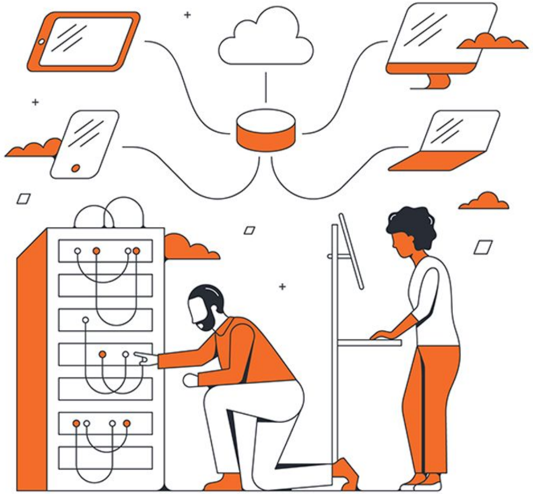
Scalable System

Common Problems

Asmit Ghimire
Software Engineer at Gurzu Inc.

What does Scalability mean for Systems and Services?

- System that can handle rapid changes to workloads and user demands.

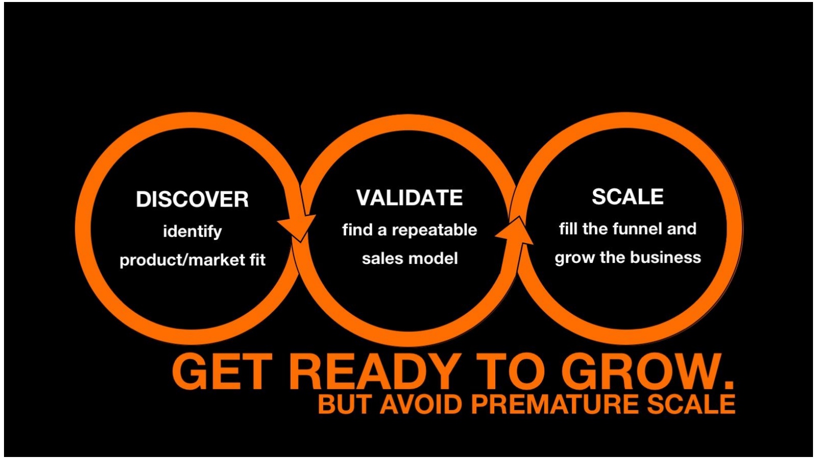


Premature Scaling

- ❖ Premature scaling happens when startups try to scale up too early.
- ❖ It can kill a startup company with unplanned budgeting.

Proper steps to scale:

1. Discover
2. Validate
3. Scale

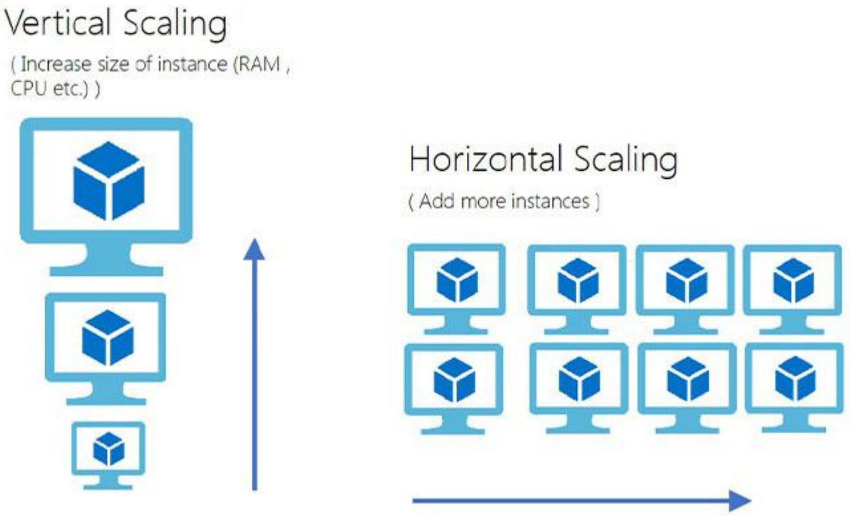


Problems Faced in a Scaling System

- Issues encountered as of my experience:
  - Insufficient system infrastructure
  - N+1 queries
  - Memory leak / bloat
  - Bulky / slow APIs
  - Missing DB indexes

Server Scaling

- There are two ways we can go about scaling servers:
  - Vertical Scaling
  - Horizontal Scaling



Vertical Scaling

- Increase the capability of a single server.
- Appropriate for a growing system.

