CS 188/219

Scalable Internet Services

Andrew Mutz
April 4, 2015

Today's Agenda

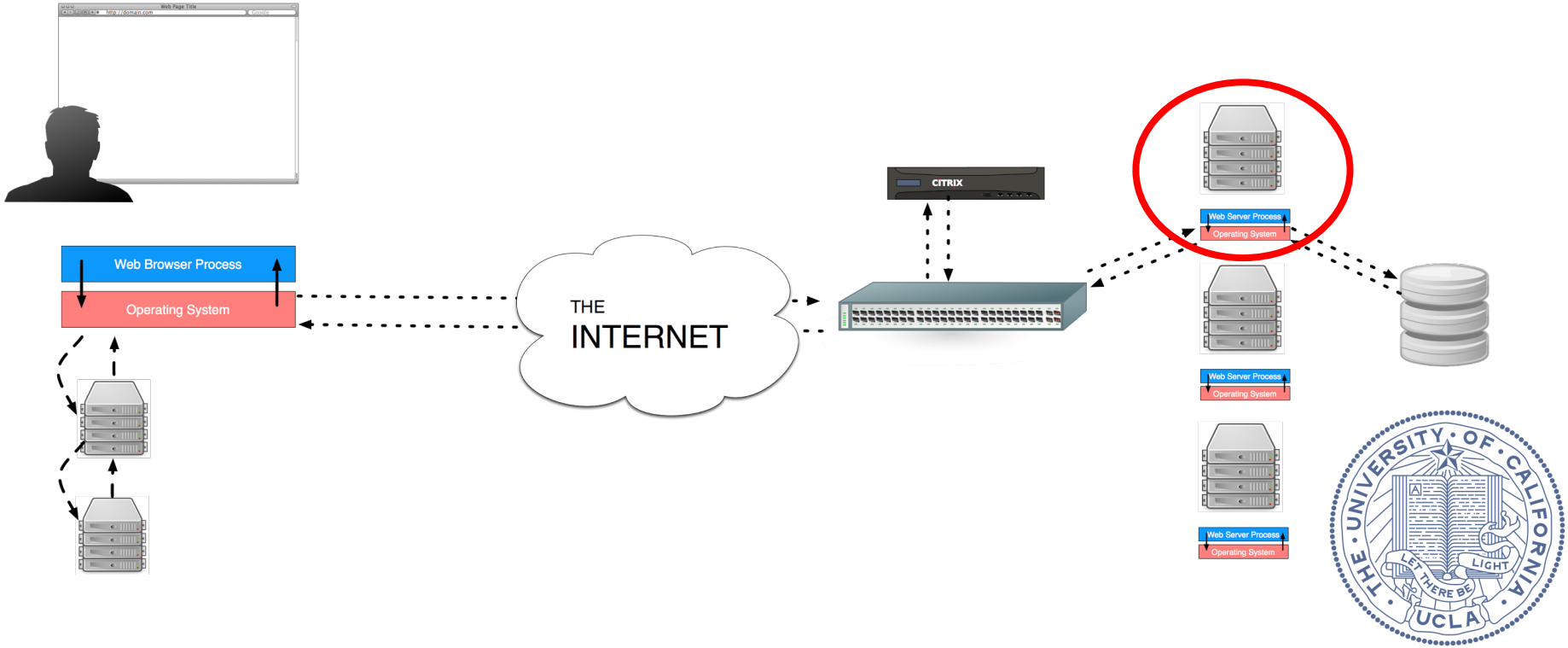
Motivation

Server-side Caching

Deploying on AWS

For Next Time

Server-side Caching



Motivation

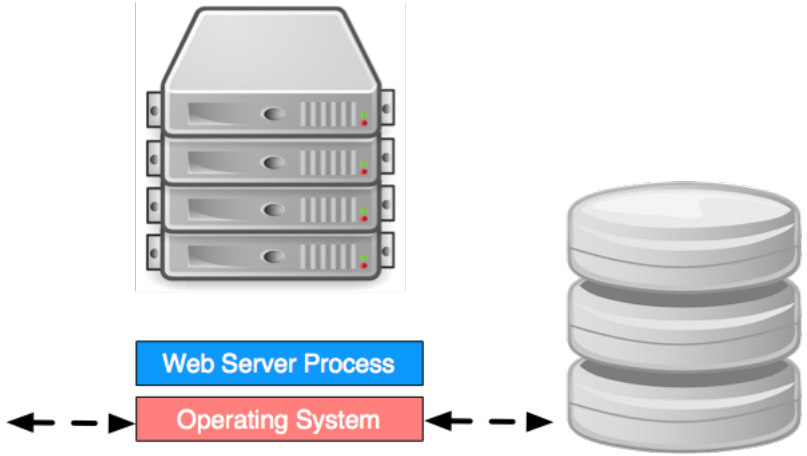
After today you should understand

- Why server side caching exists
- What options you have when using server side caching
- How to use this in your projects
- How to deploy on AWS using CloudFormation

Server-side Caching

We have a web server process that is repeatedly responding to HTTP requests from a variety of clients.

Responding to each request requires computation and I/O to be performed, and this can be expensive.

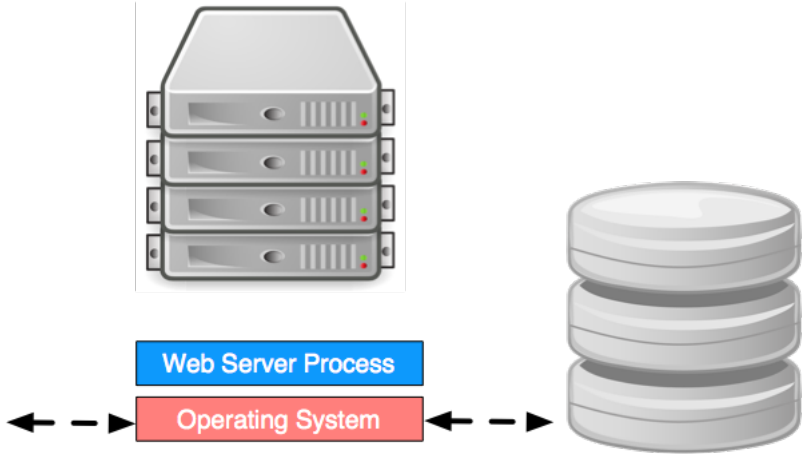


Server-side Caching

In practice, there is a great deal of similarity between responses.

