

## Sprawozdanie

# Badanie wydajności aplikacji hyperflow na nowych typach instancji w chmurze Amazon EC2

### 1. Opis problemu

Celem projektu było zbadanie i porównanie kosztów wywołania zadań opartych na workflowach uruchomionych na różnych typach instancji w chmurze Amazon EC2.

W tym celu wykorzystałem problem Montage:

<https://confluence.pegasus.isi.edu/display/pegasus/Montage+Characterization>, który z wielu plików w formacie Flexible Image Transport System (FITS) tworzy pojedyńczą mozaikę gwiazd.

### 2. Opis uruchomienia

Uruchomienie hyperflowa zostało zaczerpnięte ze strony:

<https://github.com/dice-cyfronet/hyperflow/wiki/TutorialAMQP>

Dodatkowo, do wszystkich obliczeń został wykorzystany storage S3 amazonu, na którym hyperflow umieszczał pliki powstające w trakcie wykonywania programu.

Dla wszystkich przykładów został wykorzystany rozmiar problem o wielkości 2.00, dzięki czemu uzyskałem duże czasy wykonania na różnych maszynach, co pozwoliło mi dokładniej zbadać różnice między nimi

### 3. Wyniki

Wszystkie wyniki zgrupowane dla każdej z instancji zostały umieszczone na repozytorium:

[https://github.com/gmiejski/hyperflow\\_results](https://github.com/gmiejski/hyperflow_results)

Najpierw statystyki każdej z instancji podane przez amazon:

Instance	vCPU	Memory (GiB)	Instance Storage (GB)	Network performance	Cost
T2.mikro	1	1	EBS Only	Low to Moderate	\$0.013 per Hour
T2.small	1	2	EBS Only	Low to Moderate	\$0.026 per Hour
T2.medium	2	4	EBS Only	Low to Moderate	\$0.052 per Hour
M3.medium	1	3.75	1 x 4 SSD	Moderate	\$0.067 per Hour
M3.large	2	7.5	1 x 32 SSD	Moderate	\$0.133 per Hour

Poniżej znajduje się zestawienie łącznych kosztów dla każdej z poniższych instancji:

Instance	Cost Per hour	Time rounded [hours]	Time rounded up [hours]	Cost
T2.mikro	\$0.013	7.58	8	\$0.104
T2.small	\$0.026	7.10	8	\$0.208
T2.medium	\$0.052	3.98	4	\$0.208
M3.medium	\$0.067	5.56	6	\$0.402
M3.large	\$0.133	2.96	3	\$0.399
Double T2.mikro	\$0.026	3.89	4	\$0.104

Z powyższych wyników można wywnioskować, że dodatkowa moc obliczeniowa nie przyspiesza wykonania zadania. Znacznie lepiej wypadają instancję z większą liczbą vCPU, jako że spora część zadań może zostać zrównoleglona.

Postanowiłem również sprawdzić jaki czas jest rzeczywiście wykorzystywany na obliczenia a jaki na transfer danych przez sieć ( czas w fazie stage\_in do czasu :

Instance	% czasu spędziony na przygotowywaniu do wykonania zadania
T2.mikro	86%
T2.small	86%
T2.medium	86%
M3.medium	80%
M3.large	82%

To potwierdza tezę, że rzeczywisty czas wykonywania zadań byłby kilkukrotnie mniejszy gdyby był wykorzystany storage lokalny, zamiast amazonowego S3.

Widać również, że czas przeznaczony na fazę stage in różni się dla instancji T2 oraz w M3, co potwierdza szacunkowe określenie wydajności transferu sieciowego dla tych typów instancji.

#### **4. Statystyki z hyperflow-amqp-metric-collector'a:**

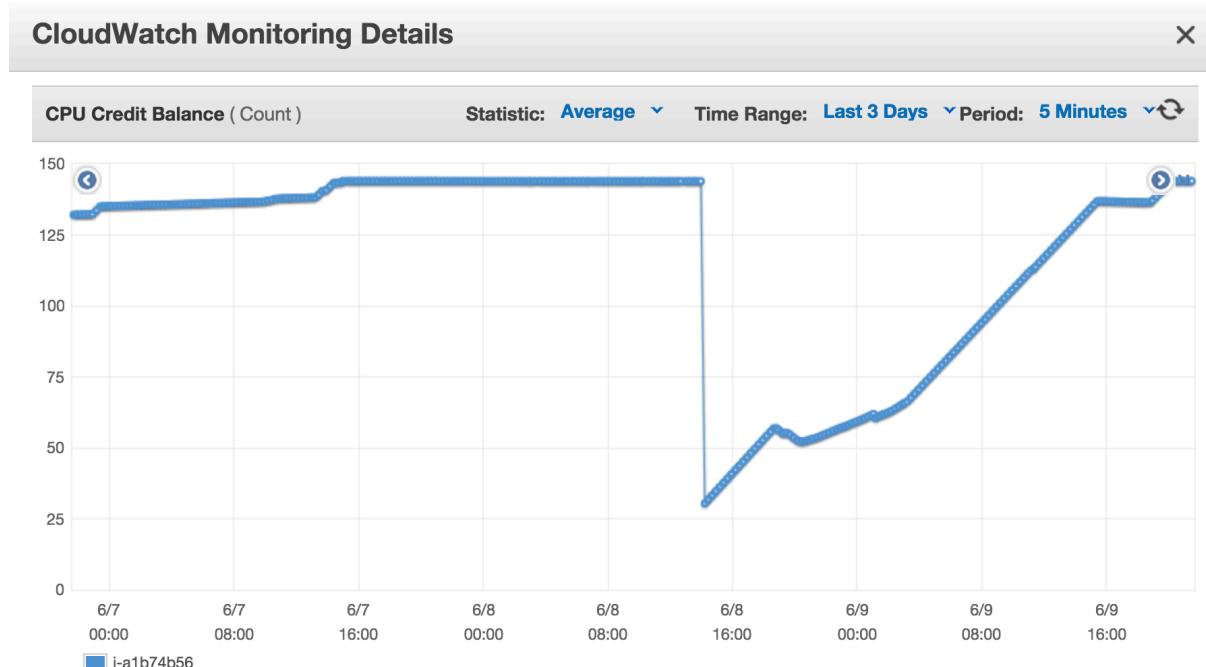
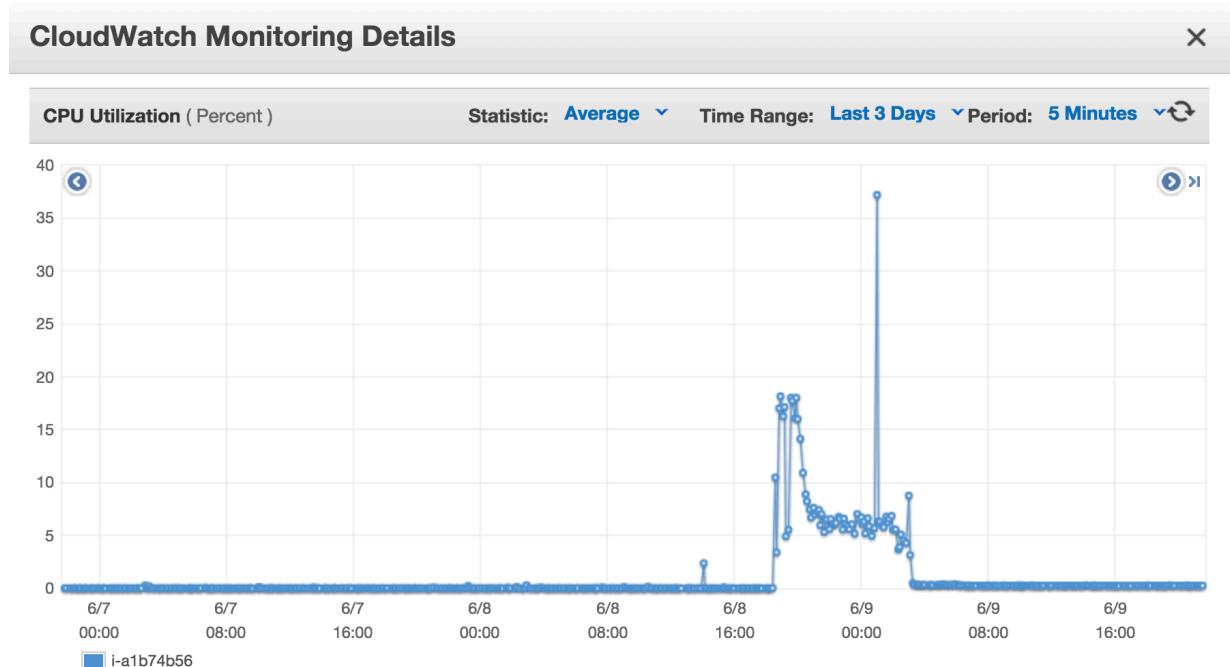
Niestety z powodu kompresji wykresy wstawione do tego sprawozdania tracą na czytelności, dlatego muszę odesłać do pdf-ów zawartych w repozytorium, natomiast tutaj wypiszę tylko wnioski:

- Można zauważyć, że w przypadku 2 vCPU, czas całkowity drastycznie maleje, właściwie dwukrotnie w porównaniu do podobnych instancji.
- Wynika z tego, że spora część zadań może zostać zrównoleglona. Tylko w ostatniej fazie, tuż przed tworzeniem wynikowego JPG kilka zadań nie ma takich właściwości
- Początkowa faza mProjectPP jest bardzo intensywna jeżeli chodzi o CPU, co będzie również potem widoczne na kolejnych wykresach.
- Przy pozostałych zadaniach widać, że większość czasu jest przeznaczana na fazy "przed" oraz "po" danym małym zadaniu. Jest to czas przeznaczony głównie na transfer plików na storage S3

## 5. Statystyki instancji amazon'a - wykorzystanie CPU I kredytów

Poniżej przedstawię kilka zdjęć ściągniętych ze strony amazon'a, który bardzo dokładnie monitoruje wiele aspektów każdej instancji:

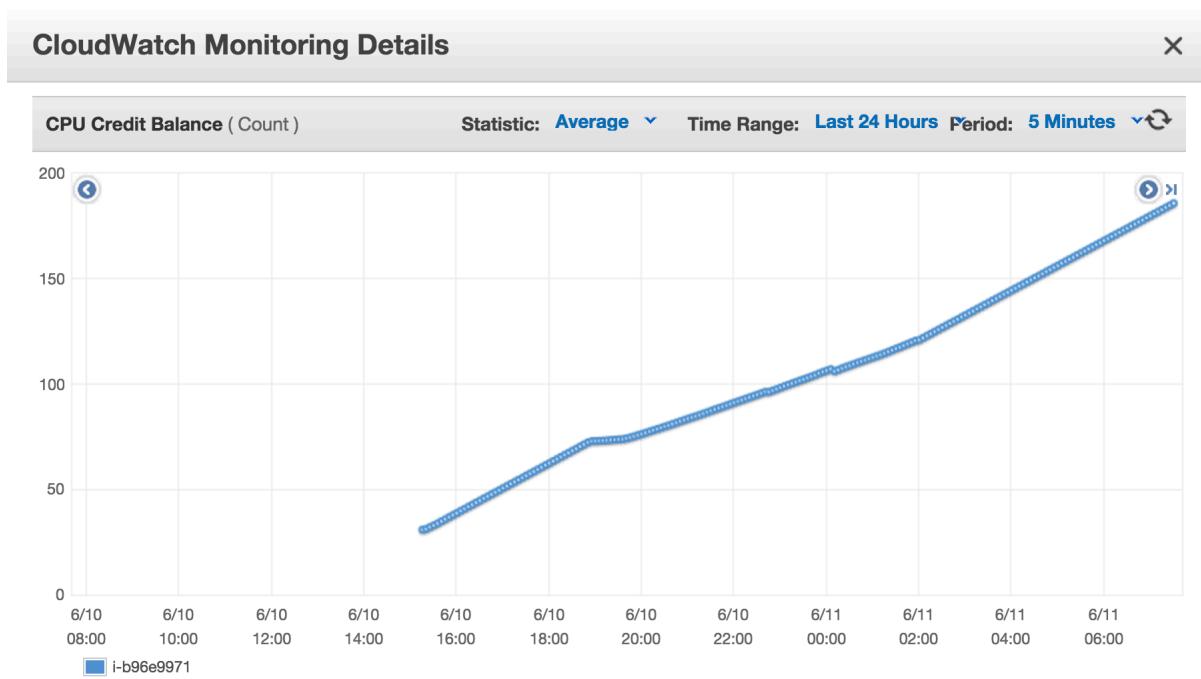
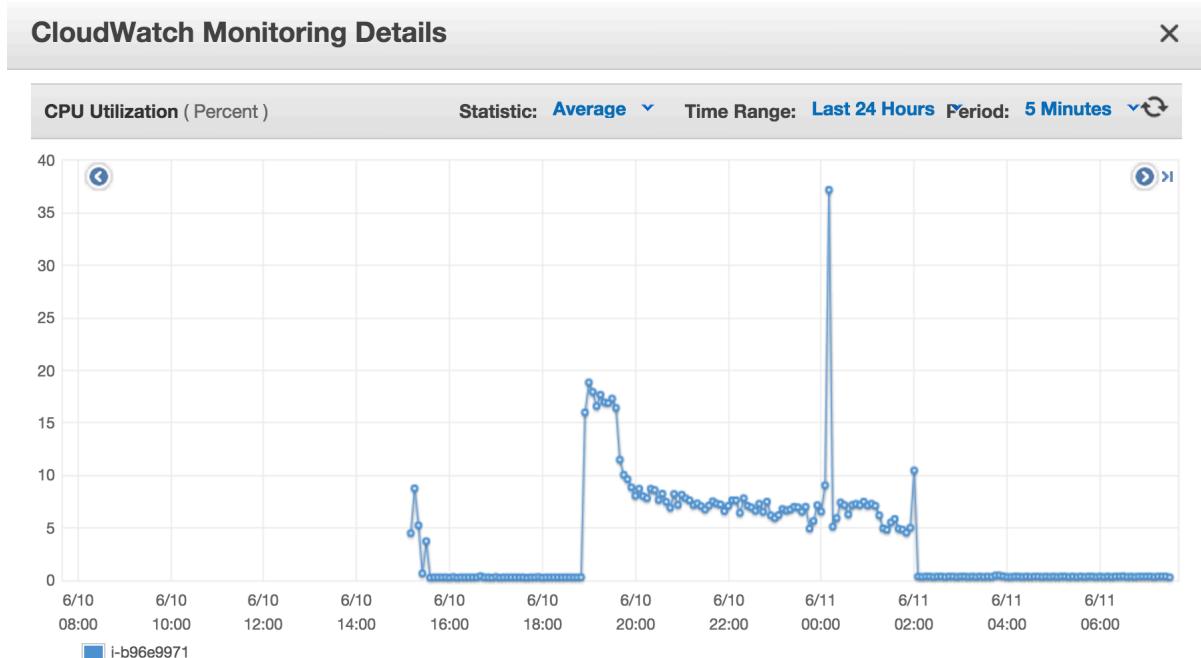
- Wykorzystanie CPU oraz kredytów
  - t2.mikro



Wnioski : widać, że pomimo wykorzystywania kredytów, ciągle zużywamy je w wolniejszym tempie niż są one generowane ( zużycie CPU na poziomie 20%