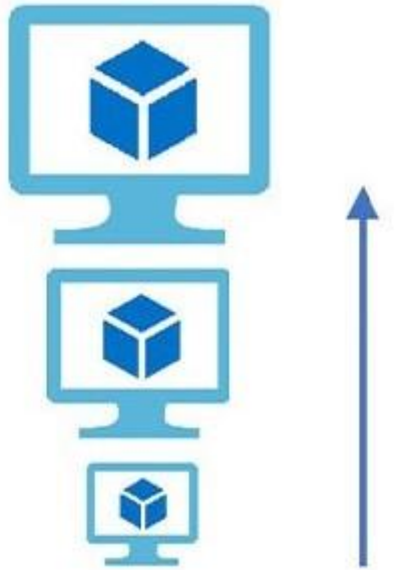
Pros:

- Easy to implement and maintain
- Cost effective

Cons:

- Single point of failure
- Hardware limitations

(Increase size of instance (RAM , CPU etc.))



Horizontal Scaling

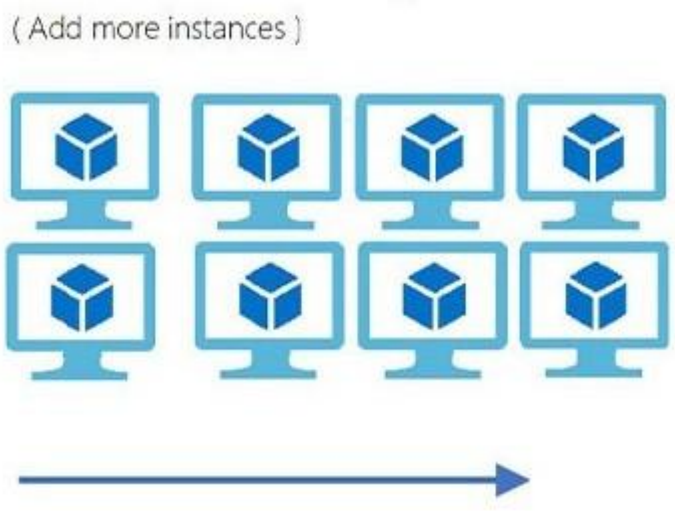
- ❖ Add more of the same instance of resources.
- ❖ Workload distributed by load balancers.
- ❖ Appropriate for a large system.

Pros:

- ➢ More fault tolerance and fewer risks of downtime
- ➢ Scales well with increasing demand

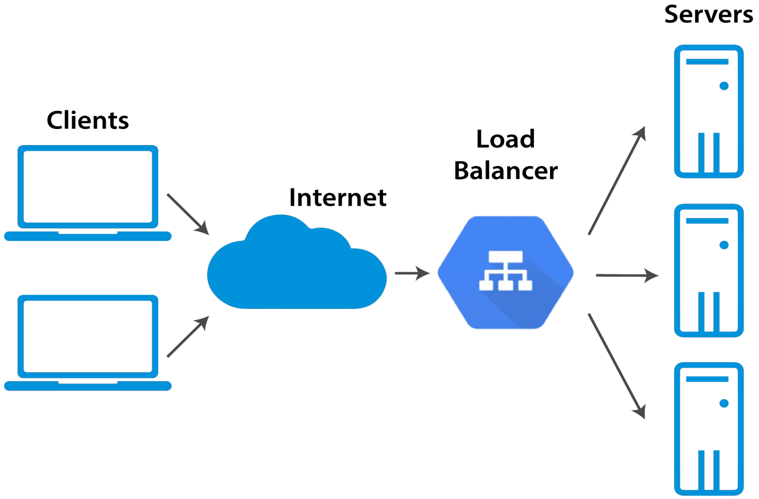
Cons:

- ➢ Load balancing is required
- ➢ Expensive



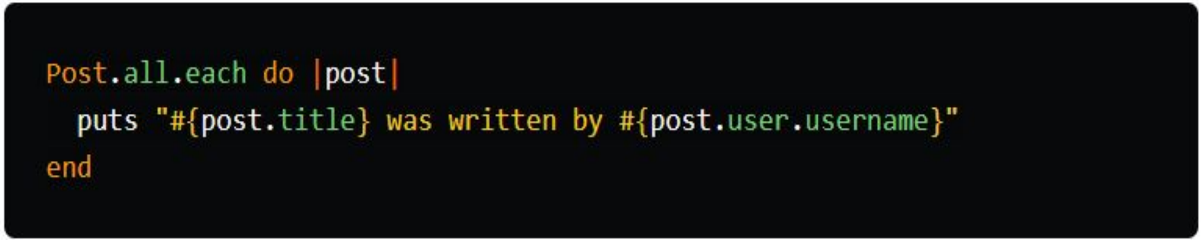
Load Balancing

- LB distributes incoming network traffic across multiple servers.
- Goals:
  - Maximize resource utilization
  - Minimize response time
  - Maximize throughput
- Some of the popular load balancers are Nginx, ELB (Elastic Load Balancing), HAProxy and Loadbalancer.org.