In the last lecture (HTTP Caching) we looked at optimizing scenarios where repeated responses are identical.

In this lecture we will look at optimizing scenarios where repeated responses are not identical, but are similar.

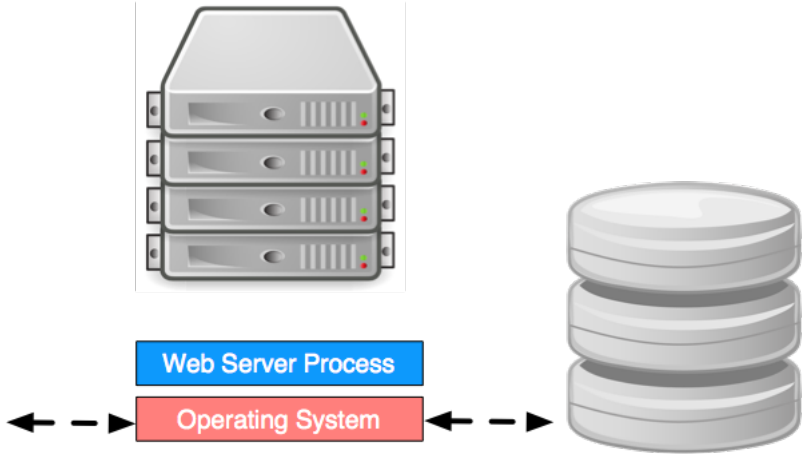


Server-side Caching

There are many parts of a response that are similar.

There are many steps to creating a response that are repeated.

What can you think of?



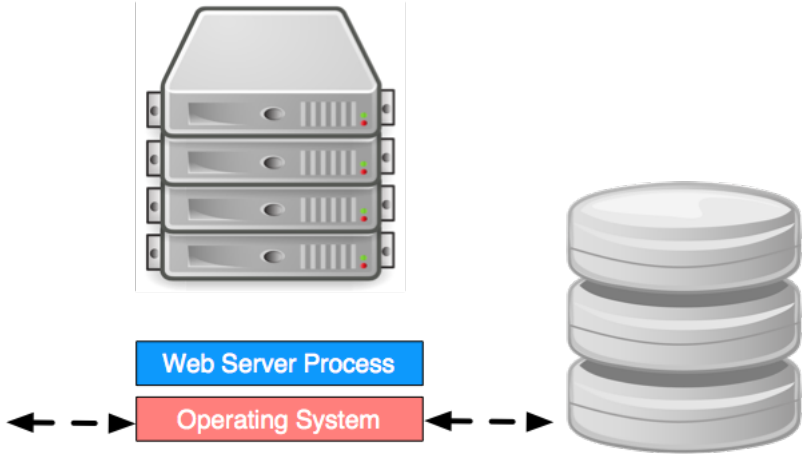
Server-side Caching

There are many parts of a response that are similar.

There are many steps to creating a response that are repeated.

What can you think of?

- View Fragments
- Rarely modified ORM objects
- Any summarized data that is expensive to compute



Server-side Caching

View Fragments: Similarity between pages

```
<html><head><meta name="referrer" content="origin"><link rel="stylesheet" type="text/css" href="news.css?Dc4WHhDIHIntiL2Oh45C">

        <link rel="shortcut icon" href="favicon.ico">

          <link rel="alternate" type="application/rss+xml" title="RSS" href="rss">

        <script type="text/javascript">

          function hide(id) { var el = document.getElementById(id); if (el) { el.style.visibility = 'hidden'; } }

          function vote(node) { var v = node.id.split(/_/); var item = v[1]; hide('up_'   + item); hide('down_' + item); var ping = new Image(); ping.src = node.href;
return false;}

        </script><title>Hacker News</title></head><body><center><table id="hnmain" op="news" border="0" cellpadding="0" cellspacing="0" width="85%" bgcolor="#f6f6ef">

        <tr><td bgcolor="#ff6600"><table border="0" cellpadding="0" cellspacing="0" width="100%" style="padding:2px"><tr><td style="width:18px;padding-right:4px"><a href="
http://www.ycombinator.com"><img src="y18.gif" width="18" height="18" style="border:1px #ffffff solid;"></a></td>

                  <td style="line-height:12pt; height:10px;"><span class="pagetop">

                              <b><a href="news">Hacker News</a></b><img src="s.gif" height="1" width="10"><a href="newest">new</a> | <a href="newcomments">comments</a> | <a
href="show">show</a> | <a href="ask">ask</a> | <a href="jobs">jobs</a> | <a href="submit">submit</a></span></td><td style="text-align:right;padding-right:4px;"><span
class="pagetop">

                              <a href="login?goto=news">login</a></span></td></tr></table></td></tr><tr style="height:10px"></tr><tr><td><table border="0" cellpadding="0"
cellspacing="0">

            <tr><td align="right" valign="top" class="title"><span class="rank">1.</span></td>

              <td><center><a id="up_9403571" href="vote?for=9403571&amp;dir=up&amp;goto=news"><div class="votearrow" title="upvote"></div></a></center></td><td class="title"
><span class="deadmark"></span><a href="http://fossdroid.com/">Fossdroid.com: Free and open source Android applications</a><span class="sitebit comhead"> (fossdroid.com)
</span></td></tr><tr><td colspan="2"></td><td class="subtext">

        <span class="score" id="score_9403571">164 points</span> by <a href="user?id=SnaKeZ">SnaKeZ</a> <a href="item?id=9403571">4 hours ago</a>                | <a
href="item?id=9403571">38 comments</a></td></tr>

      <tr class="spacer" style="height:5px"></tr><tr>.....
```