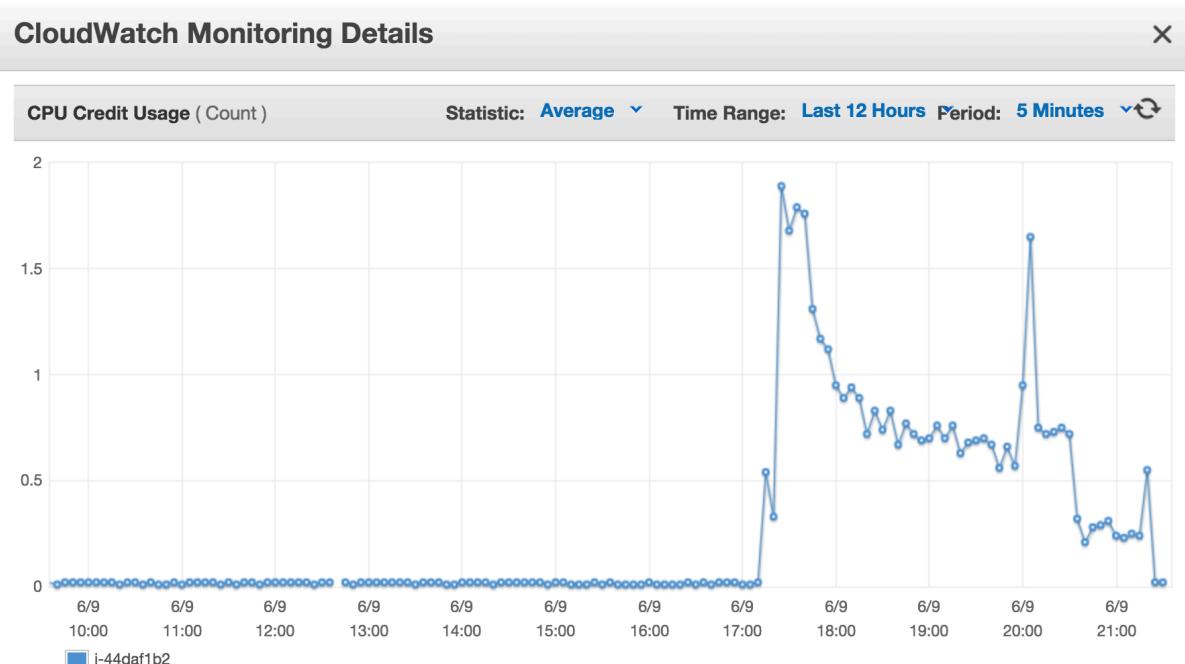
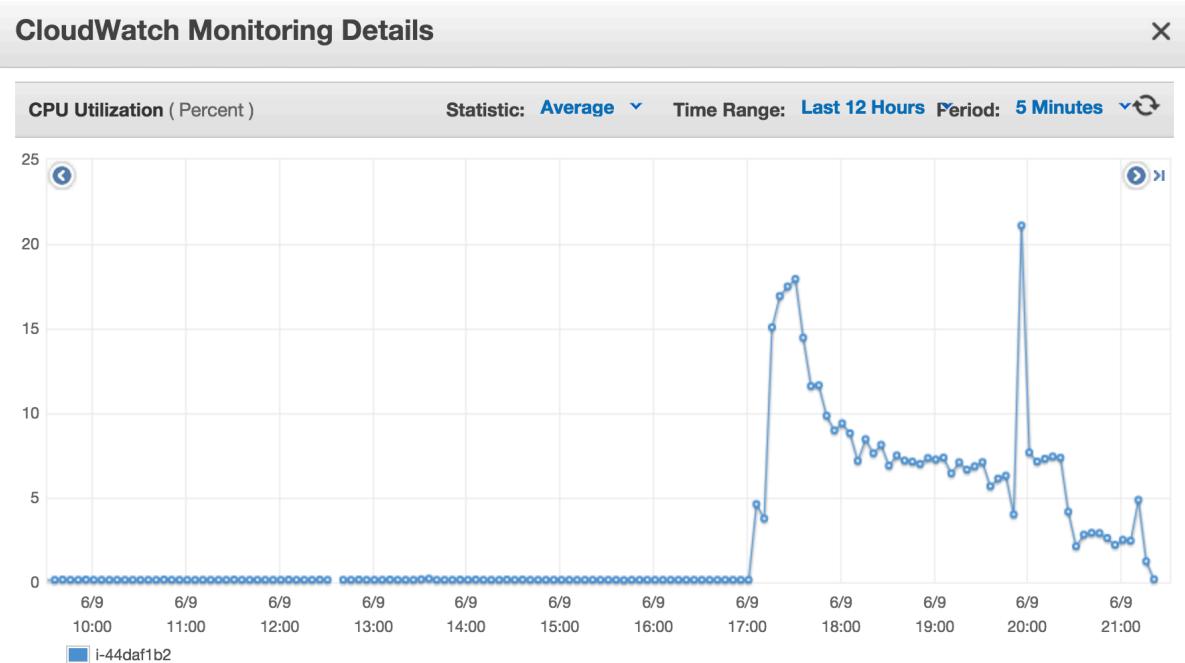
powoduje wyrównanie zużycia do generowania się kredytów ). Oznacza to, że wykorzystując tąinstancję nigdy nie znajdziemy się w sytuacji, gdy moc zostanie bardzo ograniczona przez niewystarczającą ilość kredytów.

- T2.small

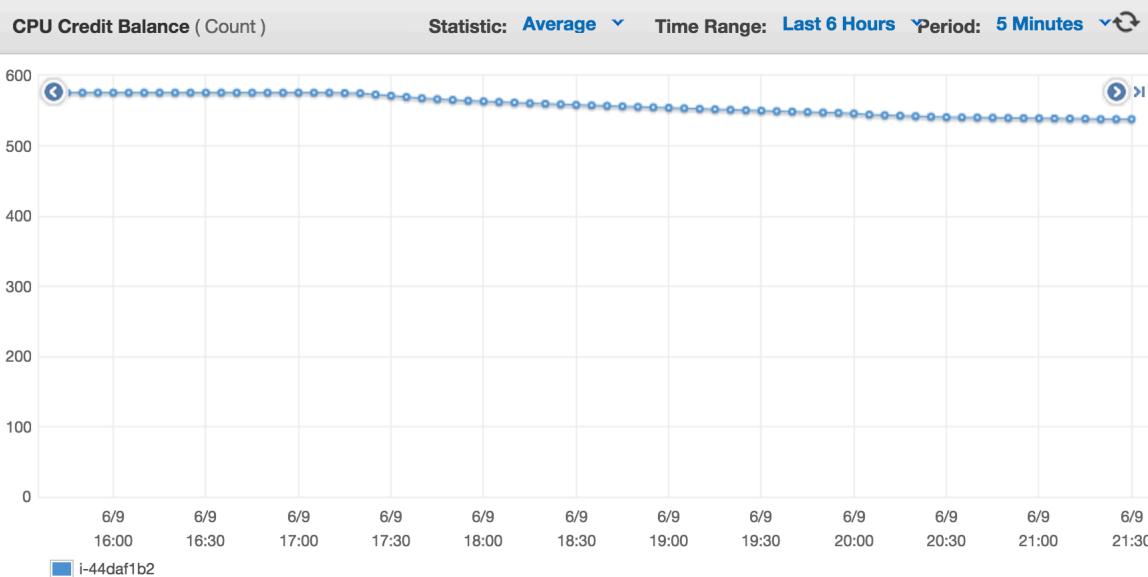


Widać, że oba wykresy w zachowują się podobnie jak dla instancji t2.mikro.

