EFFICIENT AND RESILIENT WORKLOADS WITH AMAZON EC2 AUTO SCALING

BY P.LOKESH

This workshop is designed to get you familiar with the concepts and best practices for effectively and efficiently scaling Amazon EC2 capacity using Amazon EC2 Auto Scaling and its features, including predictive scaling and warm pools.

In this workshop you will assume the role of a Cloud Architect tasked with finding an efficient Auto Scaling solution for a sample application. You have been provided with a monolithic application which receives cyclical traffic; high usage during regular business hours and low usage during evenings and weekends. The application takes a long time (over 10 minutes) to initialize, causing a noticeable latency impact on application performance when scaling out.

In this workshop you will conduct exercises to find the correct Auto Scaling solution to ensure your application is ready in minimal time and scales quickly by launching capacity in advance of forecasted load without the need to overprovision.

Prerequisites

This is an L300 workshop, we expect readers to be familiar with the concepts below

- Amazon EC2
- Amazon EC2 Auto Scaling groups
- AWS CloudWatch metrics for EC2 .

For those unfamiliar with these concepts, the lab should provide enough context to understand what's happening, and at an AWS event there should be staff nearby to help if you get stuck.

Starting the Workshop

This workshop is self-paced. You can run it in your own account or at an AWS event.

Instructions will primarily be given using the AWS Command Line Interface (CLI) as the AWS Management Console changes over time and the CLI does not. However, most steps in the workshop can be done in the AWS Management Console directly if desired, and we use screenshots as examples, as well. Feel free to use whatever is comfortable for you.

While the workshop provides step-by-step instructions, when convenient, please take a moment to look around and understand what is happening at each step. The workshop is meant as a guide to get you started, but you will learn the most by digesting each of the steps and thinking about how they would apply in your own environment. You might even consider experimenting with the steps to challenge yourself.

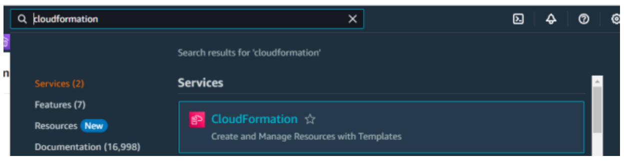
Running the workshop self-paced, in your own AWS account

Thank you for taking the time to explore Predictive Scaling and Warm Pools on your own time. The instructions below will get your account into a ready state to run this lab without the need to attend an AWS-hosted event.

Deploy the CloudFormation stack to build the lab

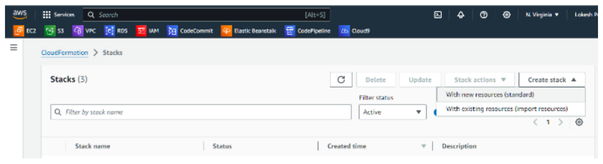
To save time on the initial setup you will deploy a CloudFormation template to create various supporting resources including IAM policies and roles, EC2 Launch Template, VPC, Subnets and a Cloud9 IDE environment in which you can execute the steps of the workshop

1. You can view and download the CloudFormation template from GitHub. Take a moment to review the CloudFormation template so you understand the resources it will be creating.
   ***Tip:** Right click the link and '**Save Link As**', to download the file*
2. Browse to the AWS CloudFormation console and click **'Create stack'**, then **'With new resources(standard)'**.

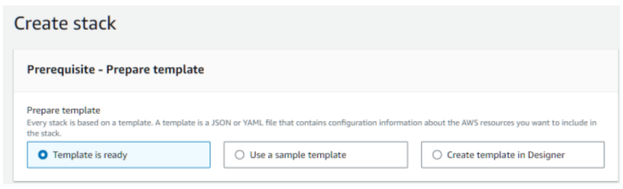


cloudformation

Search results for 'cloudformation'



3. In the **'Specify template'** section, select **'Upload a template file'**.
   Click **'Choosefile'** and select the template you downloaded in step 1 and click **'Next'**.



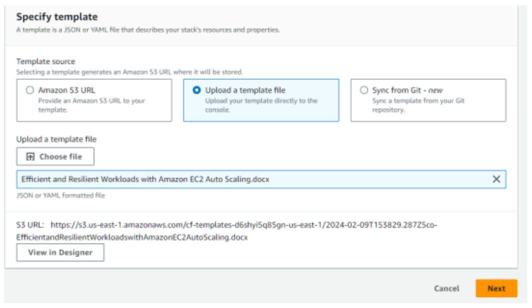
Create stack

Prerequisite - Prepare template

Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Template is ready] Use a sample template Create template in Designer



Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

Amazon S3 URL — Provide an Amazon S3 URL to your template. [✓ Upload a template file — Upload your template directly to the console.] Sync from Git - *new* — Sync a template from your Git repository.

Upload a template file

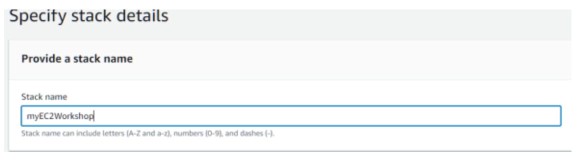
Choose file

Efficient and Resilient Workloads with Amazon EC2 Auto Scaling.docx

JSON or YAML formatted file

S3 URL: https://s3.us-east-1.amazonaws.com/cf-templates-d6shyi5q85gn-us-east-1/2024-02-09T153829.287Z5co-EfficientandResilientWorkloadswithAmazonEC2AutoScaling.docx

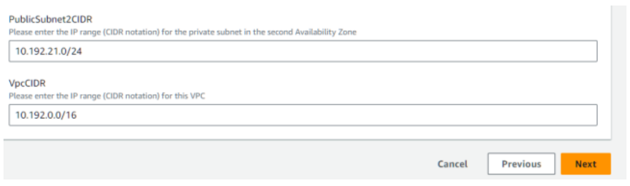
4. In the **'Specify stack details'** section, enter a Stack name and click **'Next'**. (Tip: The stack name cannot contain spaces, use **myEC2Workshop** for example.)


Specify stack details

Provide a stack name

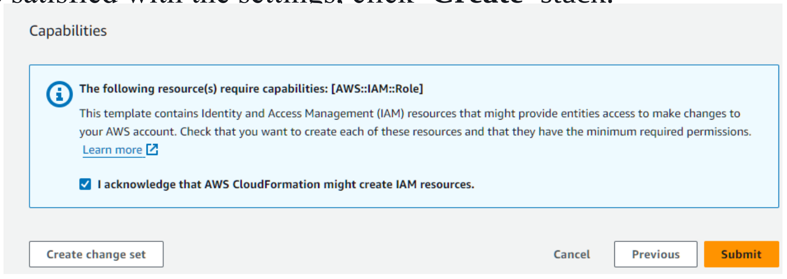
Stack name: myEC2Workshop

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).


PublicSubnet2CIDR
Please enter the IP range (CIDR notation) for the private subnet in the second Availability Zone
10.192.21.0/24

VpcCIDR
Please enter the IP range (CIDR notation) for this VPC
10.192.0.0/16

5. In **'Configure stack options'**, you don't need to make any changes and click **'Next'**.
6. Review the information for the stack. At the bottom under **'Capabilities'**, select **'I acknowledge that AWS CloudFormation might create IAM resources'**. When you're satisfied with the settings, click **'Create'** stack.


Capabilities

The following resource(s) require capabilities: [AWS::IAM::Role]

This template contains Identity and Access Management (IAM) resources that might provide entities access to make changes to your AWS account. Check that you want to create each of these resources and that they have the minimum required permissions.

Learn more

✓ I acknowledge that AWS CloudFormation might create IAM resources.

The CloudFormation stack takes about 45 minutes for the environment setup and the bootstrap script to finish creating the CloudWatch metrics data.