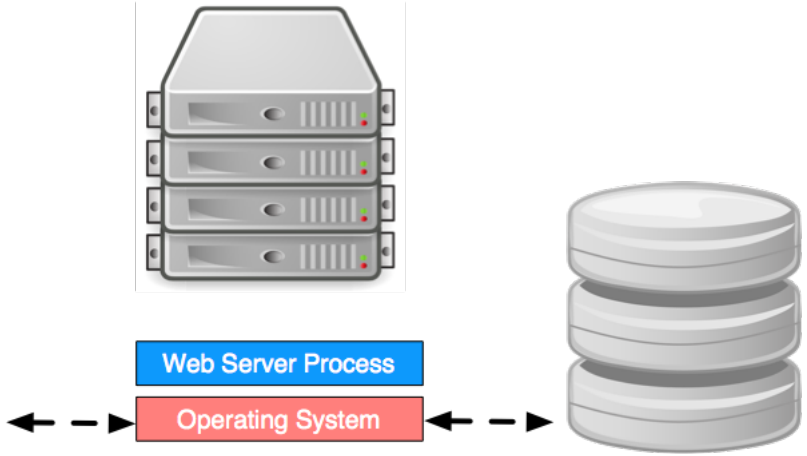
Server-side Caching

Rarely modified ORM objects?

- User permissions
- Configuration options
- Any database-backed data that changes rarely

Summarized data that is difficult to compute

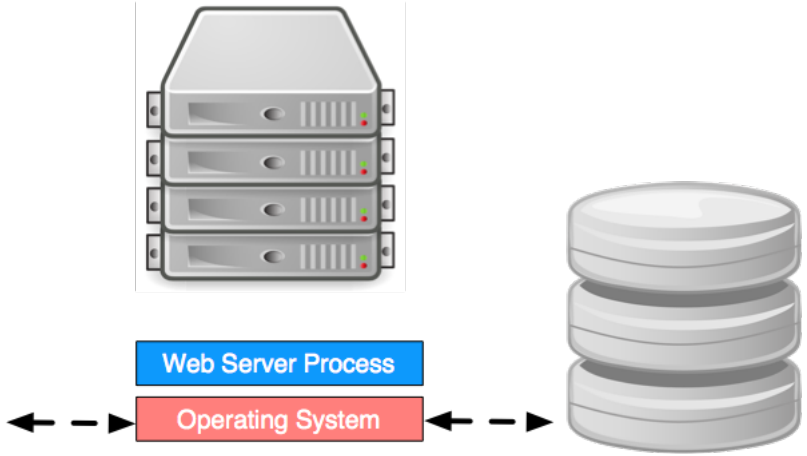
- Any particularly heavyweight SQL query
- Example: Total account balance on Mint.com



Server-side Caching

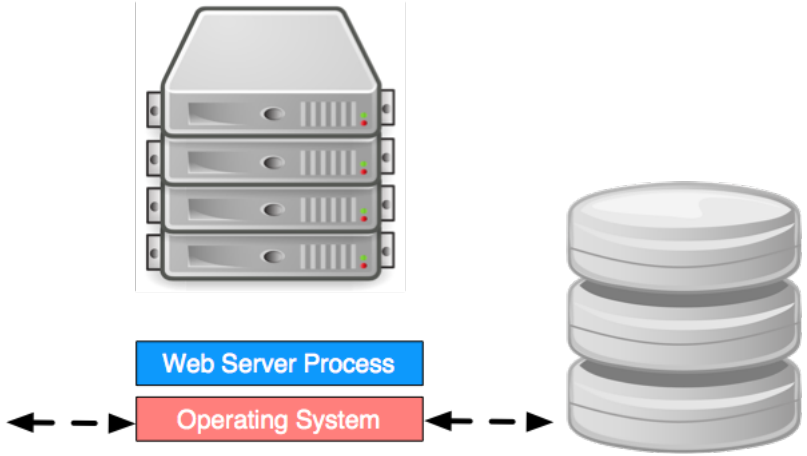
Some of these things are expensive to materialize

- View fragments are produced by extensive string manipulation.
  - Ruby optimizes humans over CPU
- The database can be a bottleneck in our current architecture
- Some SQL queries are necessarily heavyweight



Server-side Caching

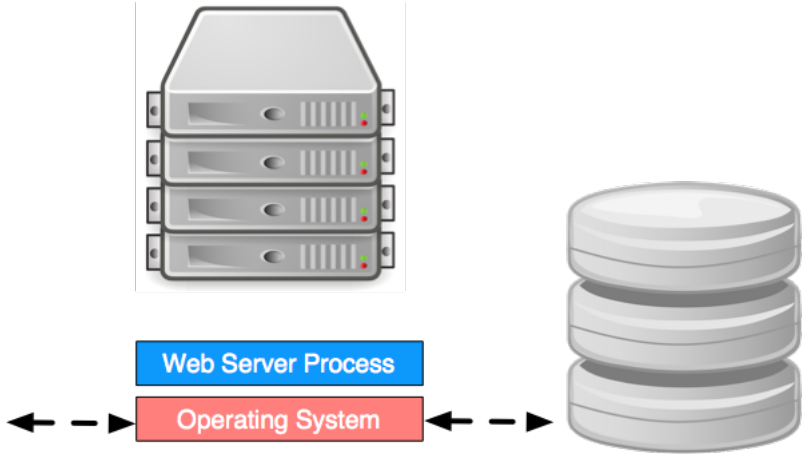
So if we want to keep previously computed results around between requests, how should we do it? Where should we put it?



