**Auto Scaling Applications on Pivotal Cloud Foundry**

As modern computing moves more and more workloads to the cloud, it is very important to have high availability and scalable design for applications to handle large amount of workload.

Pivotal Cloud Foundry (PCF) provides functionality to “auto scale” the deployed application seamlessly without manual interventions.

Autoscaling provides the ability to ramp up or ramp down the number of instances of the deployed application based on various parameters also known as ‘auto-scaling rules’.

**Setup App Auto Scaler**

To enable auto-scaling, the application must be bound to the **App Auto Scaler** service. This service can be instantiated from **Apps Manager** User interface or using **CF CLI** (which needs App Auto Scaler plugins for the CLI to be enabled).

Application can be bound to App Auto Scaler service using:

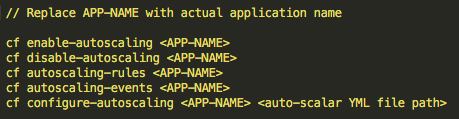
* Apps manager User interface
* CF command-line interface

**Configuring Auto Scaler**

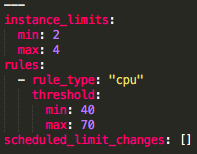
Once the application has been bound to **App Auto Scaler service**, it can be configured using various parameters (i.e. auto scaling rules) .

Configuring the scaling rules can also be achieved through **Apps Manager** user interface or **CF CLI**.

The following are some useful CLI commands for configuring auto-scaling which are self-explanatory:



Similar to the ‘manifest.yml’ file, auto-scaling rules can be maintained in **app auto scaler YML file** as seen below. *We can give any name to this file*.



Here auto-scaling has been configured using the parameter “CPU utilization”. If CPU utilization goes below 40% then CF will scale down the application up to MIN of 2 instances whereas if CPU utilization goes above 70% then CF will scale up the application up to MAX of 4 instances.

Let’s say we have created a YML file with above mentioned scaling health rule as “demoApp-AutoScalar.yml” at the same level where our build file is. Then we can use the below command for configuring auto-scaling for our app named “DemoMyApp”.

*cf configure-autoscaling DemoMyApp demoApp-AutoScalar.yml*

I would highly recommend using YML file configuration as it can be maintained alongside of code base and provides advantages of considering modern style deployment approach like**Blue-Green deployment**, etc.

**How App Auto Scaler Determines When to Scale**

**App Auto scaler** **service** determines whether to ramp up / ramp down application instance or maintain the current number of instance(s) by averaging the values of configured metric for the last 120 seconds.

After every 35 seconds, **App Auto Scaler service** evaluates whether to auto-scale the application or not by following the approach mentioned above.

**App Auto Scaler scales the apps as follows:**

* Increment by one instance when any metric exceeds the **Max**threshold specified
* Decrement by one instance only when all metrics fall below the **Min**threshold specified

**Understanding Auto Scale Health Rule**

The table below lists the metrics that you can use on App Auto Scaler rules:

|  |  |  |
| --- | --- | --- |
| **Metric** | **Description** | **rule\_type** |
| CPU  Utilization | Average CPU utilization for all instances of App | cpu |
| Memory  Utilization | Average memory percentage for all instances of App | memory |
| HTTP  Throughput | Total App requests per second divided number of instances | http\_throughtput |
| HTTP  Latency | Average latency of App response to HTTP request | http\_latency |
| Rabbit MQ Depth | Queue length of the specified queue | rabbitmq |

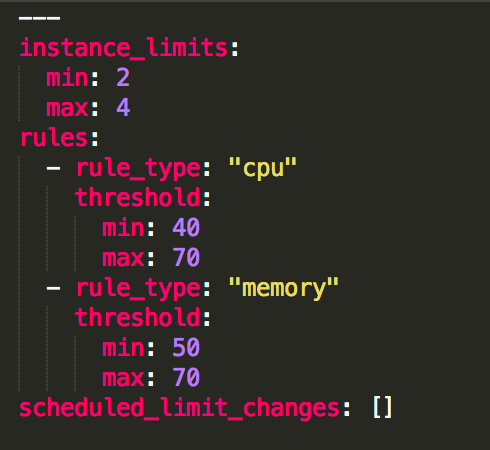
It is very important to understand application performance while applying auto scaling rules on HTTP Throughput and HTTP Latency. The following points should be considered

* Initial number of application instances.
* Performance benchmarking results of the application (to understand at what load, the performance of the application starts to deteriorate) and how many instances are needed to avoid going beyond that load.
* While calculating HTTP Latency time, any backend service/database communication should also be taken into consideration, and if there is any proportional deterioration in backing services, they should be taken into account so as to not to escalate an already deteriorated situation.
* While setting up the rule on HTTP request, we should consider **peak time traffic** coming to application which helps to configure auto-scaling in an efficient manner. Your max instances for autoscaling should also be able to accommodate traffic considering the unavailability of other datacenter your app may be hosted on.

While setting up the Rabbit MQ based auto scaling rule,  **–subtype**is a required field which holds **name of the queue**. For example, as seen below, we can also configure more than 1 rabbitmq queues.

Rabbit-MQ.png

Newer versions of CF also allows to set autoscaling rule based on a combination multiple metrics, such as those identified below:



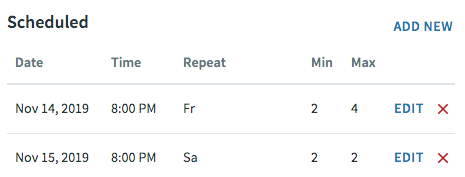
With the recent release of CF, we can also create custom metrics based on which we can configure auto scaling rule for our application.

**Schedule Application Auto Scaler**

It is best to set up auto-scaling with multiple rules to handle rare scenarios, such as an overnight increase in traffic due to holiday seasons like Thanksgiving. These kinds of occurrences can be scheduled ahead of time.

PCF **App Auto Scaler** service provides the functionality to **schedule “auto-scaling”** to handle the rare ‘known’ events which may impact application’s availability/performance.

This can be achieved from **Apps Manager user interface**. For this, go to your deployed application which is bound to **app-auto scalar service** and select **‘Manage Scaling’** and select **‘Schedule Limit Change’**. Below is the sample rule setup:



The above configuration will scale up the application on Nov 14, 2019, at 8 PM and will scale down the application on Nov 15, 2019, at 8 PM.

**Challenges While Configuring Auto Scaling**

**Some of the commands to enable or disable autoscaling from CF CLI may not be supported in future versions of the CLI, so it is best to stick with the Apps Manager or the Autoscaler API for now.**

It is very important while configuring auto-scaling rule that we are selecting correct metrics.

Improper metrics might result in unexpected results.

Consider the following scenario — it may seem like a good idea to scale on http latency since latency or response time seems like a good indicator of when the application is under load and may need to scale.

Typically your app is taking 500 ms to respond. If there is a considerable load on the application, you would expect the response time to go up. But that may not always be true.

Consider you app is under a DDOS attack. Most of the input coming to the app now is invalid and your app processes them in under 20 ms. If there are 1000s of such requests, it will actually bring down the average response time of your app, and your app may actually scale down instead of scale-up.

In such scenarios, it might be better to combine multiple metrics such as CPU, memory, http throughput, http latency or use some custom metric for scaling.

Thus, we have seen that, if used properly, application autoscaling can be an important tool to ensure the reliability and availability of your application.