Horizontal Scaling

158

Evaluate the horizontal scaling options for databases in AWS, in terms of performance and costs.

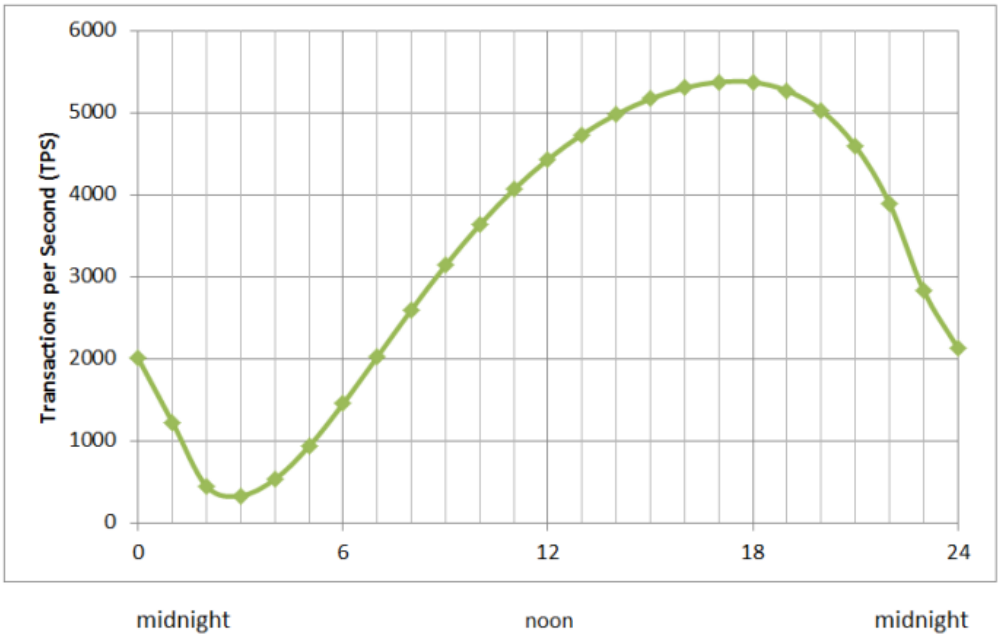
In this part of the project, you will construct a horizontally scaling database infrastructure using the tools in AWS. We shall present a scenario below along with a list of tasks to complete.

Note: For this checkpoint, assign the tag with Key: `Project` and Value: `3.3` for all resources

Scenario

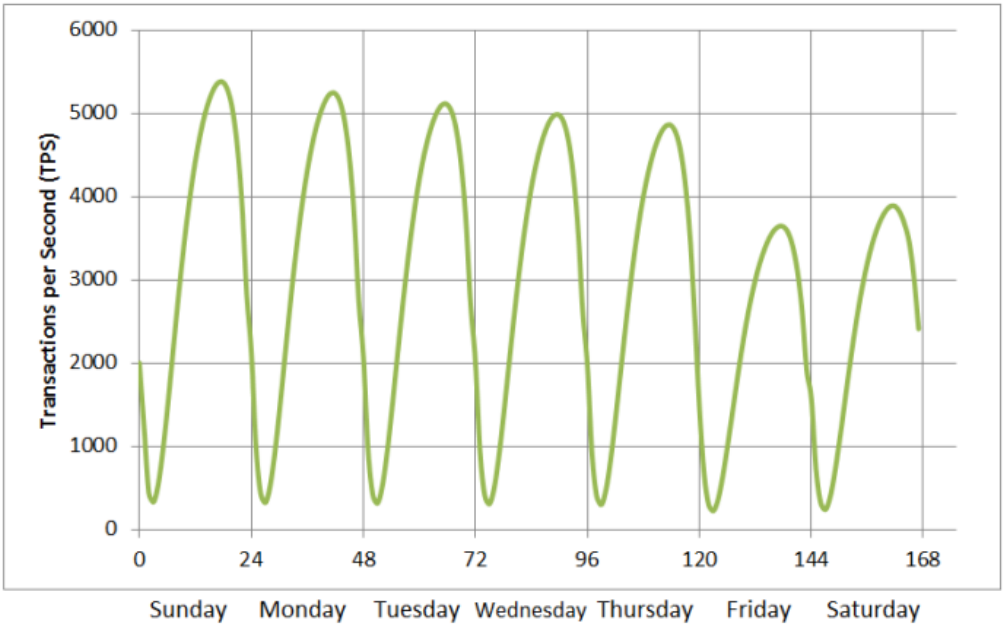
Your company has a popular wiki-based website, which has a strong database requirement. You are a member of the database team, and your task is to create a group of read-only clones of the master database - this is a valid approach when updates are infrequent, and most of the traffic comes from read (`SELECT`) statements.

The daily traffic pattern looks like the following image:



The daily traffic pattern shows peaks around 4-9pm and valleys around 3-4am (UTC)

The weekly traffic pattern is as follows:



You can notice that Sundays have an overall peak, gradually decreasing daily, with a larger drop on Fridays and Saturdays.