Server-side Caching

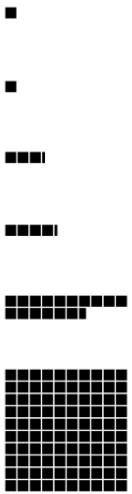
So if we want to keep previously computed results around between requests, how should we do it? Where should we put it?

- Just keep it in memory between requests?
- Store it on the filesystem?
- Store it in memory on another machine?



All of these are reasonable options. Lets look into each in more depth.

Server-side Caching



1ns

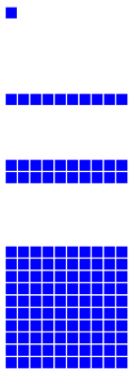
L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■

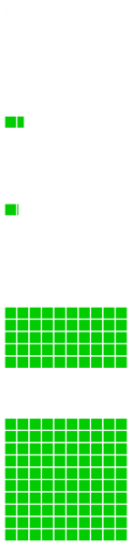


Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = ■



Send 2,000 bytes over commodity network: 0.0ns ≈ 0.4µs

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 12,000ns ≈ 12µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 200,000ns

Disk seek: 4,000,000ns ≈ 4ms

Read 1,000,000 bytes sequentially from disk: 2,000,000ns ≈ 2ms

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Server-side Caching

What can we conclude from these numbers?

- Storing in memory and reading later is fast:
  - Random reads from memory will be 0.1µs, reading 1MB will be 12µs
- Storing on disk is slow *without SSD*:
  - Disk seek is 4000µs, subsequent sequential read of 1MB is 2000µs
- Storing on disk *with SSD* is much more reasonable:
  - Random read is 16µs, sequential read of 1MB will be 200µs
- Storing on another machine is reasonable:
  - Round trip within datacenter is 500µs.

Server-side Caching

Summary

- In memory: tens of µs
- On SSD: hundreds of µs
- On Disk: thousands of µs
- And if its on a remote machine, add hundreds of µs.

Conclusion

- Always use SSD
- Memory > SSD > Remote?

Server-side Caching

It's not that simple. Why?

Server-side Caching

It's not that simple.

What effect on the cache hit rate does each of these designs have?

- In memory: Cache per process
- On SSD: Cache per machine
- On (single) remote machine: Cache per cluster

Server-side Caching

Conclusion:

- **In memory**: highest performance, lowest hit rate
- **On SSD**: lower performance, higher hit rate
- **On remote cache server**: lowest performance, highest hit rate

There is no silver bullet. How will each of these affect system performance:

- Number of processes per machine?
- Concurrency model of Application Server: threads vs. processes?
- Number of machines per cluster?

Memcached

Memcached is a commonly used implementation of a remote cache server

- Keeps a cache in memory
- Accepts TCP connections and returns lookup requests
- Distributed key-value store
  - Keys can be up to 250 bytes, values can be up to 1MB
  - Can scale horizontally
- When it runs out of space, it uses a simple LRU mechanism to make more space
- Lightweight features, everything is constant time.
- Originally developed at LiveJournal

Another commonly used tool is Redis

Rails Caching

The good news for your projects is that Rails has great support for server-side caching.

Rails emphasizes three types of caching:

- HTTP caching
- Fragment caching
- Low level caching

We covered HTTP caching in the last lecture, so today we will talk about fragment caching and low-level caching

Rails Caching

By default, caching is disabled in development and test, and enabled in production

- If you want to use it in development mode, add this to your environment:

```
config.action_controller.perform_caching = true
```

Rails can be configured to store cached data in a few different places:

- In memory
- Local file system
- Remote in-memory store

Rails Caching

ActiveSupport::Cache::MemoryStore

- Cached data is stored in memory, in the same address space as the ruby process and is retained between requests.
- Defaults to 32 megs, but is configurable.

ActiveSupport::Cache::FileStore

- Cached data is stored on the local file system.
- Can configure the location of the storage in Rails environment:
  - `config.cache_store = :file_store, "/path/to/cache/"`

Rails Caching

ActiveSupport::Cache::MemcacheStore

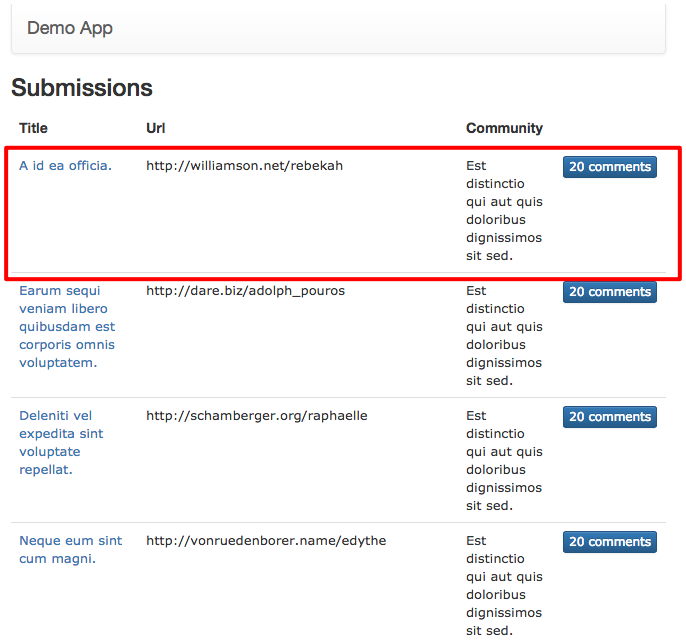
- Cached data is stored in memory on another machine.
- Can configure the location of the server in Rails environment:
  - `config.cache_store = :mem_cache_store, "cache-1.example.com"`

Rails Caching - Fragment Caching

Fragment caching caches a portion of a rendered view for reuse on future requests.