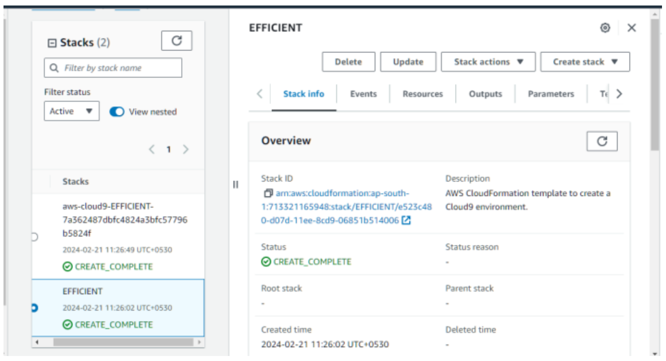
Working Environment

Log in to the AWS Cloud9 IDE

If you are not at an AWS Event (or if you're just curious), you can follow the outputs of the CloudFormation template as follows:

1. In the AWS CloudFormation console, select the **efficient-auto-scaling-quickstart-cnf** (or the name you used if you built it yourself) stack in the list.

2. In the stack details pane, click the **'Events'** tab.
3. Click the refresh icon to update the events in the stack creation.
4. When AWS CloudFormation has successfully created the stack you will see **CREATE_COMPLETE**
5. In the stack details pane, click the **'Outputs'** tab.
6. Click on the url of the AWS Cloud9 environment; it's the value of **Cloud9 IDE** in the CloudFormation stack outputs.


Stacks (2)

Filter by stack name

Filter status: Active — View nested

Stacks

aws-cloud9-EFFICIENT-7a362487dbfc4824a3bfc57796b5824f
2024-02-21 11:26:49 UTC+0530
CREATE_COMPLETE

EFFICIENT
2024-02-21 11:26:02 UTC+0530
CREATE_COMPLETE

EFFICIENT

Overview

Stack ID: arn:aws:cloudformation:ap-south-1:713321165948:stack/EFFICIENT/e523c480-d07d-11ee-8cd9-06851b514006

Description: AWS CloudFormation template to create a Cloud9 environment.

Status: CREATE_COMPLETE

Status reason: -

Root stack: -

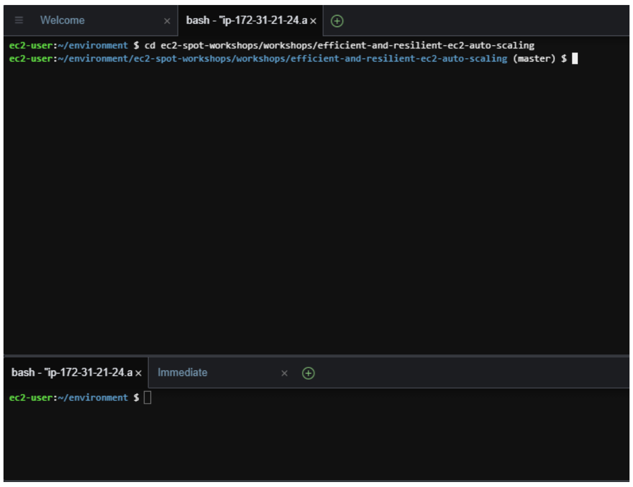
Parent stack: -

Created time: 2024-02-21 11:26:02 UTC+0530

Deleted time: -

Inside the Cloud9 IDE

1. In the Cloud9 IDE, check the folders in left navigation. Look for folder named **ec2-spot-workshops** and confirm it exists.
2. In the Cloud9 terminal, change into the workshop directory.
   cd ec2-spot-workshops/workshops/efficient-and-resilient-ec2-auto-scaling



```
ec2-user:~/environment $ cd ec2-spot-workshops/workshops/efficient-and-resilient-ec2-auto-scaling
ec2-user:~/environment/ec2-spot-workshops/workshops/efficient-and-resilient-ec2-auto-scaling (master) $
```

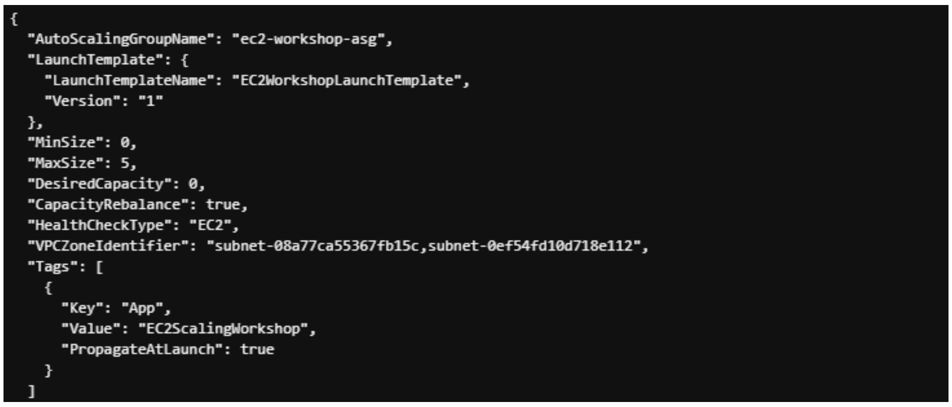
```
ec2-user:~/environment $
```

Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling helps you maintain application availability and allows you to dynamically scale your Amazon EC2 capacity up or down automatically according to conditions you define. You can use Amazon EC2 Auto Scaling for fleet management of EC2 instances to help maintain the health and availability of your fleet and ensure that you are running your desired number of Amazon EC2 instances.

You have decided to leverage EC2 Auto Scaling to scale your application efficiently. To start, you create an EC2 Auto Scaling group without any Auto Scaling policies.

1. You will create an Auto Scaling group using the **asg.json** configuration file, open the file to review the configuration.



```
{
  "AutoScalingGroupName": "ec2-workshop-asg",
  "LaunchTemplate": {
    "LaunchTemplateName": "EC2WorkshopLaunchTemplate",
    "Version": "1"
  },
  "MinSize": 0,
  "MaxSize": 5,
  "DesiredCapacity": 0,
  "CapacityRebalance": true,
  "HealthCheckType": "EC2",
  "VPCZoneIdentifier": "subnet-08a77ca55367fb15c,subnet-0ef54fd10d718e112",
  "Tags": [
    {
      "Key": "App",
      "Value": "EC2ScalingWorkshop",
      "PropagateAtLaunch": true
    }
  ]
```

2. Create the Auto Scaling group using the command below. This command does not return any output if it is successful.

   ```
   aws autoscaling create-auto-scaling-group --cli-input-json file://asg.json
   ```

3. Then run this command to enable CloudWatch metrics collection for the Auto Scaling group, which will help you in monitoring the capacity in the group.

   ```
   aws autoscaling enable-metrics-collection \

   --auto-scaling-group-name ec2-workshop-asg \

   --granularity "1Minute"
   ```

4. Browse to the Auto Scaling console and check out your newly created auto scaling group. At this step of the workshop, the auto scaling group will have no instances running, as the desired number of instances is set to 0

5. Scaling Techniques

Scaling is just part of what an Auto Scaling group can do, and there are many different ways to do it. We're going to explore one of them in-depth today, but in case you're interested in other techniques expand the snippets below to learn more.