- T2.medium



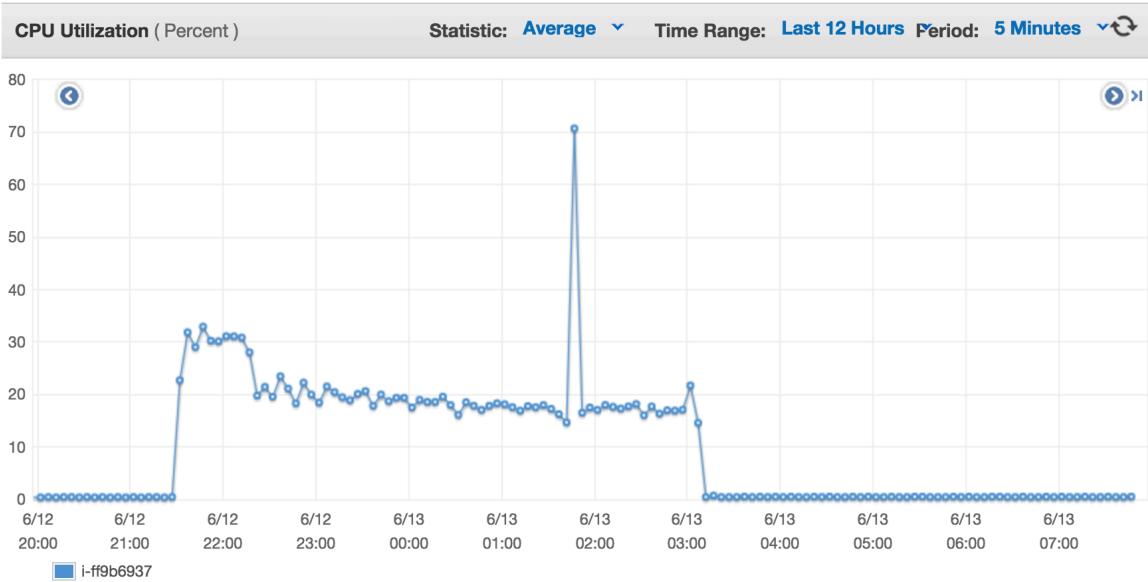
## CloudWatch Monitoring Details



Dla tej instancji można zauważyc, że tempo zużycia kredytów jest większe niż tempo ich generowania.

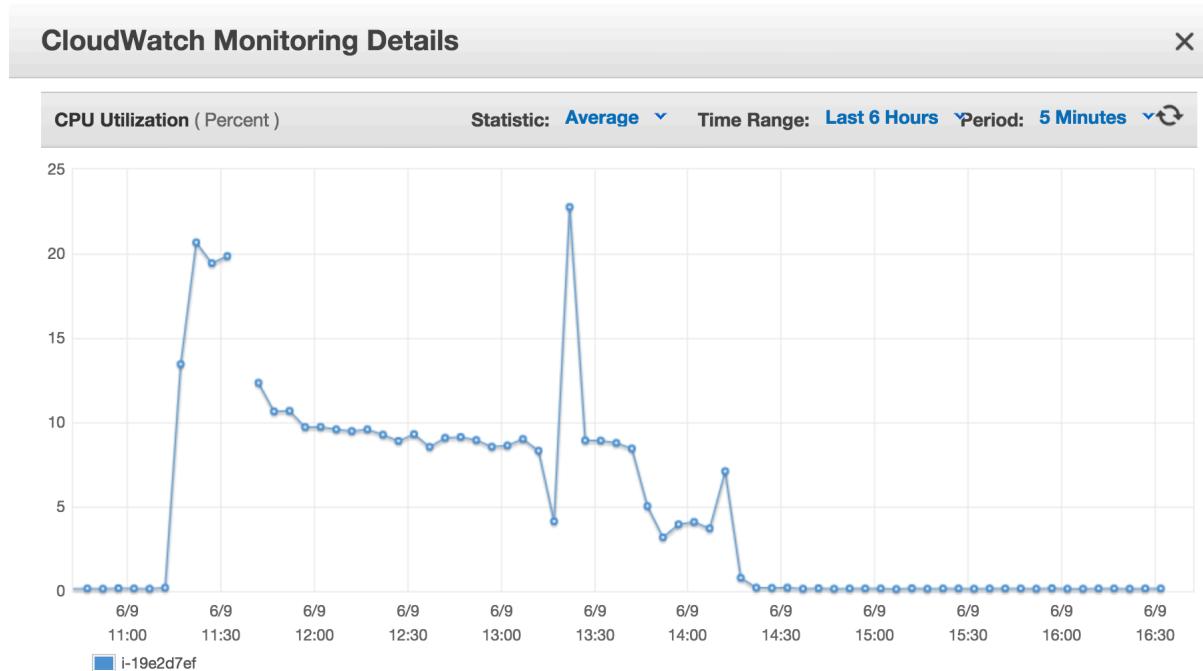
- M3.medium

## CloudWatch Monitoring Details



Tutaj charakterystyka pierwszej instancji nie opartej na kredytach. Widać od razu dużo większe procentowe zużycie CPU w porównaniu do innych instancji podczas całej pracy.

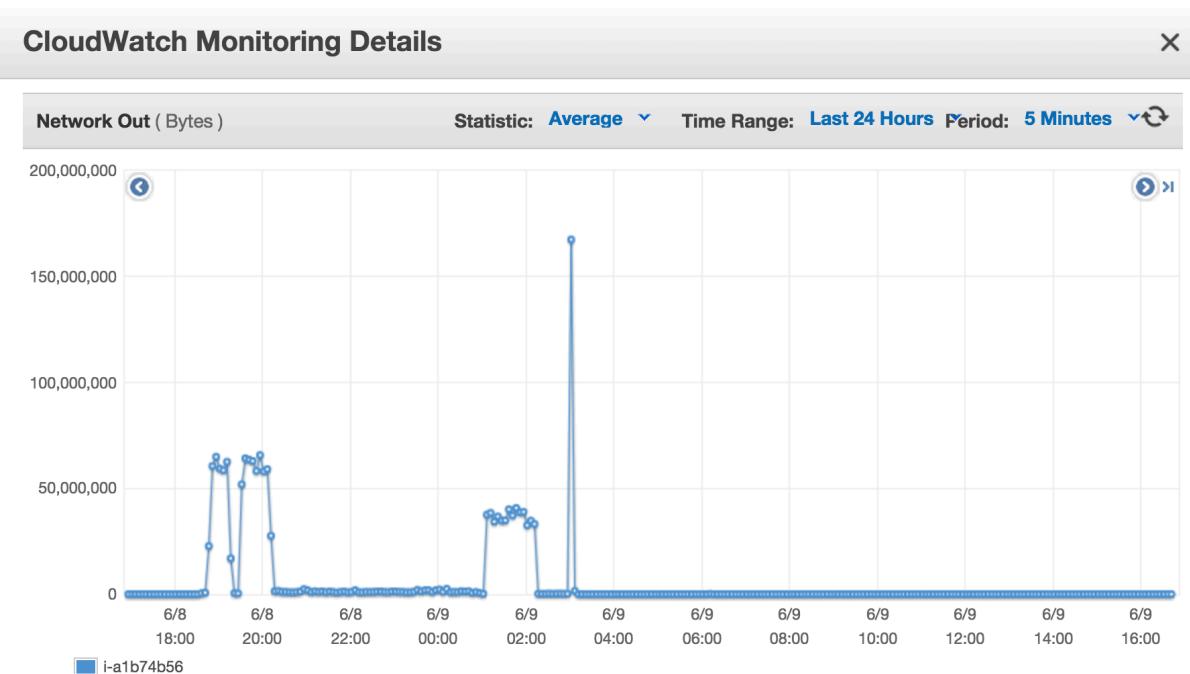
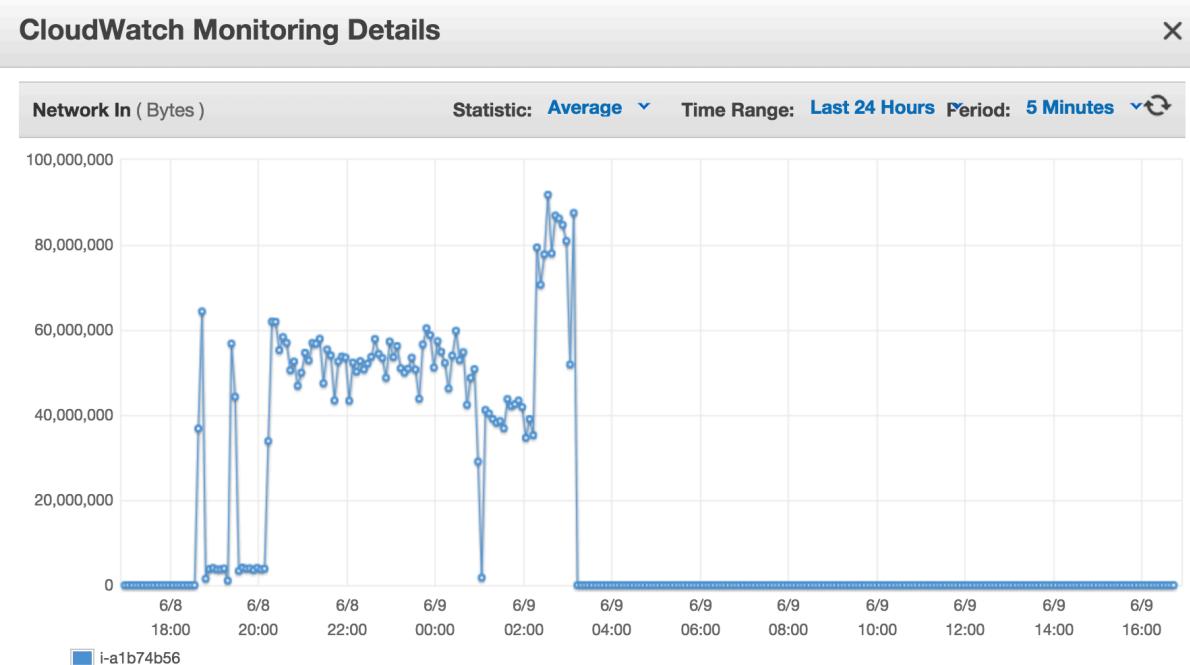
- M3.large