Let’s take a look at the demo app...

Rails Caching - Fragment Caching



We can cache each line of this markup.

Regardless of anything else that changes on the page, we can rerender this if it stays fresh

Rails Caching - Fragment Caching

```
    <tbody>
      <% @submissions.each do |submission| %>
        <tr>
          <td><%= link_to(submission.title, submission.url) %></td>
          <td><%= submission.url %></td>
          <td><%= submission.community.name %></td>
          <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn btn-
primary btn-xs'  %></td>
        </tr>
      <% end %>
    </tbody>
  </table>
```

Rails Caching - Fragment Caching

```
    <tbody>
      <% @submissions.each do |submission| %>
        <% cache(cache_key_for_submission_row(submission)) do %>
          <tr>
            <td><%= link_to(submission.title, submission.url) %></td>
            <td><%= submission.url %></td>
            <td><%= submission.community.name %></td>
            <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn btn-
primary btn-xs'  %></td>
          </tr>
        <% end %>
      <% end %>
    </tbody>
  </table>
```

Rails Caching - Fragment Caching

How should we choose a cache key?

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}"
  end
end
```

What are the weaknesses with the above approach?

Rails Caching - Fragment Caching

How should we choose a cache key?

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}"
  end
end
```

What are the weaknesses with the above approach?

- Invalidation will be annoying: clear out the cache on possible action causing staleness?

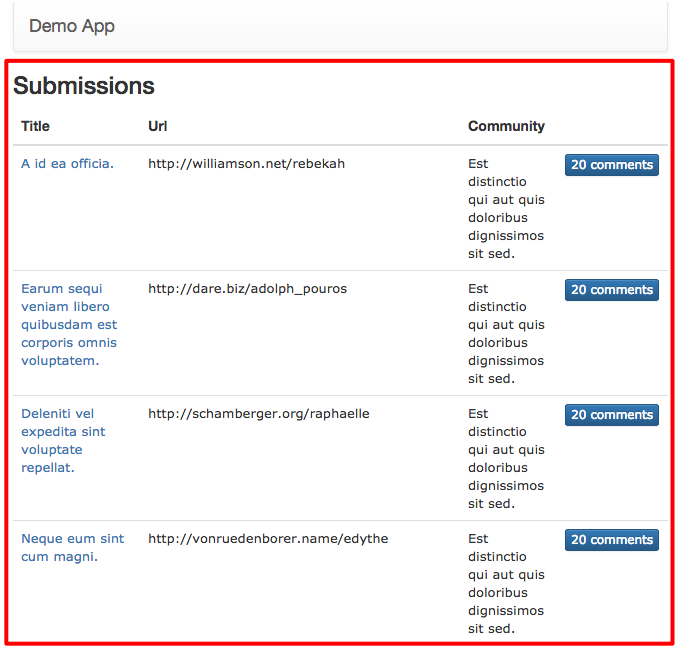
Rails Caching - Fragment Caching

Instead, let's make the key change whenever the data gets stale.

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}-#{submission.updated_at}-#{submission.comments.count}"
  end
end
```

There is no action needed to invalidate the cache: the cache key changes.

Rails Caching - Fragment Caching



If we step back and look at this page, we can observe that the whole table is expensive to compute and stays fresh for awhile.

Rails Caching - Fragment Caching

```
<h3>Submissions</h3>
<table class="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Url</th>
      <th>Community</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @submissions.each do |submission| %>
      <% cache(cache_key_for_submission_row(submission)) do %>
        <tr>
          <td><%= link_to(submission.title, submission.url) %></td>
          <td><%= submission.url %></td>
          <td><%= submission.community.name %></td>
          <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn btn-primary btn-xs'  %></td>
        </tr>
      <% end %>
    <% end %>
  </tbody>
</table>

<br>
<%= link_to 'New Submission', new_submission_path, class: 'btn btn-primary' %>
<%= link_to 'New Community', new_community_path, class: 'btn btn-primary' %>
```

Rails Caching - Fragment Caching

```
<% cache(cache_key_for_submission_table) do %>
  <h3>Submissions</h3>
  <table class="table">
    <thead>
      <tr>
        <th>Title</th>
        <th>Url</th>
        <th>Community</th>
        <th colspan="3"></th>
      </tr>
    </thead>

    <tbody>
      <% @submissions.each do |submission| %>
        <% cache(cache_key_for_submission_row(submission)) do %>
          <tr>
            <td><%= link_to(submission.title, submission.url) %></td>
            <td><%= submission.url %></td>
            <td><%= submission.community.name %></td>
            <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn btn-primary btn-xs'  %></td>
          </tr>
        <% end %>
      <% end %>
    </tbody>
  </table>

  <br>
  <%= link_to 'New Submission', new_submission_path, class: 'btn btn-primary' %>
  <%= link_to 'New Community', new_community_path, class: 'btn btn-primary' %>
<% end %>
```

Rails Caching - Fragment Caching

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}-#{submission.updated_at}-#{submission.comments.count}"
  end
  def cache_key_for_submission_table
    "submission-table-#{Submission.maximum(:updated_at)}-#{Comment.maximum(:updated_at)}"
  end
end
```

This technique of nesting cache fragments is known as “Russian Doll” caching.

Rails Caching - Low-level Caching

You can use the same mechanisms to cache anything:

```
class Product < ActiveRecord::Base
  def competing_price
    Rails.cache.fetch("#{cache_key}/competing_price", expires_in: 12.hours) do
      Competitor::API.find_price(id)
    end
  end
end
```

Rails Caching - Low-level Caching

Let’s compare the demo app’s performance!

For these tests I will compare the performance of the branch master, with the branch “server_side_caching”, which implements the caching shown in the previous slides.