- Manual scaling
- Dynamic scaling
- Scheduled scaling
- Predictive scaling

- **Manual scaling**
- At any time, you can change the size of an existing Auto Scaling group manually. You can either update the desired capacity of the Auto Scaling group, or update the instances that are attached to the Auto Scaling group. Manually scaling your group can be useful when automatic scaling is not needed or when you need to hold capacity at a fixed number of instances.
- **Dynamic scaling**
- A dynamic scaling policy instructs Amazon EC2 Auto Scaling to track a specific CloudWatch metric, and it defines what action to take when the associated CloudWatch alarm is in ALARM. The metrics that are used to invoke the alarm state are an aggregation of metrics coming from all of the instances in the Auto Scaling group. When the policy is in effect, Amazon EC2 Auto Scaling adjusts the group's desired capacity up or down when the threshold of an alarm is breached.
- Dynamic scaling policies are reactive. They allow you to track a specific CloudWatch metric and to take an action when the CloudWatch alarm is triggered. Predictive scaling policies are used in combination with dynamic scaling policies when your application demand changes rapidly but with a

recurring pattern or when your application instances require a longer time to initialize.

- **Scheduled scaling**
- Scheduled scaling helps you to set up your own scaling schedule according to predictable load changes. For example, let's say that every week the traffic to your web application starts to increase on Wednesday, remains high on Thursday, and starts to decrease on Friday. You can configure a schedule for Amazon EC2 Auto Scaling to increase capacity on Wednesday and decrease capacity on Friday.
- **Predictive scaling**
- Predictive scaling uses machine learning to predict capacity requirements based on historical data from CloudWatch. The machine learning algorithm consumes the available historical data and calculates capacity that best fits the historical load pattern, and then continuously learns based on new data to make future forecasts more accurate.

Predictive Scaling

In our example, after reviewing the differences between scaling policies, you have decided to go with Predictive Scaling because it can help you scale faster by launching capacity in advance of forecasted load compared to using only dynamic scaling, which is reactive in nature. Predictive scaling can also potentially save you money on your EC2 bill by helping you avoid the need to overprovision capacity.

How predictive scaling works

Predictive scaling requires 24 hours of metric history before it can generate forecasts. Predictive scaling finds patterns in CloudWatch metric data from the previous 14 days to create an hourly forecast for the next 48 hours. Forecast data is updated daily based on the most recent CloudWatch metric data.

To use predictive scaling, you first create a scaling policy with a metric pair (we'll cover this in a moment) and a target utilization.

For example, maybe you want the aggregate CPU load across your Auto Scaling group to be 75% after having determined that to be the best balance between perfomance and cost optimization. Your metric would be CPU and your target 75.

Most workloads will be able to use one of pre-built models native to predictive scaling that include **CPU**, **Network in**, **Network out**, and **Application Load Balancer request count**. We will use CPU utilization in this workshop.

*If your workload would benefit from scaling using a different metric, it is entirely possible to do so as long as it is published as a CloudWatch metric. You will need to provide a **load metric**, which represents total demand on an Auto Scaling group, and a **scaling metric**, representing the average utilization per instance. Predictive scaling will also use the instance count, already provided to CloudWatch metrics by the Auto Scaling group.*

A core assumption of predictive scaling is that the Auto Scaling group is homogenous and all instances are of equal capacity

In this workshop you use t3.micro, a single instance type, in the Auto Scaling group. If your actual workload uses Auto Scaling groups with multiple instance types (mixed instance groups), then use caution when creating predictive scaling policies. Because its algorithms use instance count as an input, predictive scaling with mixed instance groups can forecast capacity inaccurately and launch instance types of unequal capacity. leading to insufficient capacity or excessive capacity with excessive costs.

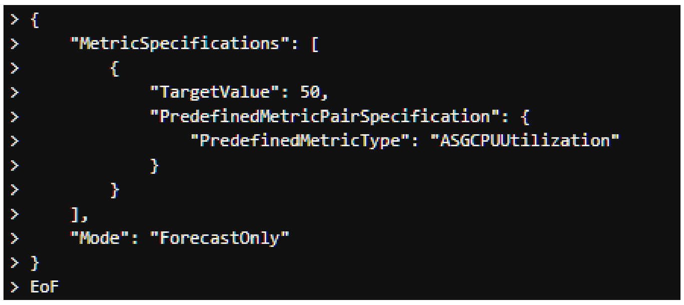
Enable and configure predictive scaling policy

Predictive scaling can be configured in **forecast only** mode so that you can evaluate the forecast before predictive scaling starts actively scaling capacity. You can view the forecast and recent metric data from CloudWatch in graph form from the Amazon EC2 Auto Scaling console.

It's generally a good idea to observe the predictions of your scaling policy and verify you have selected the correct metric. If you need to adjust the policy you can do so, or you can create a new one and watch both. When you are ready to start scaling with predictive scaling, you can switch the policy from **forecast only** mode to **forecast and scale** mode. After you switch to **forecast and scale** mode, your Auto Scaling group starts scaling based on the forecast.

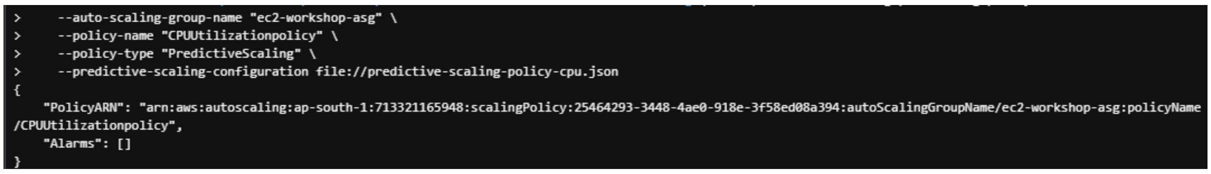
In this step, you are going to configure predictive scaling in **forecast only** mode for your Auto Scaling group.

1. In the Cloud9 terminal, run this command to create a simple configuration file for a predictive scaling policy based on a predefined metrics pair.



```
> {
>     "MetricSpecifications": [
>         {
>             "TargetValue": 50,
>             "PredefinedMetricPairSpecification": {
>                 "PredefinedMetricType": "ASGCPUUtilization"
>             }
>         }
>     ],
>     "Mode": "ForecastOnly"
> }
> EoF
```

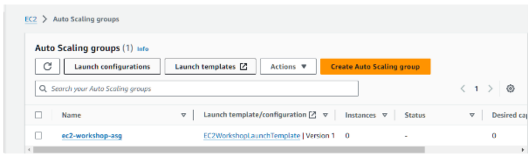
2. Create the predictive scaling policy and attach it to the Auto Scaling group.



```
>     --auto-scaling-group-name "ec2-workshop-asg" \
>     --policy-name "CPUUtilizationpolicy" \
>     --policy-type "PredictiveScaling" \
>     --predictive-scaling-configuration file://predictive-scaling-policy-cpu.json
{
    "PolicyARN": "arn:aws:autoscaling:ap-south-1:713321165948:scalingPolicy:25464293-3448-4ae0-918e-3f58ed08a394:autoScalingGroupName/ec2-workshop-asg:policyName
/CPUUtilizationpolicy",
    "Alarms": []
}
```

Verify predictive scaling policy in AWS Console

1. Navigate to the [Auto Scaling console](#) , click on Auto Scaling group **ec2-workshop-asg**
2. Click on the tab **'Automatic scaling'.**

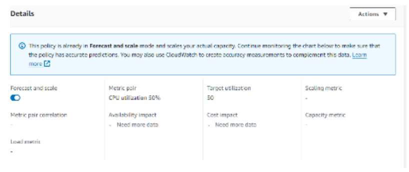

EC2 > Auto Scaling groups

Auto Scaling groups (1) Info

Launch configurations | Launch templates | Actions | Create Auto Scaling group

Search your Auto Scaling groups

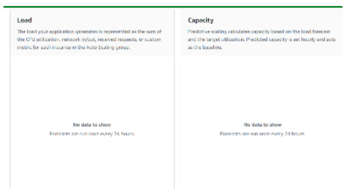
	Name	Launch template/configuration	Instances	Status	Desired ca
☐	ec2-workshop-asg	EC2WorkshopLaunchTemplate | Version 1	0	-	0


Details

Actions

This policy is already in **Forecast and scale** mode and scales your actual capacity. Continue monitoring the chart below to make sure that the policy has accurate predictions. You may also use CloudWatch to create accuracy measurements to complement this data. Learn more

Forecast and scale	Metric pair CPU utilization 50%	Target utilization 50	Scaling metric -
Metric pair correlation -	Availability impact - Need more data	Cost impact - Need more data	Capacity metric -
Load metric -			


Load

The load your application generates is represented as the sum of the CPU utilization, network in/out, received requests, or custom metric for each instance in the Auto Scaling group.