Based on projections by the marketing department, your company expects web traffic to grow **12%** over the next year, so you will expect your database Requests-per-Second to increase at the same rate.

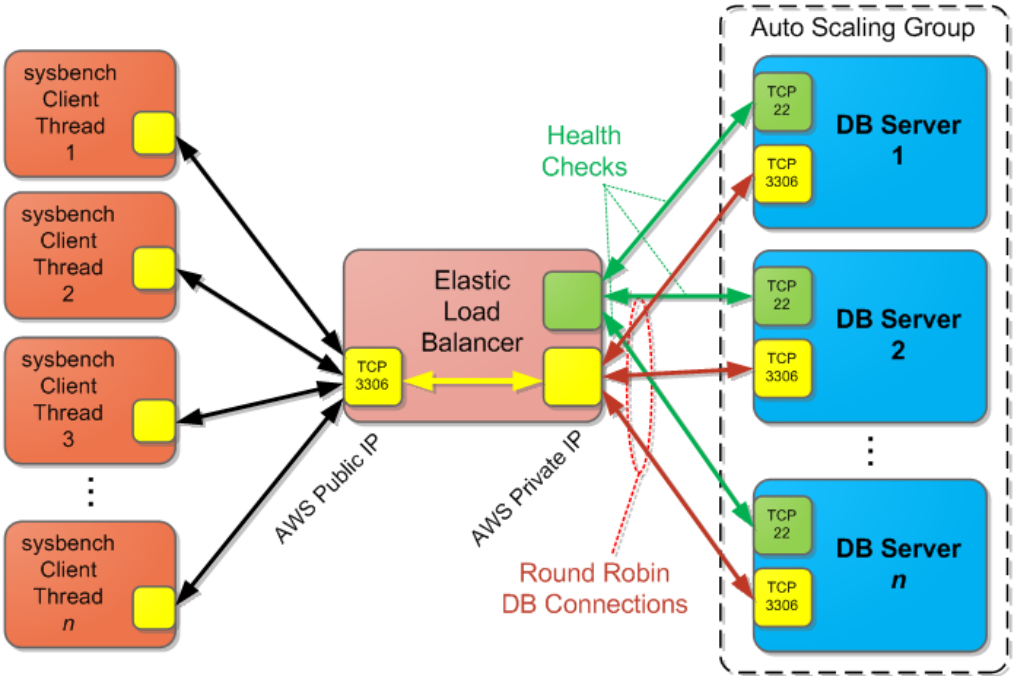
Task 1: Your first task will be to take the weekly data (`s3://15-319-s13/proj3/week0.csv`) and create a projection (in a spreadsheet or csv file) for the next year. The data in the `week0.csv` file starts at Sunday 00:00 UTC and ends at Saturday 23:00 UTC, with each row representing one hour. Call the original week 0, and each subsequent 1, 2, 3... 52 (also add 1 day to make 365 days). Each week's traffic should be ((12%/52)x week0) higher than the previous week. You will use these projected traffic numbers for checkpoint quiz and the following tasks below.

Task 2: You have found from the previous part of the project what the Transactions-per-Second (TPS) is for `m1.large` and `m1.xlarge` instance types with various storage options. Use similar methods (and AMI from previous part - `ami-fab52b93`) to determine the theoretical max TPS for a single `m1.small`. You will use m1.small instance types in an Auto-Scaling Group (ASG) behind an Elastic Load Balancer (ELB) for the next task.

Elastic Load Balancer

The Elastic Load Balancer acts as a network router that will send incoming requests to multiple EC2 Instances sitting behind it in a round-robin fashion. The Instances it points to can be added manually through the web console, programmatically through an API, or dynamically with Auto Scaling Group. It also does a Health Check to see if the host is alive (if not, it will stop sending requests to it). Using an ELB costs $0.025 per hour + $0.008 per GB transferred through it.

When the amount of queries is more than a single server can handle, we have to scale-out to multiple servers. Round-robin scheduling is not perfect, but it can do an adequate job given a large number of users, each with relatively simple requests.



Cost Projections

The Accounting department wants to have a projected cost for the following year, for your database servers. Assume that to provide consistent response time for users, you cannot load each server more than 75% of its max capacity (in TPS). The total number of servers you will need is:

$$\left\lceil \frac{\text{projected\_TPS}}{75\% \times \text{max\_TPS\_per\_server}} \right\rceil$$

Since AWS charges per-hour, you could potentially increase (or decrease) the number of database servers on an hourly basis.

Task 3.1: To figure out if the current set of servers are able to handle required QPS, we have to rely on some metrics. These metrics will be used by an autoscaling group to decide whether to add to or remove instances from the Auto Scaling Group, which is behind load balancer. The metrics you have been using (CPU, network, and disk I/O) are built into EC2 and work without any help from the guest OS. They are free when collected at 5-minute intervals (called "Basic Monitoring"), but if you want "Detailed" 1-minute intervals it costs more ($3.50 per Instance-month, or ~$0.005 per Instance-hour). To reduce costs, we want to build and use a custom metric (Custom metrics cost $0.50 per Instance-month, or ~$0.0007 per Instance-hour, and are only billed during hours in which new data is sent to CloudWatch).