What is the N+1 Query Problem?

- ❖ The N+1 query problem happens when a query is executed on each result of the previous query.
- ❖ Here’s an example with Rails:
  - ➢ Let’s say we have Post model which belongs to a User.

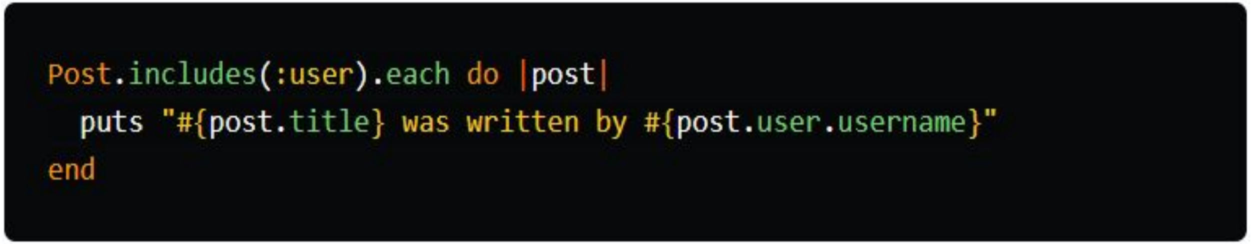


```ruby
Post.all.each do |post|
  puts "#{post.title} was written by #{post.user.username}"
end
```

  - ➢ In the above example, it first retrieved all the Post objects and then user for each post.
  - ➢ If there are 10 posts in the database then 10 + 1 = 11 queries would be executed.

Solve the N+1 problem with "eager loading"

- With eager loading, a query loads a resource as soon as the code is executed.
- In Rails, you can use includes method for eager loading.
- Let’s rewrite the previous example using includes method:



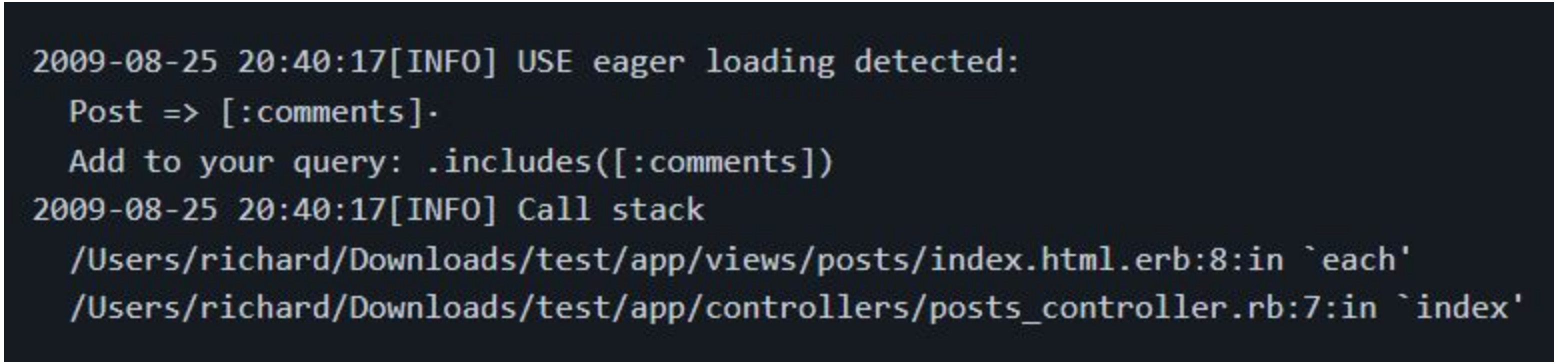
```
Post.includes(:user).each do |post|
  puts "#{post.title} was written by #{post.user.username}"
end
```

- Here, user for all the posts are pre-loaded in the memory which omits the additional DB calls to fetch user’s name.