No data to show

Forecasts are run once every 24 hours.

Capacity

Predictive scaling calculates capacity based on the load forecast and the target utilization. Predicted capacity is set hourly and acts as the baseline.

No data to show

Forecasts are run once every 24 hours.

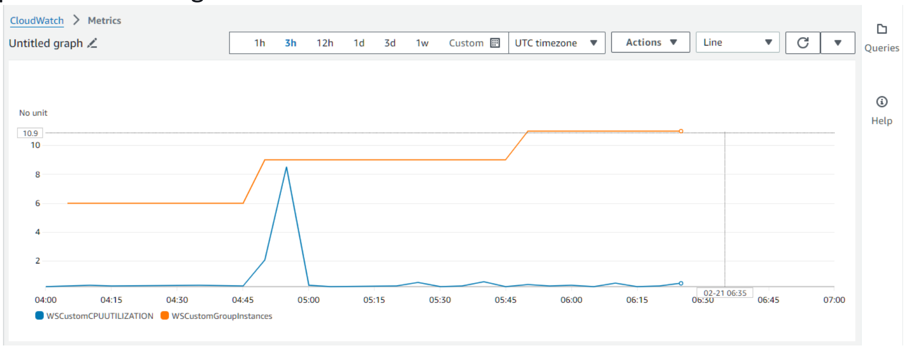
Working with Custom Metrics

As mentioned in the previous chapter, you wouldn't normally have 24 hours data for predictive scaling to start forecasting immediately after creating the policy. For the benefit of this lab, as part of the included CloudFormation stack, a script has been executed to create and populate two CloudWatch custom metrics which can be used in creating the predictive scaling policy.

Verify in CloudWatch metrics using AWS Console

Verify scaling and load metrics data in CloudWatch.

1. Navigate to [Amazon CloudWatch Console](#) .
2. Make sure the correct region is selected in the AWS Console.
3. From left side navigation, click on **'Metrics'** then **'All metrics'**.
4. In the **'Browse'** tab select **'EC2 Workshop Custom Metrics'** under Custom namespaces
5. Select **'AutoScalingGroupName'**, then select the two metrics attached with **ec2-workshop-asg**. This should add the metrics to the graph.
6. To view all metrics data, from the time window filter select **'3d'** to view data of the last 3 days
7. Note the workload pattern in the custom metrics graph, this makes it a good use case for predictive scaling.

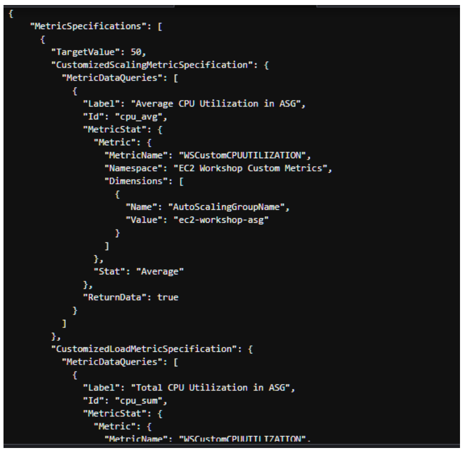


Create a Policy

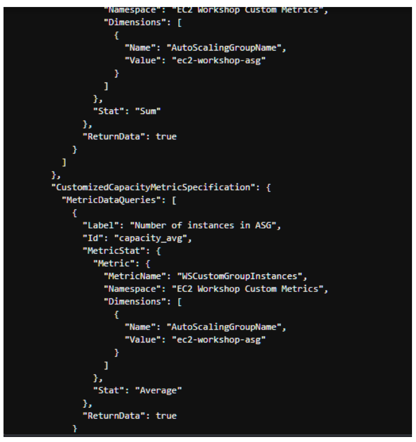
Following up with your scenario, one of the requirements is to reduce the amount of time the application takes to become ready.

Create the predictive scaling policy

1. In the Cloud9 terminal, check you’re in the directory **ec2-spot-workshops/workshops/efficient-and-resilient-ec2-auto-scaling**
2. Review the policy configuration file and note how the custom metrics have been used in it.
   cat ./policy-config.json

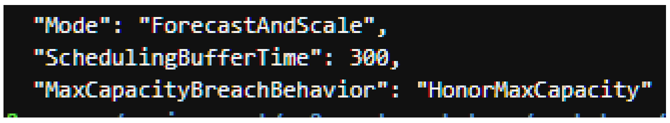


```
{
  "MetricSpecifications": [
    {
      "TargetValue": 50,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Label": "Average CPU Utilization in ASG",
            "Id": "cpu_avg",
            "MetricStat": {
              "Metric": {
                "MetricName": "WSCustomCPUUTILIZATION",
                "Namespace": "EC2 Workshop Custom Metrics",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "ec2-workshop-asg"
                  }
                ]
              },
              "Stat": "Average"
            },
            "ReturnData": true
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Label": "Total CPU Utilization in ASG",
            "Id": "cpu_sum",
            "MetricStat": {
              "Metric": {
                "MetricName": "WSCustomCPUUTILIZATION",
```



```
                "Namespace": "EC2 Workshop Custom Metrics",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "ec2-workshop-asg"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": true
          }
        ]
      },
      "CustomizedCapacityMetricSpecification": {
        "MetricDataQueries": [
          {
            "Label": "Number of instances in ASG",
            "Id": "capacity_avg",
            "MetricStat": {
              "Metric": {
                "MetricName": "WSCustomGroupInstances",
                "Namespace": "EC2 Workshop Custom Metrics",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "ec2-workshop-asg"
                  }
                ]
              },
              "Stat": "Average"
            },
            "ReturnData": true
          }
```

   Because predictive scaling may calculate a need for capacity that exceeds the defined limit of your Auto Scaling group, there is a parameter called **MaxCapacityBreachBehavior**. It is included, along with a value, in the snippet below.



```
  "Mode": "ForecastAndScale",
  "SchedulingBufferTime": 300,
  "MaxCapacityBreachBehavior": "HonorMaxCapacity"
```

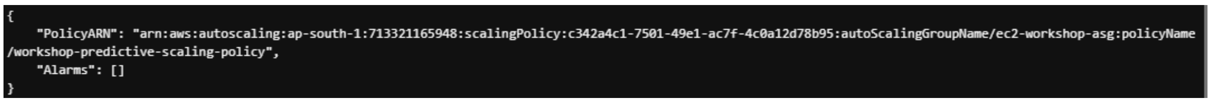
The **MaxCapacityBreachBehavior** parameter defines the behavior of the Auto Scaling group when predicted capacity is greater than the ASG's maximum capacity. It’s currently set to **HonorMaxCapacity** to enforce the ASG's maximum capacity as a hard limit. You can also set it to **IncreaseMaxCapacity** to allow exceeding the ASG's maximum capacity with an upper limit that can be set by another parameter called **MaxCapacityBuffer**. Note that this is an update to the ASG configuration.