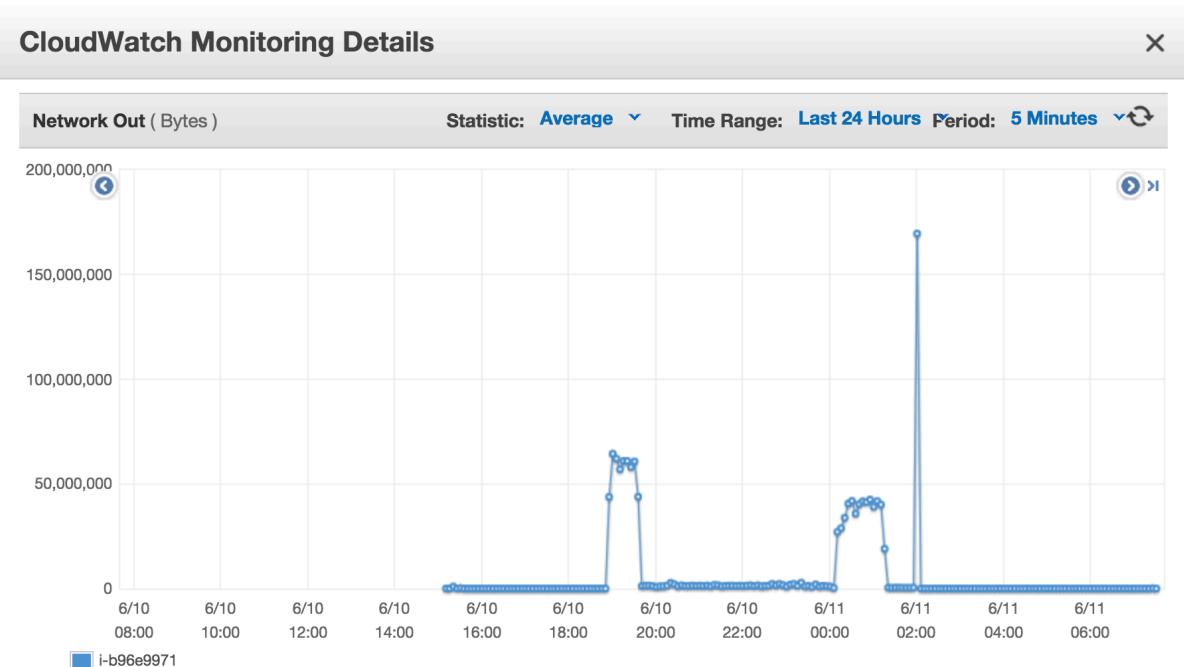
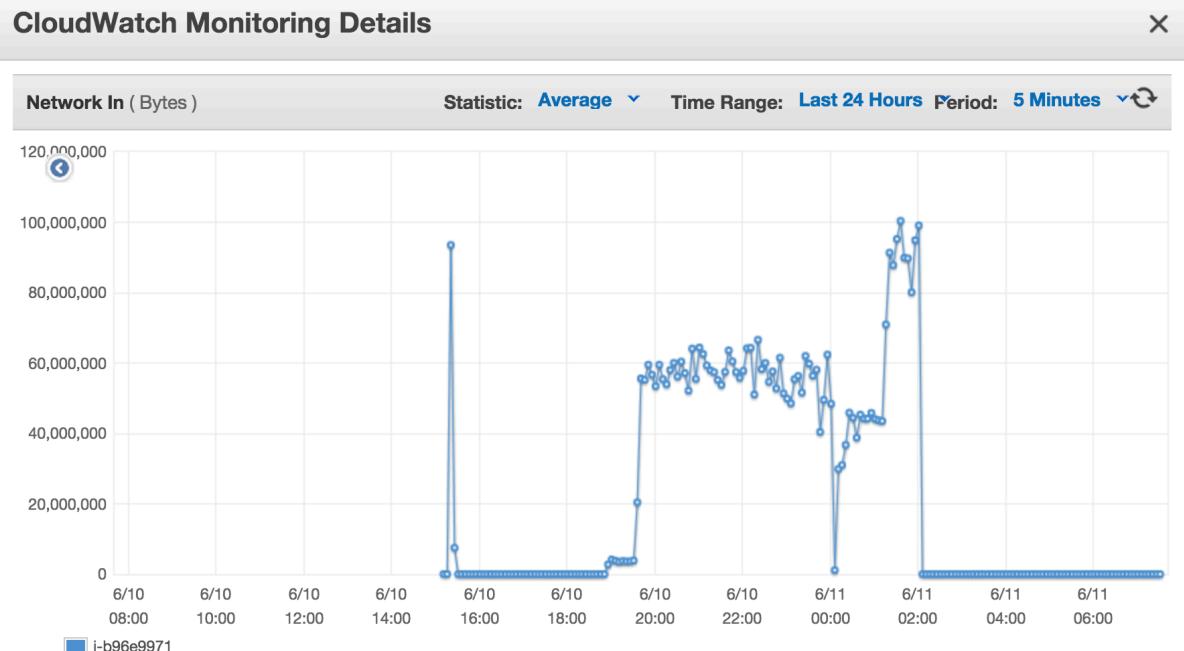
## 6. Statystyki instancji amazon'a – network in/out

Ciekawe również mogą okazać się wykresy transferu sieciowego "network in" oraz "network out", które pokazują że instancje t2 mają dużo mniejsze możliwości w tym zakresie w porównaniu do instancji m3:

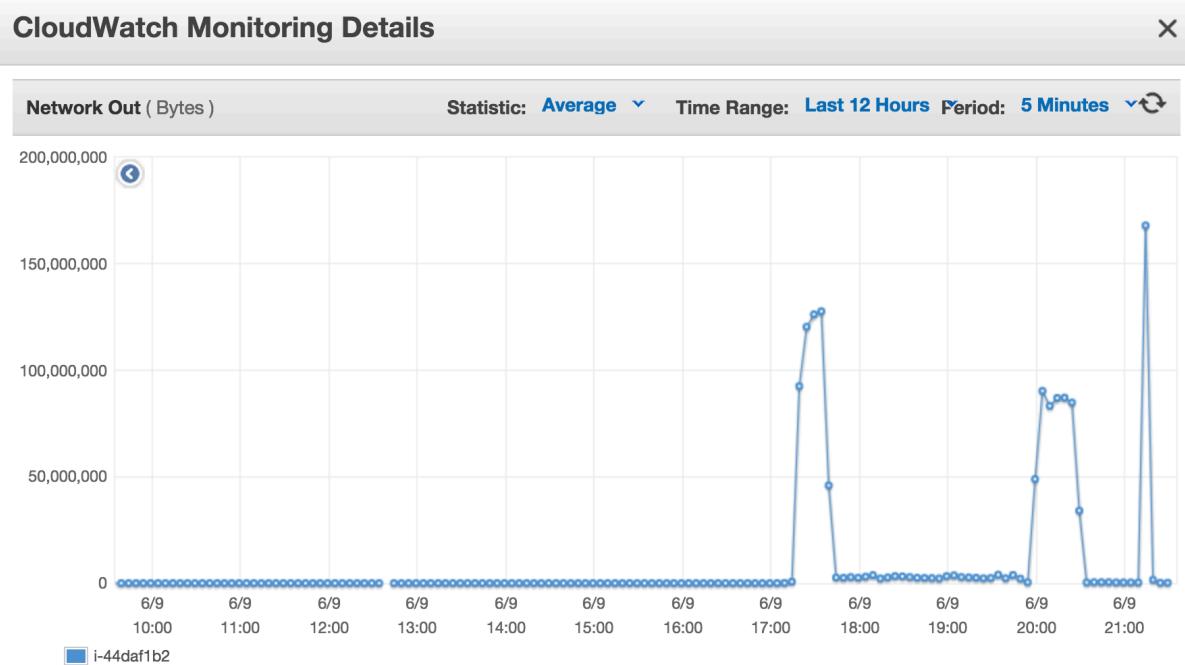
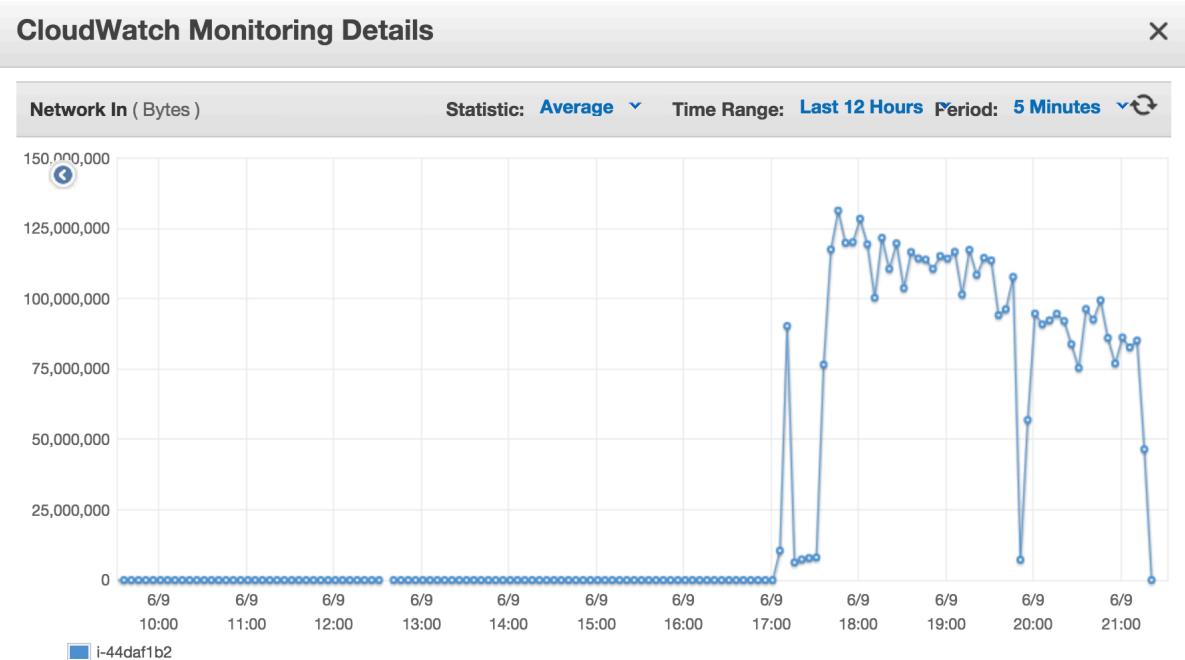
- T2.mikro



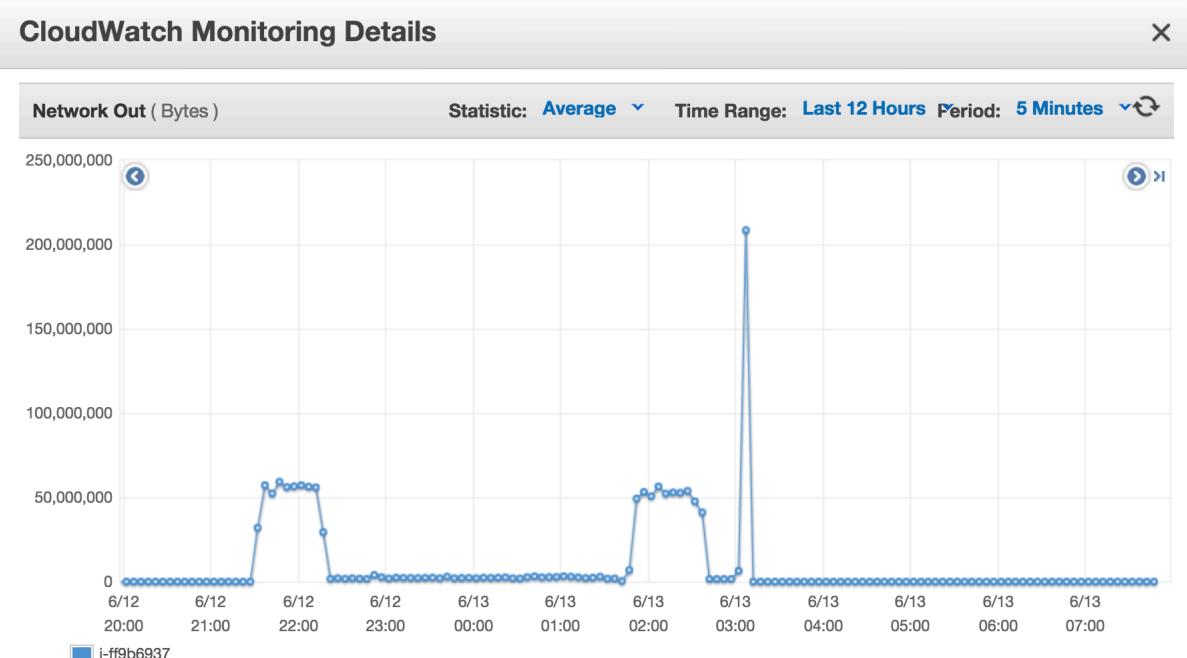
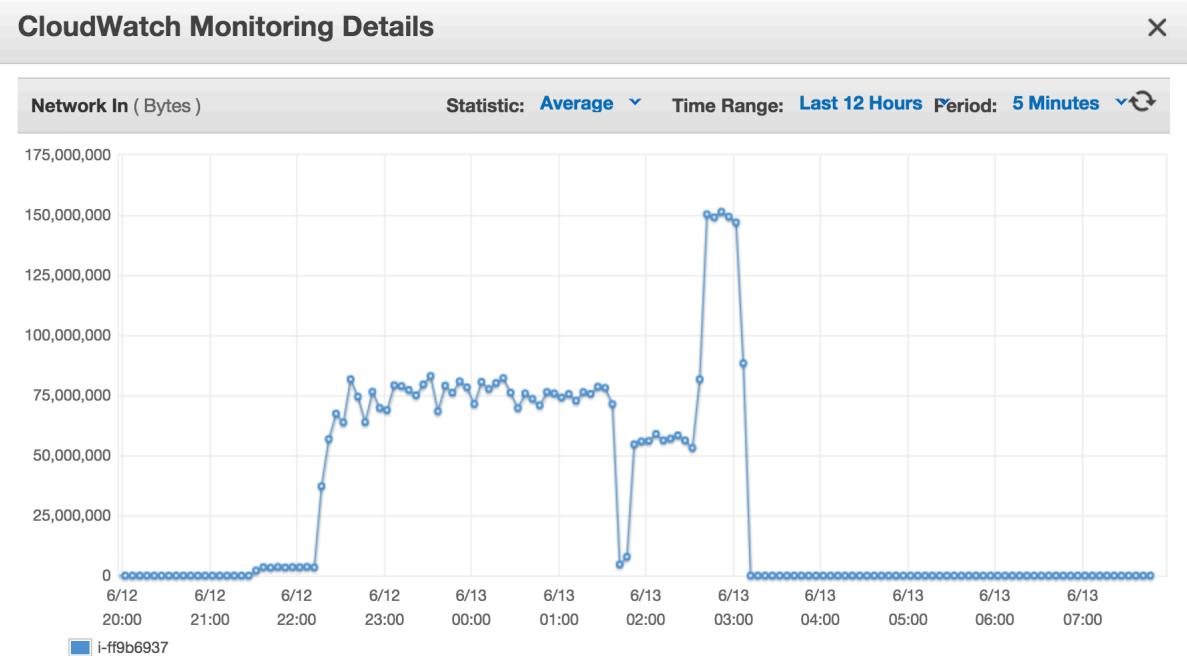
- T2.small