Find the N+1 query problem with the Bullet Gem

- Simple and easy to integrate
- Automate N+1 tracing
- Log recorded issues

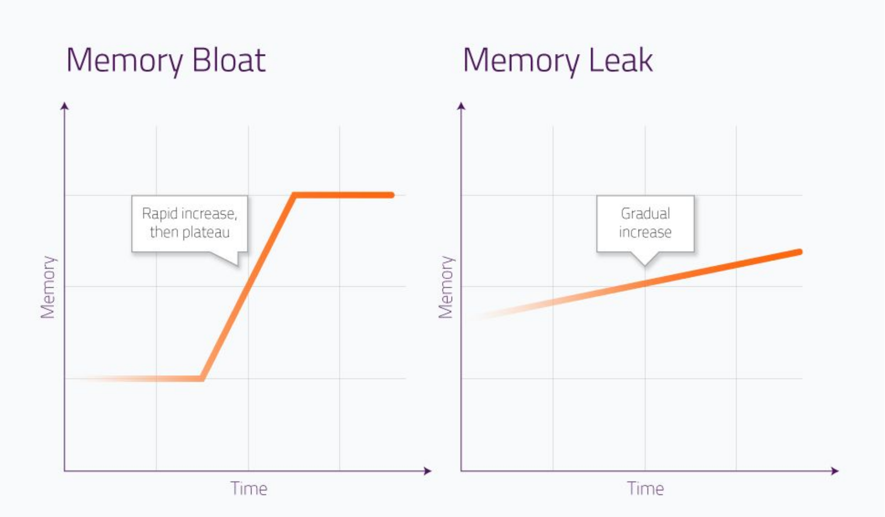
Tracing example:



```
2009-08-25 20:40:17[INFO] USE eager loading detected:
  Post => [:comments]·
  Add to your query: .includes([:comments])
2009-08-25 20:40:17[INFO] Call stack
  /Users/richard/Downloads/test/app/views/posts/index.html.erb:8:in `each'
  /Users/richard/Downloads/test/app/controllers/posts_controller.rb:7:in `index'
```

Memory Leak / Bloat

- Memory bloat:
  - Sudden increase in memory consumption
- Memory leak:
  - More of a slow, gradual increase in memory usage



Locate Memory Leak / Bloat with APM

- APM (Application Performance Monitoring) helps to track errors and monitor applications with insights on CPU and memory usage.
- Some of the popular APM tools are New Relic and Datadog.