3. Run this command to create the policy with the custom metrics and attach it to the Auto Scaling group.

   **aws autoscaling put-scaling-policy --policy-name workshop-predictive-scaling-policy \
     --auto-scaling-group-name "ec2-workshop-asg" --policy-type PredictiveScaling **

--predictive-scaling-configuration file://policy-config.json

If successful, the command should return the created policy ARN

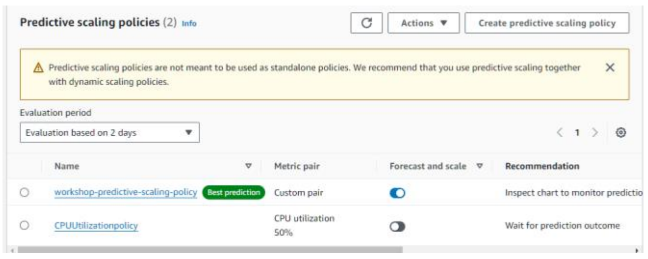


```
{
    "PolicyARN": "arn:aws:autoscaling:ap-south-1:713321165948:scalingPolicy:c342a4c1-7501-49e1-ac7f-4c0a12d78b95:autoScalingGroupName/ec2-workshop-asg:policyName
/workshop-predictive-scaling-policy",
    "Alarms": []
}
```

Review the Outcome

Verify predictive scaling policy in AWS Console

1. Navigate to the Auto Scaling console , click on Auto Scaling group **ec2-workshop-asg**
2. Click on the tab **'Automatic scaling'**
3. A new policy has been created under **'Predictive scaling policies'**

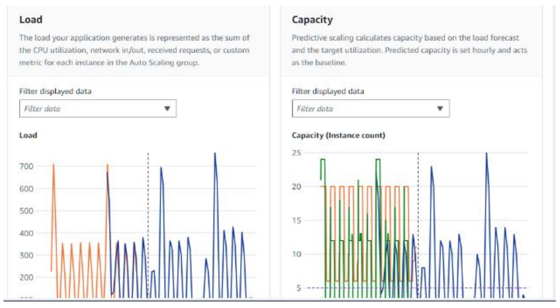


Predictive scaling policies (2) Info

Predictive scaling policies are not meant to be used as standalone policies. We recommend that you use predictive scaling together with dynamic scaling policies.

Evaluation period: Evaluation based on 2 days

	Name	Metric pair	Forecast and scale	Recommendation
○	workshop-predictive-scaling-policy Best prediction	Custom pair	on	Inspect chart to monitor predictio
○	CPUUtilizationpolicy	CPU utilization 50%	off	Wait for prediction outcome



Load

The load your application generates is represented as the sum of the CPU utilization, network in/out, received requests, or custom metric for each instance in the Auto Scaling group.

Capacity

Predictive scaling calculates capacity based on the load forecast and the target utilization. Predicted capacity is set hourly and acts as the baseline.

At the head of next hour, predictive scaling is forecasting capacity of 5 instances

How predictive scaling affects an Auto Scaling group's target capacity

Using the forecast from predictive scaling, Amazon EC2 Auto Scaling scales the number of instances at the beginning of each hour (or before, if configured). Here are the different behaviors you can encounter:

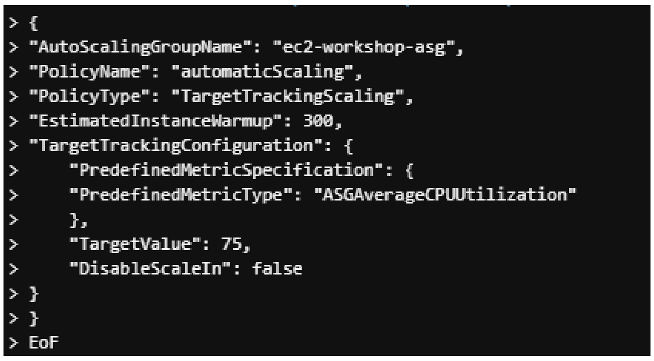
Dynamic Scaling

With predictive scaling, EC2 Auto Scaling does not scale in when the predicted capacity is lower than the actual capacity. Due to this behavior, you need to combine predictive scaling with another scaling policy to reduce capacity when it’s not needed.

Dynamic scaling is used to automatically scale capacity in response to real-time changes in resource utilization. Using it with predictive scaling helps you follow the demand curve for your application closely, scaling in during periods of low traffic and scaling out when traffic is higher than expected to react to demand changes or spikes that were not forecasted by predictive scaling.

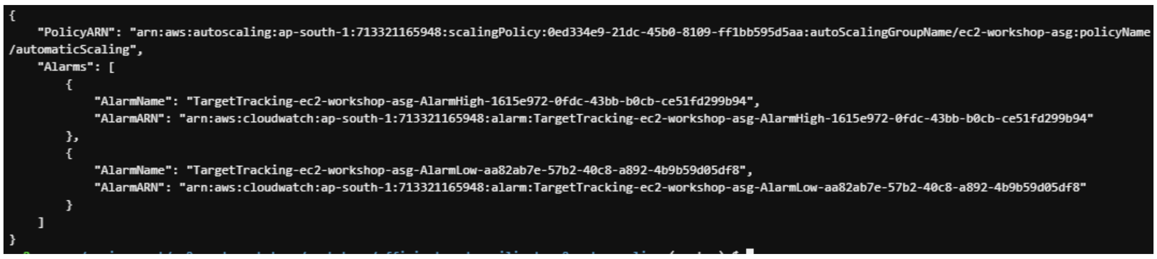
In this section, you will configure the Auto Scaling group to automatically scale out and scale in as your application load fluctuates. You will use the average CPU utilization across the Auto Scaling group instances to accomplish this.

1. Review this command to understand the options, then run it



```
> {
> "AutoScalingGroupName": "ec2-workshop-asg",
> "PolicyName": "automaticScaling",
> "PolicyType": "TargetTrackingScaling",
> "EstimatedInstanceWarmup": 300,
> "TargetTrackingConfiguration": {
>     "PredefinedMetricSpecification": {
>     "PredefinedMetricType": "ASGAverageCPUUtilization"
>     },
>     "TargetValue": 75,
>     "DisableScaleIn": false
> }
> }
> EoF
```

2. Apply the scaling policy:
   aws autoscaling put-scaling-policy --cli-input-json file://asg-automatic-scaling.json
   In response, you should receive a policy ARN and target tracking alarms that



```
{
    "PolicyARN": "arn:aws:autoscaling:ap-south-1:713321165948:scalingPolicy:0ed334e9-21dc-45b0-8109-ff1bb595d5aa:autoScalingGroupName/ec2-workshop-asg:policyName
/automaticScaling",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-ec2-workshop-asg-AlarmHigh-1615e972-0fdc-43bb-b0cb-ce51fd299b94",
            "AlarmARN": "arn:aws:cloudwatch:ap-south-1:713321165948:alarm:TargetTracking-ec2-workshop-asg-AlarmHigh-1615e972-0fdc-43bb-b0cb-ce51fd299b94"
        },
        {
            "AlarmName": "TargetTracking-ec2-workshop-asg-AlarmLow-aa82ab7e-57b2-40c8-a892-4b9b59d05df8",
            "AlarmARN": "arn:aws:cloudwatch:ap-south-1:713321165948:alarm:TargetTracking-ec2-workshop-asg-AlarmLow-aa82ab7e-57b2-40c8-a892-4b9b59d05df8"
        }
    ]
}
```

3. Navigate to the Auto Scaling console and check out your newly created scaling policy in the Scaling Policies tab.
4. The dynamic scaling policy reduces the capacity at the times with low demand using the same average CPU utilization metric that predictive scaling uses.