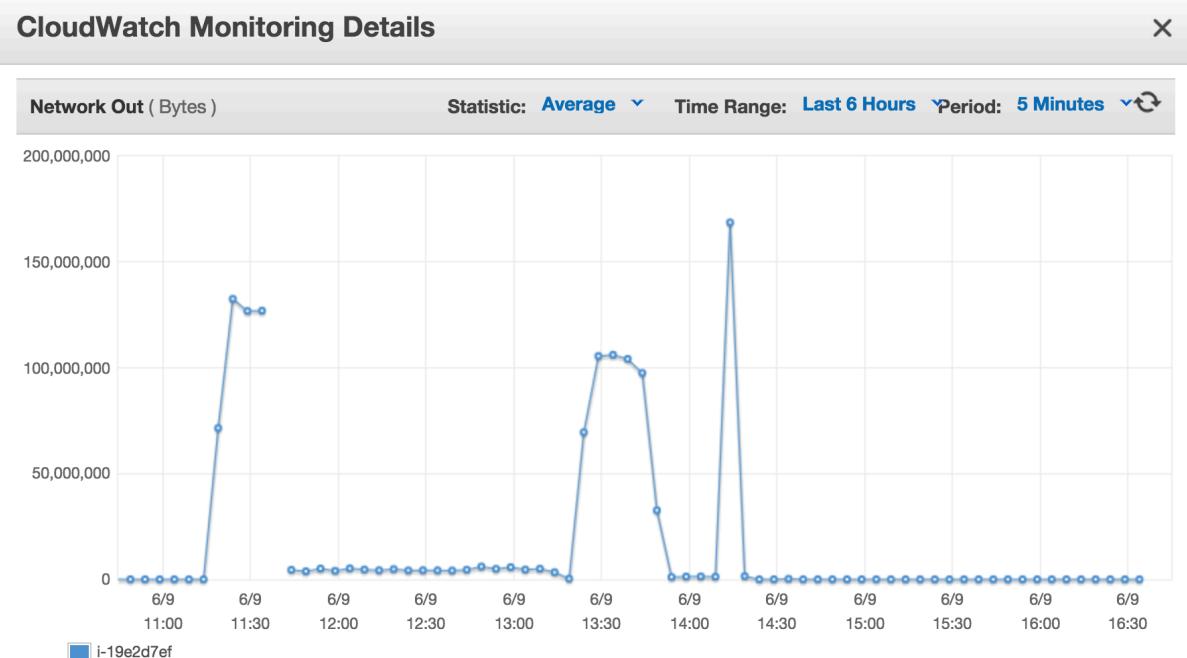
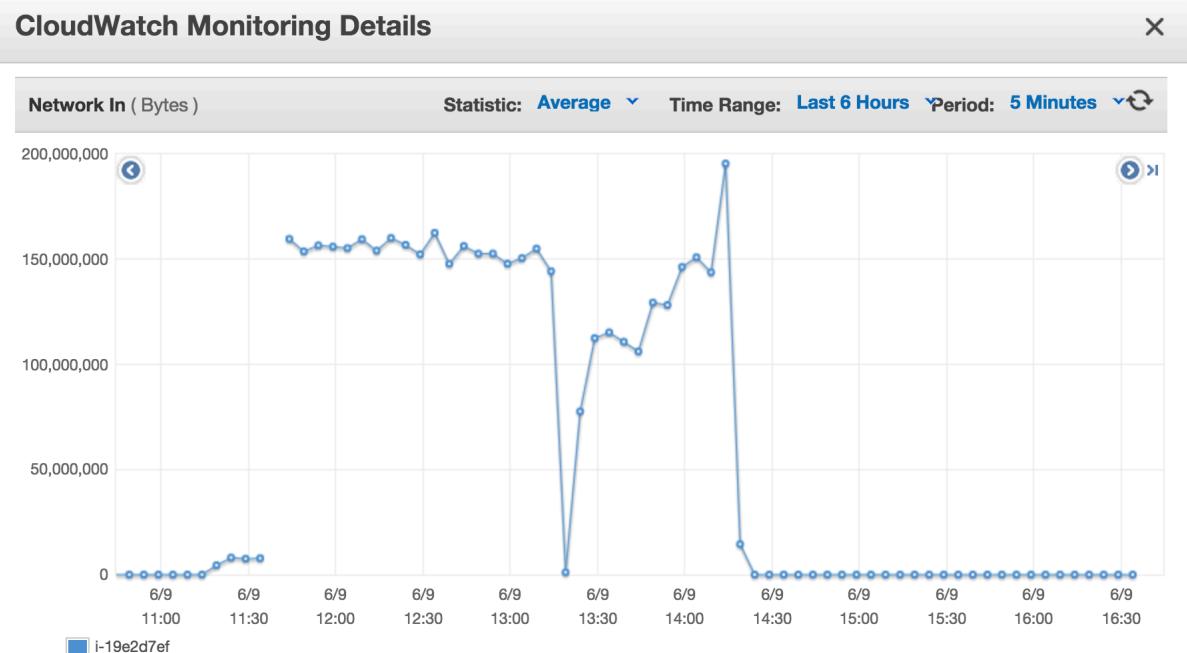
○ T2.medium



○ M3.medium



○ M3.large



Na podstawie powyższych wykresów można zauważać następujące rzeczy:

- Wszystkie instancje w swojej grupie mają bardzo podobne możliwości transferu sieciowego (gdy dla instancji posiadających 2 vCPU podzielibyśmy skalę Y przez 2 otrzymalibyśmy dokładnie takie same statystyki, co dla podobnych instancji z pojedyńczym vCPU)
- Instancje m3 wypadają lepiej niż instancje t2, co potwierdza oszacowanie jakości transferu sieciowego przez amazon dla poszczególnych typów instancji

## 7. Podwójna instancja T2.mikro

Najciekawsze efekty uzyskałem zrównolegając działanie hyperflowa na 2 instancje T2.mikro.