Master intentionally includes no optimizations to the way we interact with the database.

We will use the default (memory) caching.

Rails Caching

We'll use an M3-medium instance with the usual workload.

When we test we will have 12 phases, of 60 seconds each:

Phase	1	2	3	4	5	6	7	8	9	10	11	12
Users/sec	1	1.5	2	4	6	10	16	20	25	35	45	55

Deployed using nginx & passenger.

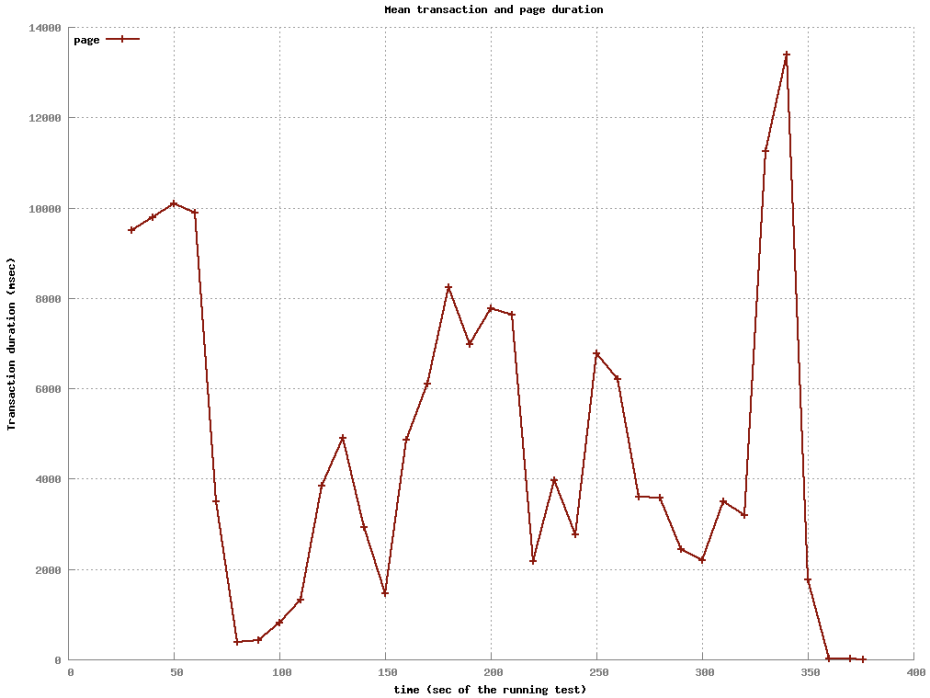
Rails Caching

We will use the same testing script from before...

1. Going to the homepage
2. Waiting for up to 2 seconds
3. Requesting a form to create a new community
4. Waiting for up to 2 seconds
5. Submitting the new community
6. Requesting a form to create a new link submission
7. Waiting for up to 2 seconds
8. Submitting the new link
9. Waiting for up to 2 seconds
10. Delete the link
11. Waiting for up to 2 seconds
12. Delete the community

Rails Caching

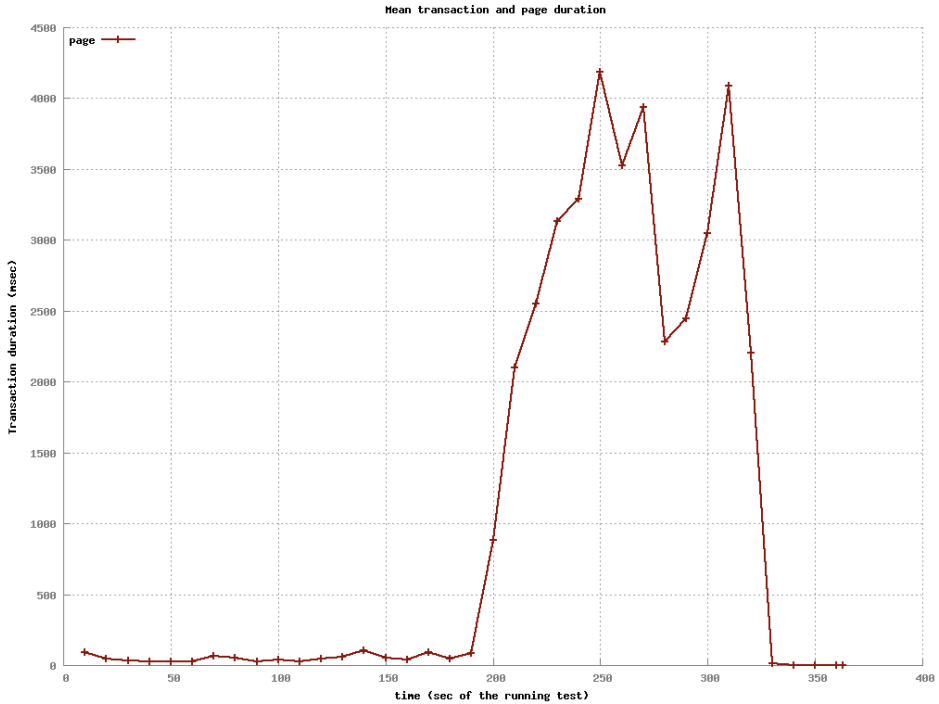
Performance on `master`:



From the start, the application can't handle a single user arriving each second

Rails Caching

Performance on `server_side_caching`:



With the server-side caching implemented, the server can handle up to two new users a second easily.

AWS Instructions

Now that you've all got a blank Rails app (or more) pushed to Github, it's time to learn how to deploy to AWS.

We will be using CloudFormation, and here are step-by-step instructions for deploying.

But first, some warnings and rules.

AWS Instructions

These are scarce resources

- Amazon has been very generous to give you all free time on their infrastructure.
- Unless you have a specific reason to do otherwise, always use micro instances.
  - Example of a good reason: testing vertical scaling.