Because predictive scaling starts launching instances only at the beginning of each hour, you have to wait for the end of the hour to see dynamic scaling in action alongside predictive scaling. If you are learning on your own, grab a snack and watch it happen. If, however, you're at an AWS event it is probably best to proceed.

Congratulations, you now have your Auto Scaling group configured with both:

- a predictive scaling policy in place to ensure application responsiveness ahead of the times of increased demand
- a dynamic policy to
  - scale in and reduce costs when demand is low
  - scale out if the load exceeds prediction

Instance Lifecycles

You have successfully configured scaling policies in your Auto Scaling Group. You still need to solve the challenge of instances taking a long time, as much as 5-10 minutes, to start up. To elaborate the challenge further, the application provided to you has a long warm up time (10 minutes) and you have attempted to work with the application owners to speed up boot

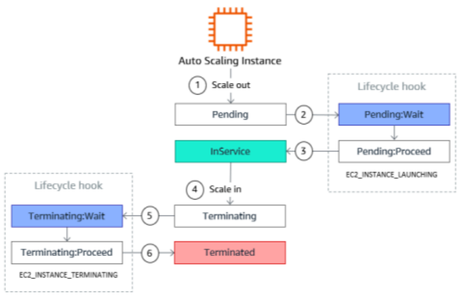
strapping and instance warm up time. While there has been some improvement, it is not possible to speed this process up further without a complete re-architecture of the application. In the following chapters, you will explore how you can fix the challenge of slow boot strapping times with warm pools . With warm pools you can pre-initialize instances and store them in a cache, ready for scale-out when needed. This approach significantly reduces the overall scale-out time.
Before you start working with warm pools, you explore **lifecycle hooks**, a feature that enables a level of control of instance lifecycles.

Lifecycle hooks

Amazon EC2 Auto Scaling offers the ability to add lifecycle hooks to your Auto Scaling groups. These hooks let you create solutions that are aware of events in the Auto Scaling instance lifecycle, and then perform a custom action on instances when the corresponding lifecycle event occurs.

How it works

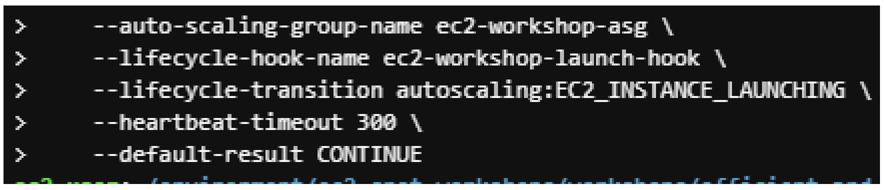
An Amazon EC2 instance transitions through different states from the time it launches until it is terminated. You can create lifecycle hooks to act when an instance transitions into a wait state.



Create a Lifecycle hook for instance launching

When a scale-out event occurs, your newly launched instance completes its startup sequence and transitions to a wait state. While the instance is in the **Pending:Wait** state, it can do many things, including running a script to download and install necessary software packages for your application, making sure that your instance is fully ready before it starts receiving traffic. When the script is finished installing software (or whatever your particular needs might be), it sends a command to EC2 to move into a **Pending:Proceed** state, and ultimately an **InService** state. In our example application, this logic is scripted in the instance userdata which is configured in the launch template.

Run this command to create a lifecycle hook at instance launch. Instances can only remain in a wait state for a finite period of time, so here we will set it to 300 seconds using **heartbeat-timeout** parameter. When the heartbeat times out you can set the default action to **ABANDON** to terminate the instance and launch a new one, or **CONTINUE** to launch the instance anyway.



```
>     --auto-scaling-group-name ec2-workshop-asg \
>     --lifecycle-hook-name ec2-workshop-launch-hook \
>     --lifecycle-transition autoscaling:EC2_INSTANCE_LAUNCHING \
>     --heartbeat-timeout 300 \
>     --default-result CONTINUE
```

Measure Launch Speed

In this section, you will launch an instance directly into the Auto Scaling group you created, **ec2-workshop-asg**

As a part of the initial set up with CloudFormation, a launch template was created with a user data script. This script is executed on the instance during startup. In our example, it installs and starts an application. Once the application is installed, another command is executed to complete the lifecycle action and allow the instance to transition to the next lifecycle step.

Pausing scaling before next activity

To ensure that no instances are added or removed by the Auto Scaling policies you created earlier, in this step you will pause the predictive scaling and reset desired capacity to 0.

1. In AWS Console window, navigate to EC2 Auto Scaling .
2. Click on tab **'Automatic scaling'**
3. Under **'Predictive scaling policies'**, find policy **workshop-predictive-scaling-policy** and toggle off **'Forecast and scale'**.
4. Click **'Switch off Forecast and scale'** in the confirmation message.
5. To ensure the accuracy of the next steps, you need to stop instances that are already running. In the Cloud9 terminal, run this command to ensure current capacity is set to 0 instances and wait a moment for the instances to start shutting down.
   ```
   aws autoscaling set-desired-capacity --auto-scaling-group-name "ec2-workshop-asg" --desired-capacity 0
   ```

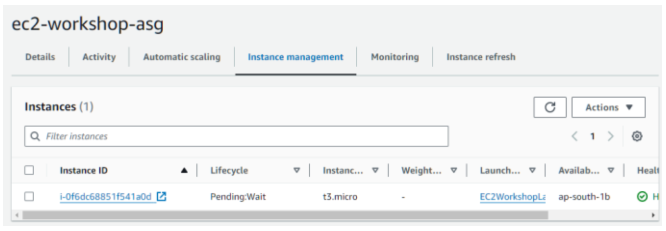
Increase desired capacity

Set the desired capacity of the Auto Scaling group to 1 to launch an instance directly into the Auto Scaling group.

```
aws autoscaling set-desired-capacity --auto-scaling-group-name "ec2-workshop-asg" --desired-capacity 1
```

Watch the instance being launched in the AWS Console

1. Navigate to the Auto Scaling console, click on Auto Scaling group **ec2-workshop-asg**
2. Click on the **'Instance management'** tab.



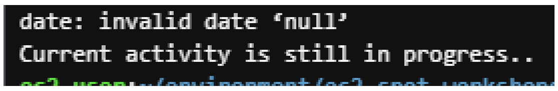
Instances (1)

	Instance ID	Lifecycle	Instanc...	Weight...	Launch...	Availab...	Healt
	i-0f6dc68851f541a0d	Pending:Wait	t3.micro	-	EC2WorkshopLa	ap-south-1b	H

Measure launch speed

You need to wait a few minutes for the instance to be launched by the previous step. Then run this script to measure the launch speed of the instance.

```
activities=$(aws autoscaling describe-scaling-activities --auto-scaling-group-name "ec2-workshop-asg" | jq -r '.Activities[0]') && \

start_time=$(date -d "$(echo $activities | jq -r '.StartTime')" "+%s") && \

end_time=$(date -d "$(echo $activities | jq -r '.EndTime')" "+%s") && \

activity=$(echo $activities | jq -r '.Description') && \

echo $activity Duration: $(($end_time - $start_time))"s" || echo "Current activity is still in progress.."
```

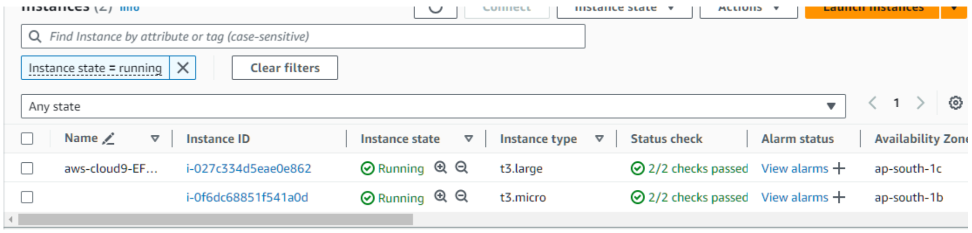


```
date: invalid date ‘null’
Current activity is still in progress..
```

Observe launch duration

Because the instance launched directly into the Auto Scaling group, all initialization actions needed to complete to prepare the instance to be placed in-service. From the results below you can see that these actions took a long time to complete, delaying how quickly your Auto Scaling group can scale and increases the application response time.