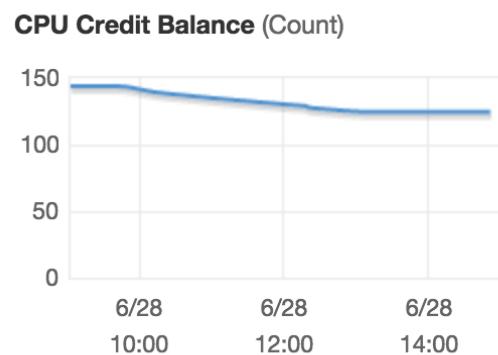
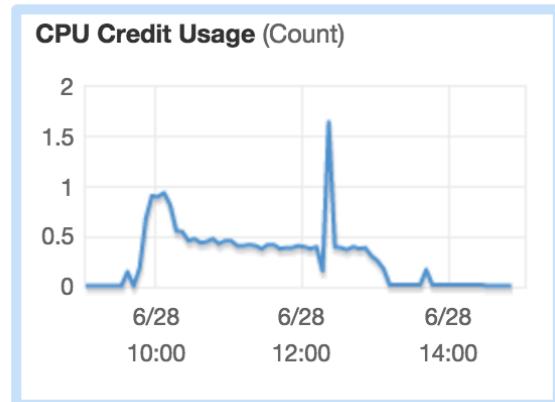
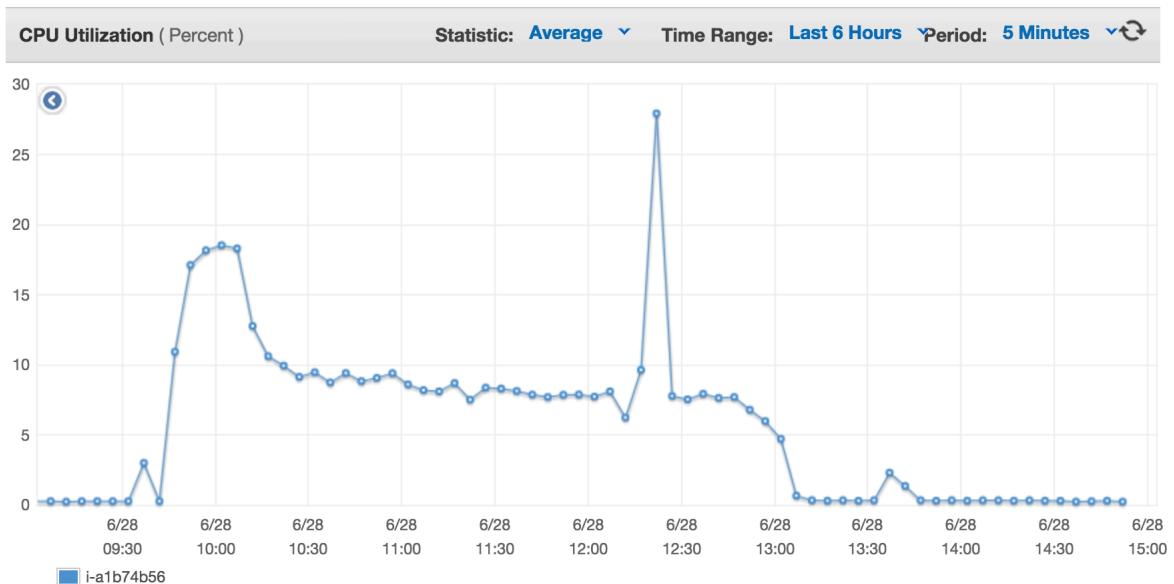
Z poprzednich satystyk można było podejrzewać, że skoro w przypadku instancji T2.mikro czas wykonanie efektywnie zależy od ilości procesorów, to łącząc 2 instancje T2.mikro możliwe jest uzyskanie czasu wykonania podobnego do pojedyńczej instancji T2.small (czyli 4h) jednocześnie uzyskując 2 razy mniejszy koszt.

Wyniki potwierdzają tę tezę. Na 2 instancjach T2.mikro uruchomiłem executor hyperflow'a, z czego na jednej działał on lokalnie na kolejce rabbitmq, a druga instancja łączyła się z kolejką rabbitmq zdalnie z pierwszą instancją.

Łączny czas wykonania wyniósł 3.89h, czyli zaokrąglając – 4h.

Łączny koszt wyniósł więc: \$0.104. Jak widać jest to taki sam koszt jak w przypadku wykonania zadania na pojedyńczej instancji T2.mikro, lecz zmniejszając czas wykonania dwukrotnie.

Oto kilka wykresów dostarczonych przez amazon:  
(dla obu instancji wykresy nie różniły się w ogóle, więc tutaj umieściłem tylko wykresy dla jednej instancji):



Jak widać w porównaniu do pojedyńczej instancji T2.mikro można zauważyć, że ilość zgromadzonych kredytów maleje. Spowodowane jest to najprawdopodobniej przez mniejszą ilość operacji przygotowania do wykonania poszczególnych zadań (transfer sieciowy ze storag'em S3) przez co zadania mogły się wykonywać znacznie częściej, więc i CPU pracował wydajniej.

Warto zwrócić uwagę również na 2 rzeczy – skrypt zbierający statystyki z hyperflowa od razu wykrywa, że zadanie wykonuje się na 2 maszynach z łącznie 2 CPU, i narysowane statystyki od razu pokazują wykres dla obu instancji

Przy wykonaniu zadania na 2 maszynach w momencie startu potrzebne są dokładnie te same pliki wygenerowane przez ./bootstrap.sh. Czyli na jednej instancji generujemy pliki I kopujemy je na drugą instancję. Jest to spowodowane tym, że niektóre pliki mają datę utworzenia w swojej nazwie, i wygenerowanie 2 osobnych zestawów danych spowoduje, że na jednej instancji będzie brakowało podstawowych plików do wykonania jakiegokolwiek zadania, co będzie skutkowało niemożliwością ich wykonania w ogóle.

## **8. Wnioski końcowe**

Po przeanalizowaniu poszczególnych statystyk i wykresów, oraz po obliczeniu rzeczywistego kosztu wykonania zadania na poszczególnych instancjach można zauważać znaczącą przewagę w całkowitym koszcie dla instancji t2.mikro w porównaniu do innych instancji.

Przez charakterystykę problem oraz działania hyperflow'a widać, że czas wykonania faktycznych zadań zajmuje jedynie kilkanaście procent całkowitego czasu wykonania. Oznacza to, że najwięcej czasu jest przeznaczane na przykład transfer plików z i do storage'u S3.

Stąd niewielka różnica pomiędzy instancjami t2.mikro oraz t2.small. Znaczącą różnicę (dwukrotnie szybszy czas wykonania) można zauważać dopiero w sytuacji gdy instancja posiada 2 vCPU. Niestety koszt za godzinę tych instancji jest dużo większy w porównaniu do innych instancji.

Widac również, że dla instancji t2.small i t2.medium, oraz m3.medium I m3.large, koszty całkowite są dokładnie takie same. Oznacza to że bardziej opłaca się wybrać do zadania instancję z 2 vCPU, bo przy tym samym koszcie uzyskujemy 2-krotnie krótszy czas wykonania.

W związku z tym pojawiają się możliwości przeprowadzenia dalszych badań. Skoro czas maleje 2-krotnie przy wykorzystaniu vCPU, a instancja t2.mikro wypada najlepiej pod względem całkowitego kosztu, możliwe jest że przy użyciu 2 instancji t2.mikro, które razem wykonują zadanie równolegle, czas całkowity wyniósł by 4h (tak jak dla instancji t2.medium z 2 vCPU oraz 2 razy krótszy niż dla pojedynczego t2.mikro), natomiast koszt całkowity wyniósł by tyle samo co dla pojedynczej instancji t2.mikro!

Wyniki badań na 2 instancjach T2.mikro potwierdziły przypuszczenia. Czas wykonania wyniósł prawie tyle samo co w przypadku pojedynczej instancji m3.large I dokładnie tyle samo dla pojedynczej instancji T2.mikro, uzyskując koszt mniejszy odpowiednio: 4 i 2 krotnie!

## **9. Mozaika nieba**

Na koniec należałyby już tylko pokazać wynik końcowy – piękną mozaikę nieba:

