

# AUTOSCALING LIKE A PRO: HOW TO USE EC2 AUTO SCALING LIFECYCLE HOOKS, THE RIGHT WAY.

Amazon EC2 Auto Scaling

Lifecycle Hooks



beSharp | 4 October 2019

---

Have you ever found it useful to **access an EC2 instance, belonging to an Auto Scaling Group**, to configure it or to simply get an idea of what went wrong before an unexpected termination?

Well, in those cases EC2 Auto Scaling Lifecycle Hooks come in handy. In this article, we will give you a deep understanding of **how Lifecycle Hooks work and what are their common use cases**.

Just for clarity let's review EC2 Auto Scaling's main characteristics and features, before going in detail with EC2 Auto Scaling Lifecycle Hooks. **EC2 Auto Scaling** is an AWS service that allows you to **adapt the number of EC2 instances, serving your application, to meet the incoming load**. These instances are logically grouped into **Auto Scaling groups** for which you can set the **minimum, maximum and desired capacity**. If there are no scaling policies attached to the Auto Scaling group, by default it maintains the desired amount of healthy instances, performing periodic health checks. Unhealthy instances will be terminated and replaced with new ones.

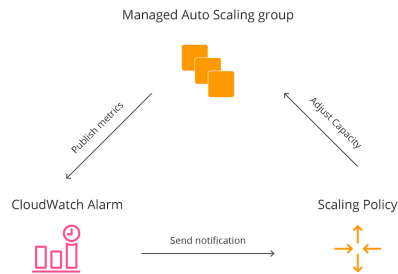
Besides manual scaling, in which you actively change minimum, maximum or desired capacity, there are two **automatic scaling** types: **Scheduled Scaling and Dynamic Scaling**.

**Scheduled Scaling** should be used if you know in advance how your **application's traffic patterns** behave. In this case, you have to create a rule, called scheduled action, that instructs Auto Scaling to **increase and decrease the Auto Scaling group's capacity based on predicted load changes**.

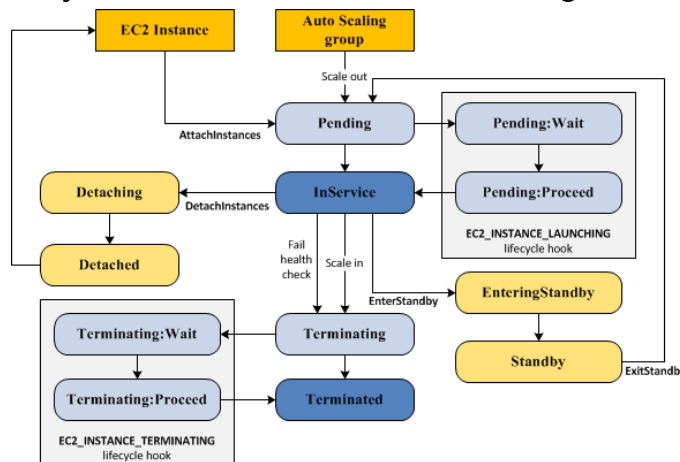
Instead, when you need to scale in response to **load changes that are not predictable Dynamic Scaling comes to the rescue**. It provides different policy types that you can use to instruct Auto Scaling how to react to demand changes. There are three types of Dynamic Scaling policies:

- **Simple Scaling;**
- **Step Scaling** - adjustments varies based on the size of the alarm breach;

- **Target Tracking Scaling** – you can think of it as a Feedback Loop where scaling adjustments are calculated based on the gap between the metric and the target. Auto Scaling acts to keep the metric close to the target. This kind of policy is recommended in situations in which a utilization metric is proportional to the Auto Scaling group’s capacity.



If an instance belongs to an Auto Scaling group, **its lifecycle differs from other EC2 instances’ one**. EC2 Auto Scaling Lifecycle is well illustrated in the following scheme.



When a new instance is **launched in response to a scale-out event**, it first enters the **Pending state**. This state is designed to allow you to configure (preferably automatically) the EC2 instance and to check its healthiness before moving into the InService state. While into the InService state, the EC2 instance can potentially move **to one of the following three states**:

- **Detaching** – in response to a detach request;
- **EnteringStandby** – if you want to put the instance into a Standby state;
- **Terminating** – if a scale-in event occurs or if the EC2 instance fails a number of consecutive health checks sufficient to consider it unhealthy.