Launching a new EC2 instance: i-075fa0ad6a018cdfc Duration: 243s



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
☐	aws-cloud9-EF...	i-027c334d5eae0e862	Running	t3.large	2/2 checks passed	View alarms +	ap-south-1c
☐		i-0f6dc68851f541a0d	Running	t3.micro	2/2 checks passed	View alarms +	ap-south-1b

Warm Pools

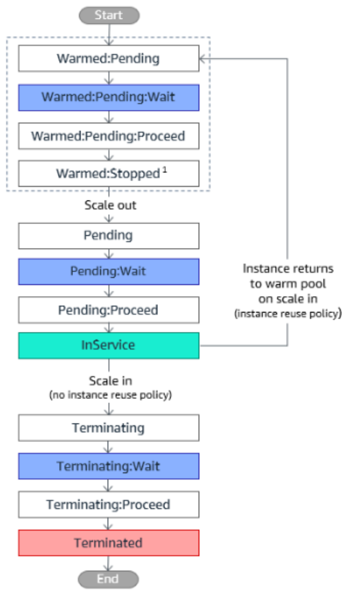
A warm pool is a pool of pre-initialized EC2 instances that sits alongside an Auto Scaling group. Whenever your application needs to scale out, warm pools give you the ability to decrease the time for new instances to join the Auto Scaling group. They are beneficial for applications that have long boot times.

Planning your warm pool

It’s important to decide how to configure a warm pool as it can make a difference in associated cost. You can do this by defining a minimum and a maximum boundary for the warm pool:

1. By default, and in this step, warm pool size is the difference between the number of currently running instances and the maximum capacity.
2. However, if you’re running a large Auto Scaling group that might not be the best cost optimized option. As a cost control, you can set a maximum prepared capacity which counts current running instances as part of it.
3. You can also set a static number as the minimum pool size to ensure that there are always at least a certain number of warmed instances *(regardless of the number of running instances)* available to react to demand spikes.

You can keep instances in the warm pool in one of three states: stopped, running, or hibernated *(if supported by the instance type)*. Keeping instances in a stopped state is an effective way to minimize costs; with stopped instances, you pay only for the EBS volumes and the Elastic IP addresses attached to the instances.
By default, when ASG scales in, instances get terminated. In warm pools, however, you can configure the reuse policy to return instances to the warm pool when the Auto Scaling group scales in.



Add a warm pool to your Auto Scaling group

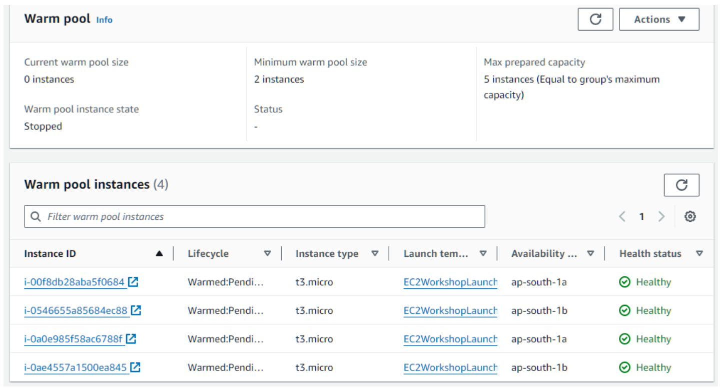
In this step, you will add a warm pool to your Auto Scaling group to pre-initialize your instances, so that instances can be brought into service more rapidly.

You decide that you want to keep your warm pool instances in a stopped state after they have completed their initialization actions. You also decide that you want some number of instances in the warm pool at all times, and as many as possible based on the maximum size of the Auto Scaling group. To do this, you will set the optional warm pool sizing parameters **--min-size** to 2 and leave **--max-group-prepared-capacity** empty. This means that this warm pool has a minimum size of 2 and a maximum prepared capacity equal to the maximum size of the Auto Scaling group. The maximum prepared capacity includes instances launched into the Auto Scaling group, and instances launched into the warm pool.

aws autoscaling put-warm-pool --auto-scaling-group-name "ec2-workshop-asg" --pool-state Stopped --min-size 2

Now let’s check the warm pool in the AWS Console

1. Navigate to the Auto Scaling console , click on Auto Scaling group **ec2-workshop-asg**
2. Click on tab **'Instance management'**
3. A warm pool has been created and the instances have started initializing, note the current lifecycle for the instances is **Warmed:Pending:Wait**
4. Once the instances are initialized, the lifecycle will be changed to **Warmed:Stopped**



Warm pool Info

Current warm pool size	Minimum warm pool size	Max prepared capacity
0 instances	2 instances	5 instances (Equal to group's maximum capacity)
Warm pool instance state	Status	
Stopped	-	

Warm pool instances (4)

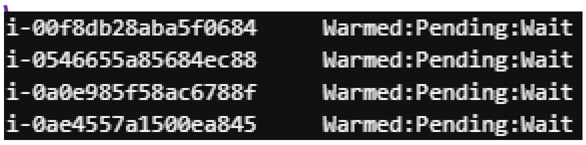
Instance ID	Lifecycle	Instance type	Launch tem...	Availability ...	Health status
i-00f8db28aba5f0684	Warmed:Pendi...	t3.micro	EC2WorkshopLaunch	ap-south-1a	✓ Healthy
i-0546655a85684ec88	Warmed:Pendi...	t3.micro	EC2WorkshopLaunch	ap-south-1b	✓ Healthy
i-0a0e985f58ac6788f	Warmed:Pendi...	t3.micro	EC2WorkshopLaunch	ap-south-1a	✓ Healthy
i-0ae4557a1500ea845	Warmed:Pendi...	t3.micro	EC2WorkshopLaunch	ap-south-1b	✓ Healthy

How Warm Pools Work

You can also use the AWS CLI to observe changes in the warm pool at any state of the instances lifecycle. Run this command to list all instances with their state in the warm pool now.

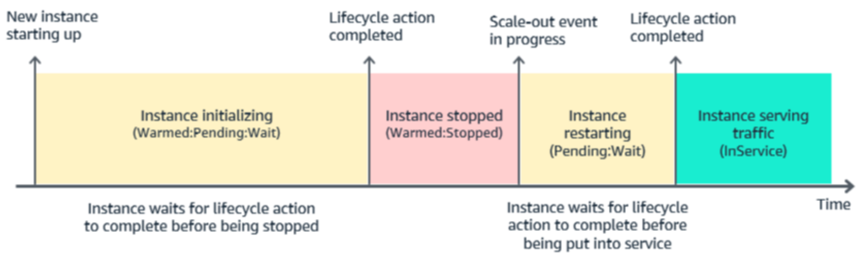
```
aws autoscaling describe-warm-pool --auto-scaling-group-name "ec2-workshop-asg" | jq -r '.Instances[]| "\(.InstanceId)\t\(.LifecycleState)"'
```

You can see that multiple instances were launched into the warm pool. The number of instances is the difference between the number of current running instances and the Auto Scaling group max capacity. Since you have one instance already in service and the Auto Scaling group's maximum size is five, four instances were launched into the warm pool.



```
i-00f8db28aba5f0684     Warmed:Pending:Wait
i-0546655a85684ec88     Warmed:Pending:Wait
i-0a0e985f58ac6788f     Warmed:Pending:Wait
i-0ae4557a1500ea845     Warmed:Pending:Wait
```

When an instance is launched into a warm pool it will transition through lifecycle states, starting with **Warmed:Pending**.
If a lifecycle hook is configured, the instance will move to a **Warmed:Pending:Wait** state until initialization actions are completed.
After initialization actions are completed, and the lifecycle hook sends a **CONTINUE** signal, the instance will move to a **Warmed:Pending:Proceed** state.
Since you configured instances in your warm pool to be stopped after initialization, the instance launch will complete with the instance in a **Warmed:Stopped** state. The instance is now pre-initialized and ready to be launched into the Auto Scaling group as additional capacity is needed.