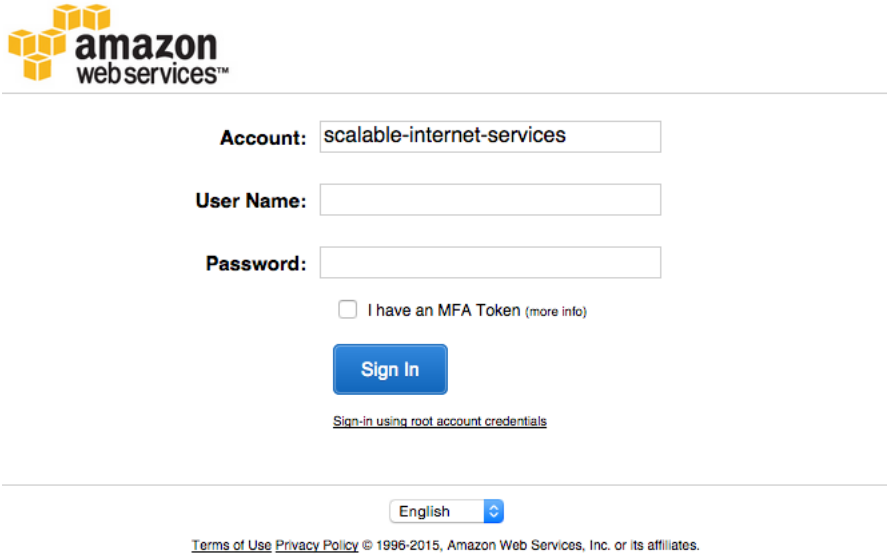
AWS Instructions

These are scarce resources

- Our AWS budget has a fixed limit.
- Whenever you are done with an instance, shut it down.
- Never keep important data on the instance, because it can go down at any time.
  - SCP important data back to your laptop.
- I will periodically run a script that terminates all instances that have been on longer than X hours.

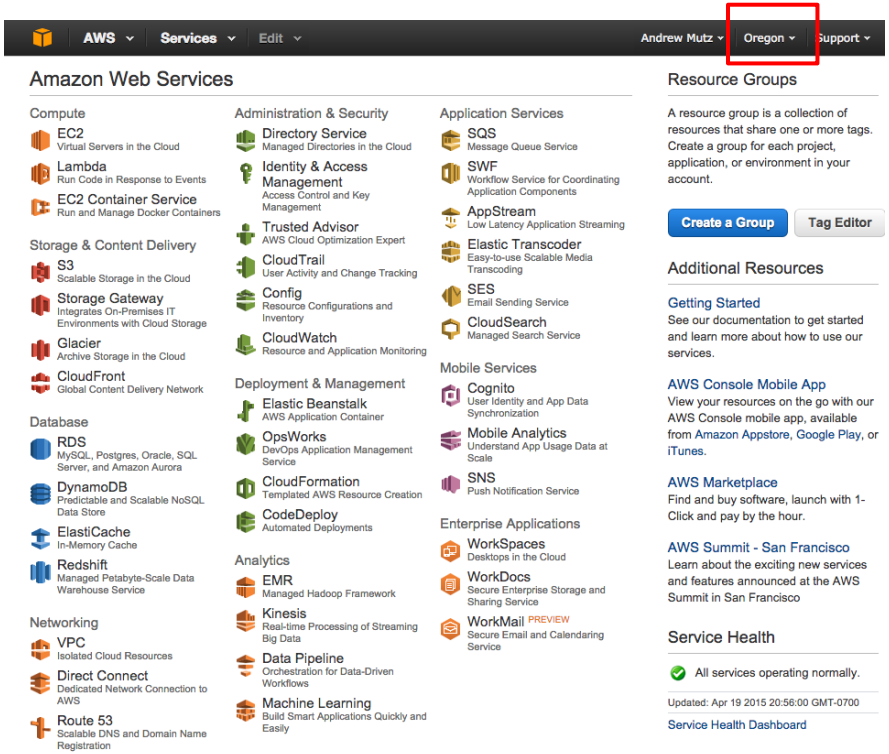
AWS Instructions

Go to https://scalable-internet-services.signin.aws.amazon.com/console



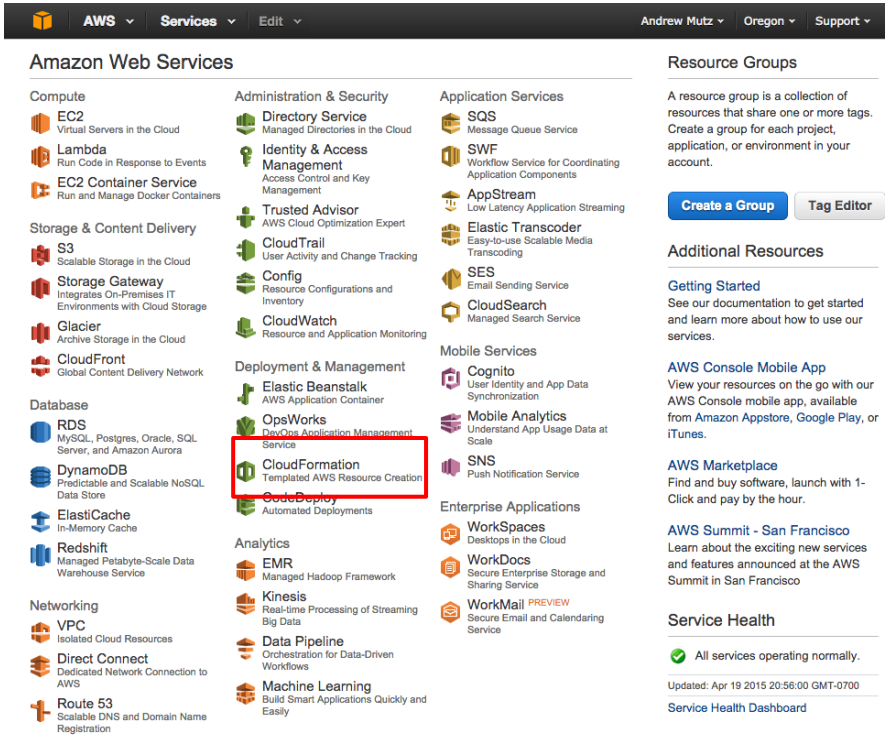
Login with Username and Password provided in email.