Pending, Detaching, EnteringStandby and Terminating can be considered **transitional states** that anticipate, respectively, **InService, Detached, StandBy and Terminated states**.

**Pending and Terminating transitional states can be paused** through EC2 Auto Scaling Lifecycle Hooks, to perform custom logics during instances’ launch or termination.

Pending state’s Lifecycle Hook is called **EC2\_INSTANCE\_LAUNCHING** and it handles **Pending:Wait** and **Pending:Proceed** additional states.

In the same way, Terminating the state's Lifecycle Hook is called **EC2\_INSTANCE\_TERMINATING** and it handles **Terminating:Wait** and **Terminating:Proceed** additional states.

In both cases, EC2 instances remain in a Wait state until the **timeout period** ends. There is also the possibility to conclude the Lifecycle Hook and continue to the next state before the timeout period expires.

Lifecycle Hook's result can be either **CONTINUE** or **ABANDON**. EC2\_INSTANCE\_LAUNCHING's result determines whether the instance should be terminated or put into InService state. On the other hand, EC2\_INSTANCE\_TERMINATING's result does not affect EC2 instance's lifecycle; it always ends up in the Terminated state.

You may be wondering why you should **hook Pending and Terminating states!** Well, it obviously depends on your application scenario.

The following are some of the common use cases where pausing Pending, Terminating or both transitional states is helpful.

## Pending state hook

# Instance configuration

As mentioned before, you can pause an EC2 instance lifecycle to Pending state and configure it before it moves to InService state. This is critical when the setup takes considerable time, **avoiding the instance to receive requests when it is not yet ready to handle them.**

For example, AWS CodeDeploy service relies on EC2\_INSTANCE\_LAUNCHING lifecycle hook to manage application deployments to EC2 instances that belong to an Auto Scaling group. This behavior is perfectly described in the AWS DevOps Blog's article "[Under the Hood: AWS CodeDeploy and Auto Scaling Integration](#)".

This mechanism actually works also for custom solutions, such as configuration scripts embedded in EC2 instances' User Data.

## Terminating state hook

# Log files download

A: "The EC2 instance is down! "

B: "Ok, let's find the cause."

A: "Ehm... logs were on the instance. We've no other copy of them."

Well, if your common practice is to exploit an agent to push logs periodically to another place, you will never end up in such a situation. But you know, bad things happen!

...

B: "Sure! The log exporting agent was installed on it."

A: "Yes, but it seems it wasn't working. Maybe it crashed."

In this case, EC2\_INSTANCE\_TERMINATING hook allows you to upload logs to another place, for example, an S3 bucket.

## What went wrong?

Sometimes it can be useful to log into an EC2 instance to understand what went wrong. Using EC2 Auto Scaling Lifecycle Hooks, an SNS Topic and a Lambda Function, you can easily set up a system that notifies you when an unhealthy instance is terminated. The role of the Lambda Function is to let the instance's lifecycle proceed if it is healthy and to notify the user if it is unhealthy.

## Hands-on!

Now we will demonstrate **how to set up an EC2 Auto Scaling Lifecycle Hook** that waits until initial configuration completes before putting a new instance to InService state.

The configuration step is represented by a User Data script, defined in the Auto Scaling group's Launch Template.

```
#!/bin/bash
apt update && \

apt install snapd && \

snap install aws-cli --classic && \

apt install -y apache2 && \

ufw allow 'Apache' && \

INSTANCE_ID=`wget -q -O - http://instance-data/latest/meta-data/instance-id` && \

/snap/bin/aws autoscaling complete-lifecycle-action --lifecycle-action-result CONTINUE --instance-id $INSTANCE_ID --lifecycle-hook-name test-hook --auto-scaling-group-name test-asg --region eu-west-1 || \

/snap/bin/aws autoscaling complete-lifecycle-action --lifecycle-action-result ABANDON --instance-id $INSTANCE_ID --lifecycle-hook-name test-hook --auto-scaling-group-name test-asg --region eu-west-1
```

The User Data is going to **install Apache Web Server on Ubuntu 18.04**. When the installation is completed, it harvests the instance-id and uses it to complete the lifecycle action with CONTINUE result, which tells to the Auto Scaling group to put the instance into InService state.

If anything goes wrong, **complete-lifecycle-action** command will be invoked with **ABANDON** result, telling the Auto Scaling group to terminate the instance.

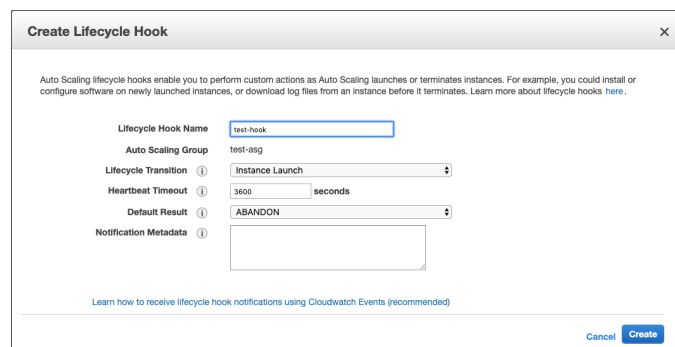
Take care of **-lifecycle-hook-name** and **-auto-scaling-group-name** parameters' values used in the configuration script; they should correspond, respectively, to the name of the EC2 Auto Scaling Lifecycle Hook and to the name of the EC2 Auto Scaling group that you are going to create later.

Firstly, you have to create a **new Launch Template**, specifying **ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20190722.1** as the source AMI and the configuration script, described above, as User Data field.

Then, you can create a new Auto Scaling group based on the just created Launch Template. During the Auto Scaling group configuration, specify one or more subnets in which the EC2 instances will have internet access to. **Otherwise, they will not be able to install Apache Web Server.**

Before moving on, make sure that the Auto Scaling group's initial DesiredCapacity and min parameters' values are set to zero; the initial max parameter's value can be set to one.

Move to the Auto Scaling group's Lifecycle Hooks tab and click the Create Lifecycle Hook button. Configure the lifecycle hook as illustrated in the picture below.



The screenshot shows a 'Create Lifecycle Hook' dialog box with the following configuration:

- Lifecycle Hook Name: test-hook
- Auto Scaling Group: test-asg
- Lifecycle Transition: Instance Launch
- Heartbeat Timeout: 3600 seconds
- Default Result: ABANDON
- Notification Metadata: (empty text area)

At the bottom right, there are 'Cancel' and 'Create' buttons. A link at the bottom left reads 'Learn how to receive lifecycle hook notifications using Cloudwatch Events (recommended)'.

To test the lifecycle hook's behavior, increase the DesiredCapacity parameter's value by one.

A new instance will be added to the Auto Scaling group. Under the Auto Scaling group's Instances tab, you should see the newly created instance's Lifecycle column set to Pending:Wait. That means the instance's lifecycle is paused to the Pending state until complete-lifecycle-action command is invoked from the User Data script.

When the Status column changes to InService, log into the new EC2 instance through SSH. To check the status of the Apache Web Server, type

```
sudo systemctl status apache2
```

If the response of this command is something like...

- apache2.service - The Apache HTTP Server

Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)

Drop-In: /lib/systemd/system/apache2.service.d

└─apache2-systemd.conf

Active: active (running) since Wed 2019-10-02 12:56:20 UTC; 3min 30s ago

Main PID: 3334 (apache2)

Tasks: 55 (limit: 1125)

CGroup: /system.slice/apache2.service

├─3334 /usr/sbin/apache2 -k start

├─3336 /usr/sbin/apache2 -k start

└─3337 /usr/sbin/apache2 -k start

... it means **you have successfully installed Apache Web Server on your Ubuntu 18.04** instance during the instance lifecycle's Pending:Wait state; the instance changed its state to InService only after a successful configuration!

Satisfied?

In this article, we focused on how Lifecycle Hooks work and how they can come in handy in everyday dev-life. Did you try any further configuration? Tell us more in the comments!

See you, guys!



## **beSharp**

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

## **Get in touch**

beSharp.it  
proud2becloud@besharp.it

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189