Note: You do NOT have to turn on detailed metrics in order to use Custom Metrics.

In this checkpoint, we will report Transactions-per-second of the MySQL server to CloudWatch. Your custom metric will need to report "TPS Utilization", which is the ratio of the current TPS divided by the max capable for your m1.small Instance. To get the current TPS, there are mysql commands:

```
show status like 'Queries';
show status like 'Uptime';
```

- Note that each of the above commands counts as 3 queries to mysqld, so you will have to add (or subtract) 6 to get the correct number of queries.
- Every sysbench Transaction = 16 queries (so find queries-per-second first, then divide by 16)
- The unit for your metric will be Percentage

In order to send a custom metric value to CloudWatch, you need some way of authenticating. The CloudWatch CLI Tools use a credentials file where you enter your Access and Secret Keys. There is also a

CloudWatch Monitoring Script (perl) that can use credentials files, but can also use IAM (Identity and Access Management) Roles. You can build custom applications with the APIs which can also use Roles to gain temporary security credentials for specific purposes. Using IAM is more secure than putting your credentials on an Instance, because if that system has a security breach, the amount of damage an intruder can do to your AWS account is minimized. The use of IAM is optional for this project, we just wanted to point it out as a best practice.

When you launch an Instance, you can assign it an IAM Role (you cannot change which role that instance uses during its lifetime, but you can modify the role itself and those changes take place immediately). More information on creating Roles is here. A role's policy will look similar to this:

```
{
  "Statement": [
    {
      "Action": [
        "cloudwatch:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Action": [
        "ec2:DescribeTags"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

In a role policy, you need a minimum action autority of **`cloudwatch:PutMetricData`** to add custom metrics (full list of Actions here), and **`ec2:DescribeTags`** is needed when using an AutoScaling Group.

Use **`m1.small`** instances as server (which runs **`mysql`**) and client (which runs **`sysbench`**). You can use any storage type that you have used before. Use sysbench to generate varying loads, by changing the number of threads and/or processes. Figure out a way to send out custom "TPS Utilization" metric to the cloudwatch every one minute. Verify that the Cloudwatch metric reported is the same TPS that sysbench is reporting on the client side.

Task 3.2: Now that we have a custom metric, we can do auto-scaling. But we still have a problem. The autoscaling group dynamically adds/removes servers. We can't manually partition an EBS, mount it, copy data and start mysql whenever a server is added. That defeats the purpose of autoscaling. Hence you have to create an AMI which does all these things when it gets booted. Remember to include the code which sends custom metrics to CloudWatch in the AMI. You should use your new AMI while creating auto-scaling group.

You must test the feasibility of Auto-scaling based on the new custom metric. Do this testing with m1.small and ordinary EBS storage, using Policy of Min=1, Max=4, Desired=1. Again use sysbench to generate varying loads, by changing the number of threads and/or processes. Verify that the Cloudwatch metric reported is the same TPS that sysbench is reporting on the client side (or sum of multiple clients).

Task 4: Calculate **maximum-Transactions-per-dollar** for each instance type (m1.small, m1.large, m1.large+EBS_Optimized, m1.xlarge, m1.xlarge+EBS_Optimized) assuming all have one 500GB EBS volume (Standard IOPS, no Snapshots) attached. Do not include Network/Bandwidth, ELB, or CloudWatch costs for this task. Record your findings for the Checkpoint Quiz.

Task 5: Using the Instance type with the highest Transactions-per-dollar you found above, calculate the total projected cost of your databases for a year. For this task, include ELB costs (assume 1kB per transaction) and CloudWatch (1 custom metric and Detailed monitoring per Instance). Still exclude out bound network/bandwidth costs (other than ELB). Record your findings for the Checkpoint Quiz.

Notes, Assumptions and Hints

- You can use port 22 (SSH) for ELB Health Checks since that port is already open. There is a specific Security Group added for your ELB, if you want to add that to your Firewall rules. Do not use port 3306 (MySQL) for ELB Health Checks - mysqld thinks these are malformed requests and may subsequently block regular requests coming from ELB.
- When testing using sysbench through ELB, it may be necessary to run multiple separate processes in parallel (with fewer threads each), rather than a single process with many threads. This is because ELB is session oriented. Similarly, you might find that you need multiple m1.small clients running sysbench before you fully exercise your servers behind ELB.
- For your time calculations, Assume 1 year = 365 days = 52 weeks + 1 day = 8760 hours; 1 month = 730 hours.
- 1 kB = 1000 Bytes ; 1MB = 1000 kB; 1GB = 1000 MB; 1TB = 1000 GB.
- For EBS calculations, assume 1 transaction = 1 IOP. Each Instance will get its own 500GB volume. This size does not change throughout the year.
- The AWS Monthly Calculator might be helpful for getting an idea of total costs.
- Assume that AWS prices will not increase within a year.
- Use **On-Demand** pricing for us-east (N. Virginia)

checkpoint

Horizontal Scaling