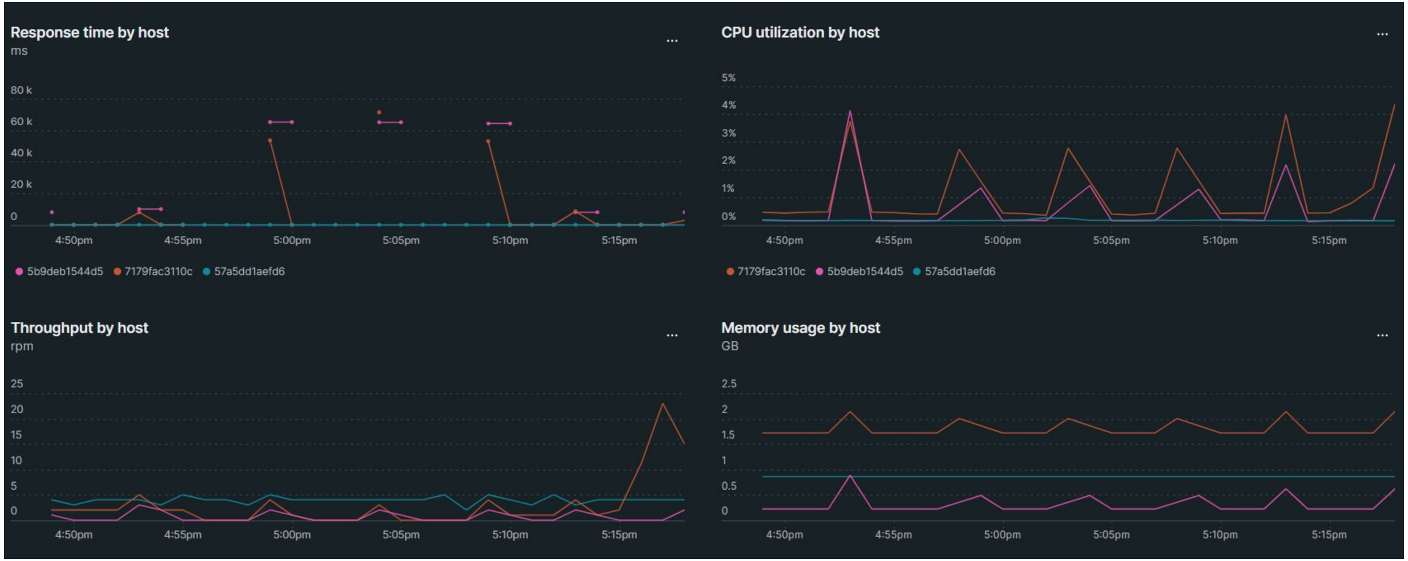
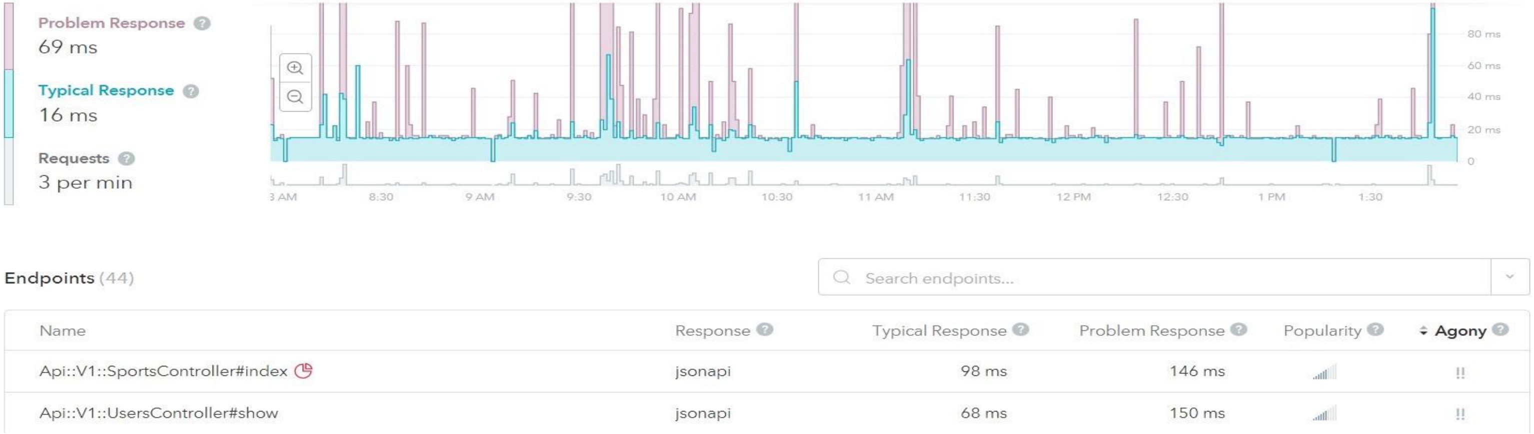


Fig: New Relic Chart

Profile Rails Endpoints with Skylight.io

- Besides APMs, Skylight.io is a smart profiler for Ruby and Rails applications which can aid in pinpointing problem endpoints.



Problem Response
69 ms

Typical Response
16 ms

Requests
3 per min

Endpoints (44)

Search endpoints...

Name	Response	Typical Response	Problem Response	Popularity	Agony
Api::V1::SportsController#index	jsonapi	98 ms	146 ms		!!
Api::V1::UsersController#show	jsonapi	68 ms	150 ms		!!

Fig: Skylight.io Dashboard

- Helps locate APIs with:
  - Slow response
  - High object allocations
  - N+1 query

Cutting down Bulky APIs into Snappy Ones

- Bulky APIs issues:
  - Slow response cycle
  - Poor experience for end users
  - Heavy server resource usage

- Snappy APIs benefits:
  - Quick response cycle
  - Good user experience
  - Distributed server load

Conclusion

- Facing multiple issues in our experience of scaling a Rails application, we were able to make the following improvements:
  - There was around a 146% increment in the throughput
  - The response time was decreased by 190% in an average
  - Average memory consumption was brought down to ~2.5 GB from ~8 GB
  - Got rid of CPU spikes and server crashes

THANK YOU.

Feel free to ask questions.

Useful Links

Premature Scaling: https://bizxpand.com/go-to-market/premature-scaling/#:~:text=What%20exactly%20is%20premature%20scaling,the%20rest%20of%20the%20operation.%E2%80%9D

System Scaling: https://www.lucidchart.com/blog/what-does-scalability-mean-for-systems-and-services

Rails N+1 queries and eager loading: https://dev.to/junko911/rails-n-1-queries-and-eager-loading-10eh

Load Balancing: https://www.nginx.com/resources/glossary/load-balancing/

- Find me on:
  - twitter: @as_m_it
  - linkedIn: https://www.linkedin.com/in/asmit-ghimire/