?php

```
echo "Hello VilniusPHP 0v63";
```

```
$metadata = json_decode(file_get_contents(getenv('ECS_CONTAINER_METADATA_URI_V4')));
```

```
echo "<hr/><pre>"; json_encode($metadata, JSON_PRETTY_PRINT) "</pre>";
```

Raw

<https://gist.github.com/aurelijusb/b3e6ddd6191fe759805efa51f2349135>

```
eu-west-1.elb.amazonaws.com
Hello VilniusPHP 0v63

{
  "DockerId": "07e84267a3caef98e5558c5f922bd-265927825",
  "Name": "web",
  "DockerName": "web",
  "Image": "327920853133.dkr.ecr.eu-west-1.amazonaws.com/aw-cdk/assets-8ad19778f91d673d5bec63b1b79c8b4730a7bf4c427c23a678b99747f",
  "ImageId": "sha256:a59092c0bf4f6d8a7c1e52a7087682fcd8af6b097c8a0db74515c79197",
  "Labels": {
    "com.amazonaws.ecs.cluster": "arn:aws:ecs:eu-west-1:327920853133:cluster/vilniusPHP-0x63", "vilniusPHPClusterASCI903A-L20zcfQ0NMq",
    "com.amazonaws.ecs.container-name": "web",
    "com.amazonaws.ecs.task-arn": "arn:aws:ecs:eu-west-1:327920853133:task/vilniusPHP-0x63-vilniusPHP0x63testVilniusPHPServiceTaskSet1KACB4BA", "vilniusPHPClusterASCI903A-L20zcfQ0NMq/07b0429973c447f80b055c5f922bd",
    "com.amazonaws.ecs.task-definition-family": "VilniusPHP0x63testVilniusPHPServiceTaskSet1KACB4BA",
    "com.amazonaws.ecs.task-definition-version": "0"
  },
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Status": {
    "CPU": 2
  },
  "CreatedAt": "2021-01-30T17:15:20.172802415Z",
  "StartedAt": "2021-01-30T17:15:20.172802415Z",
  "Type": "ECS",
  "Networks": {
    "NetworkMode": "awsvpc",
    "IPAddresses": [
      {
        "ip": "10.0.227.233"
      }
    ],
    "InterfaceName": "eth0",
    "MACAddress": "80:00:02:0e:51:21",
    "IPv4SubnetCIDRBlock": "10.0.192.0/18",
    "VilniusPHP0x63testVilniusPHPServiceTaskSet1KACB4BA": {
      "DomainName": "eu-west-1.elb.amazonaws.com",
      "DomainNameSearchPath": [
        "eu-west-1.elb.amazonaws.com"
      ],
      "PrivateDNSName": "ip-10-0-227-233.eu-west-1.elb.amazonaws.com",
      "Subnet": "subnet-1a4d8f8c"
    }
  },
  "ContainerARN": "arn:aws:ecs:eu-west-1:327920853133:container/vilniusPHP-0x63-4769-42fc-378535577216",
  "LogOptions": {
    "awslogs-group": "vilniusPHP-0x63-test-vilniusPHPServiceTaskSet1KACB4BA",
    "awslogs-region": "eu-west-1",
    "awslogs-stream": "VilniusPHPServiceWebV07e84267a3caef98e5558c5f922bd"
  },
  "LogDriver": "awslogs"
}
```

Services Tasks EC2 Instances Metrics Schedulers Pools Tags Capacity Providers

Run VilniusPHP-0x63-test

Desired/Running/Stopped/Failed

| Task                 | Task Definition      | Current Instance     | Current Status | Started At                     | Stopped At | Group                | Launch Type | Platform Version |
|----------------------|----------------------|----------------------|----------------|--------------------------------|------------|----------------------|-------------|------------------|
| VilniusPHP-0x63-test | VilniusPHP-0x63-test | i-07b0429973c447f80b | Running        | 2021-01-30T17:15:20.172802415Z |            | VilniusPHP-0x63-test | ECS         | 1.4.0            |
| VilniusPHP-0x63-test | VilniusPHP-0x63-test | i-07b0429973c447f80b | Running        | 2021-01-30T17:15:20.172802415Z |            | VilniusPHP-0x63-test | ECS         | 1.4.0            |

21e6f0b5d46e: Layer already exists  
eap397dd1e81: Layer already exists  
cb424133e4c: Layer already exists  
c4a5852dbab07580e07bd58b69c8c0d3a77a91692c48f81e8d908d78ed003d: digest: sha256:a59092c0bf4f6d8a7c1e52a7087682fcd8af6b097c8a0db74515c79197 size: 3242 [100%] success: Pull completed c4a5852dbab07580e07bd58b69c8c0d3a77a91692c48f81e8d908d78ed003d:current  
VilniusPHP-0x63: creating CloudFormation changeset...

VilniusPHP-0x63-test

Outputs:  
VilniusPHP-0x63-test.VilniusPHPServiceLoadBalancerDNS7FIA37EF = Vilni-Vilni-10E 270830.eu-west-1.elb.amazonaws.com  
VilniusPHP-0x63-test.VilniusPHPServiceServiceURLAE06DF2 = http://Vilni-Vilni-10E 10270830.eu-west-1.elb.amazonaws.com

Stack ARN:  
arn:aws:cloudformation:eu-west-1:327920853133:stack/VilniusPHP-0x63-80b5cef0-630f-11eb-88ee-0a27cca8e7e9





**What**

**Fargate – serverless  
compute for containers**

**Why**

**Autoscaling  
(overprovisioning)  
for spiky traffic**

**How**

**Short Demo**



The background is a dark, monochromatic image of a winter forest. It features numerous thin, snow-covered evergreen trees and bare deciduous branches. The ground is covered in a thick layer of snow, with some small, snow-laden shrubs in the foreground. The overall tone is dark and moody, with the white text providing a sharp contrast.

**Theory vs Practice**  
**Let's test it/small**





Thank you  
**Fargate**  
Time for questions

Aurelijus Banelis



VilniusPHP 0x63  
2021-02-04