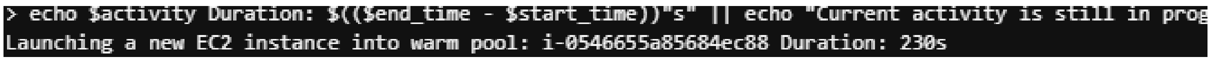

New instance starting up
Lifecycle action completed
Scale-out event in progress
Lifecycle action completed
Instance initializing (Warmed:Pending:Wait)
Instance stopped (Warmed:Stopped)
Instance restarting (Pending:Wait)
Instance serving traffic (InService)
Instance waits for lifecycle action to complete before being stopped
Instance waits for lifecycle action to complete before being put into service
Time

Observe launch speed into warm pool

Now let’s see how long it took to launch an instance into the warm pool.

```
activities=$(aws autoscaling describe-scaling-activities --auto-scaling-group-name "ec2-workshop-asg" | jq -r '.Activities[0]') && \

start_time=$(date -d "$(echo $activities | jq -r '.StartTime')" "+%s") && \

end_time=$(date -d "$(echo $activities | jq -r '.EndTime')" "+%s") && \

activity=$(echo $activities | jq -r '.Description') && \

echo $activity Duration: $(($end_time - $start_time))"s" || echo "Current activity is still in progress.."
```

As you can see from the following results, launching an instance into a warm pool took a similar length of time to launching an instance directly into the Auto Scaling group.



```
> echo $activity Duration: $(($end_time - $start_time))"s" || echo "Current activity is still in prog
Launching a new EC2 instance into warm pool: i-0546655a85684ec88 Duration: 230s
```

Launch from Warm Pool

Now that you have pre-initialized instances in the warm pool, you can scale your Auto Scaling group and launch a pre-initialized instance rather than launching a new instance that has not been pre-initialized.

Increase desired capacity

Increase the desired capacity of your Auto Scaling group to 2 instances.

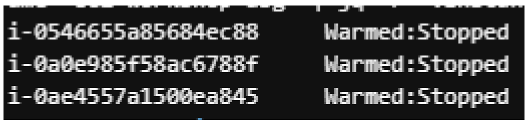
```
aws autoscaling set-desired-capacity --auto-scaling-group-name "ec2-workshop-asg" --desired-capacity 2
```

Observe warm pool change

Now, describe the warm pool and observe changes. As you can see below, the additional instance you just launched is no longer in the warm pool. It was launched from the warm pool into the Auto Scaling group in response to your increase in desired capacity.

```
aws autoscaling describe-warm-pool --auto-scaling-group-name "ec2-workshop-asg" | jq -r '.Instances[]| "\(.InstanceId)\t\(.LifecycleState)"'
```

You can see that multiple instances were launched into the warm pool. The number of instances is the difference between the number of current running instances and the Auto Scaling group max capacity. Since you have one instance already in service and the Auto Scaling group's maximum size is five, four instances were launched into the warm pool



```
i-0546655a85684ec88     Warmed:Stopped
i-0a0e985f58ac6788f     Warmed:Stopped
i-0ae4557a1500ea845     Warmed:Stopped
```

Measure launch speed

Now you can measure how long that instance took to be ready as it moved from the warm pool to the Auto Scaling group.

**activities=$(aws autoscaling describe-scaling-activities --auto-scaling-group-name "ec2-workshop-asg" | jq -r '.Activities[0]') && **

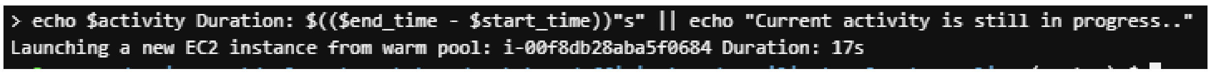
**start_time=$(date -d "$(echo $activities | jq -r '.StartTime')" "+%s") && **

**end_time=$(date -d "$(echo $activities | jq -r '.EndTime')" "+%s") && **

**activity=$(echo $activities | jq -r '.Description') && **

echo $activity Duration: $(($end_time - $start_time))"s" || echo "Current activity is still in progress.."

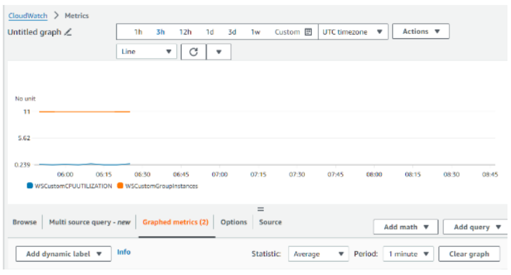
As you can see from the following sample results, because the instance was pre-initialized, the instance launch duration was significantly reduced. This means you can now more rapidly place instances into service in response to load placed on your workload by launching pre-initialized instances from the warm pool, making your application more resilient and responsive to demand spikes.



```
> echo $activity Duration: $(($end_time - $start_time))"s" || echo "Current activity is still in progress.."
Launching a new EC2 instance from warm pool: i-00f8db28aba5f0684 Duration: 17s
```

Since you have enabled detailed CloudWatch metrics collection for the Auto Scaling group, let's compare the capacity metrics before and after enabling warm pools.

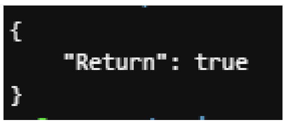
1. Navigate to Amazon CloudWatch Console .
2. From the left side navigation, click on **'Metrics'** then All metrics.
3. In the **'Browse'** tab select **'Auto Scaling'** under **'AWS namespaces'**
4. Select **'Group Metrics'**, then select these two metrics: **GroupDesiredCapacity** and **GroupInServiceCapacity** attached with **ec2-workshop-asg**. This should add the metrics to the graph.
5. Switch to **'Graphed metrics'** tab and change **'Granularity Period'** to 1 minute for both metrics.
6. Note the difference in time between **'DesiredCapacity'** and **'InServiceCapacity'** before and after enabling warm pools.



Cleanup

Delete all manually created resources.

aws autoscaling delete-auto-scaling-group --auto-scaling-group-name ec2-workshop-asg --force-delete



```
{
    "Return": true
}
```

```
aws ec2 delete-key-pair --key-name asgworkshop
```

```
aws cloudformation delete-stack --stack-name $stack_name
```

```
aws cloudformation wait stack-delete-complete --stack-name $stack_name
```

Finished

Congratulations on completing the workshop…or at least giving it a good go! This is the workshop’s permanent home, so feel free to revisit as often as you’d like. In typical Amazon fashion, we’ll be listening to your feedback and iterating to make it better. If you have feedback, we’re all ears!

This is what you have accomplished in this workshop:

1. Learned about the different types of scaling in Amazon EC2 Auto Scaling groups when to use each one.
2. Configured and applied auto scaling policies to increase application efficiency.
3. Learned and used lifecycle hooks and warm pools to improve application resiliency, and making it ready to handle peak and spike demands.

Check the resources on next page to learn more about the topics we discussed..