AWS Instructions



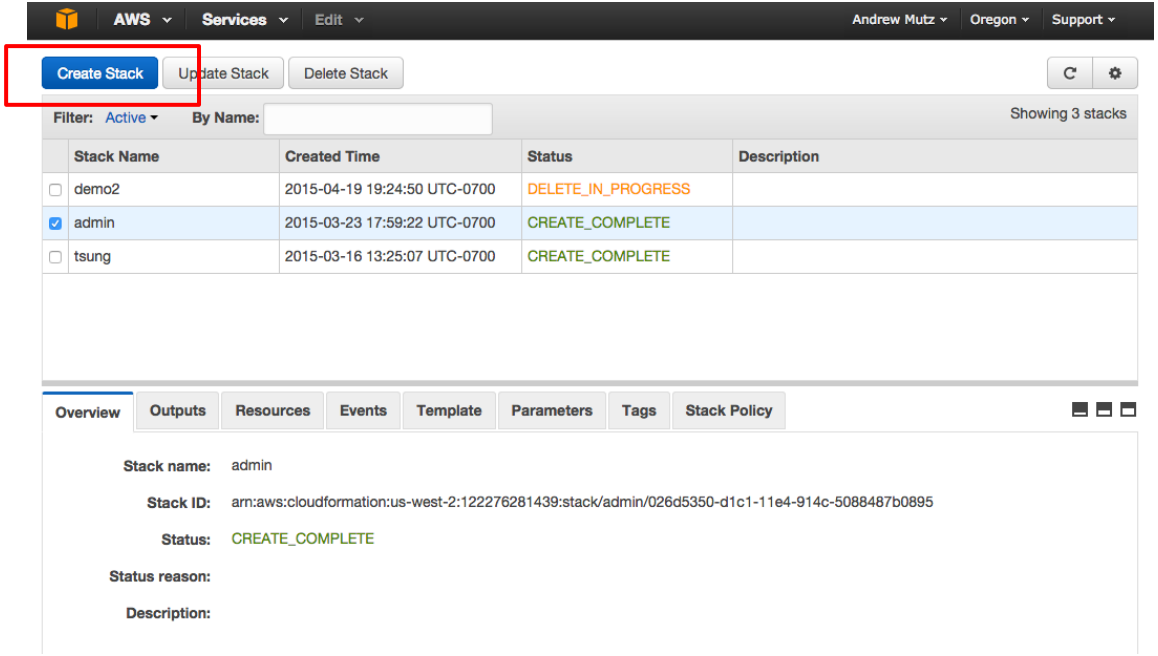
Make sure your region is set to “US-West Oregon”

AWS Instructions



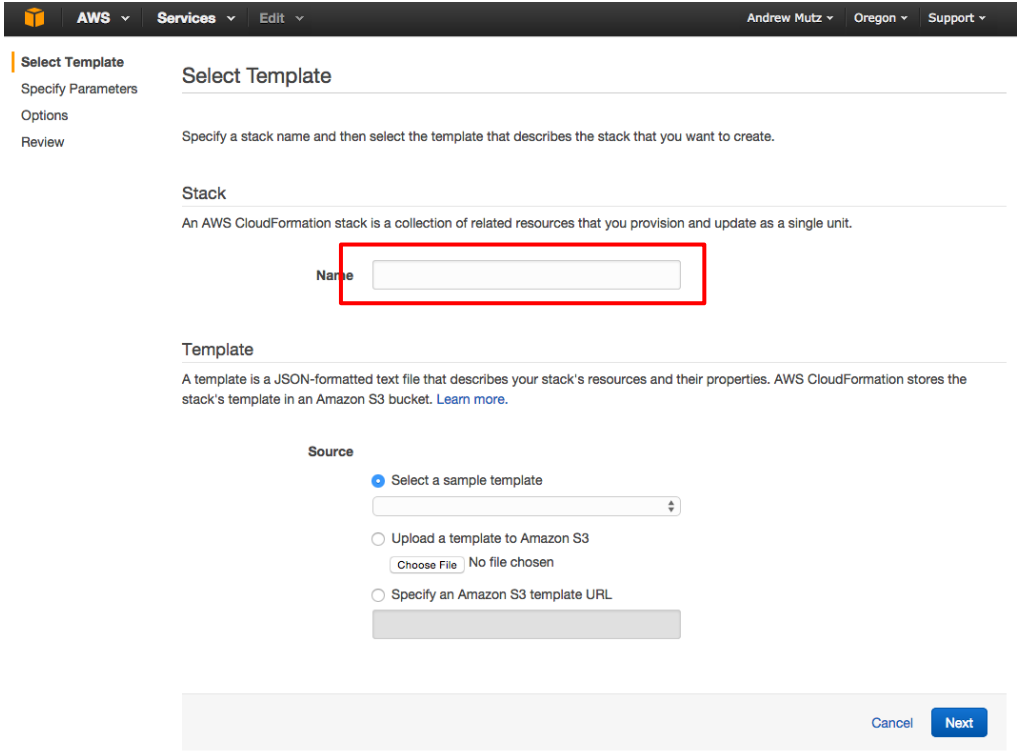
We will be deploying with cloudformation, so click on that

AWS Instructions



Click on “Create Stack”

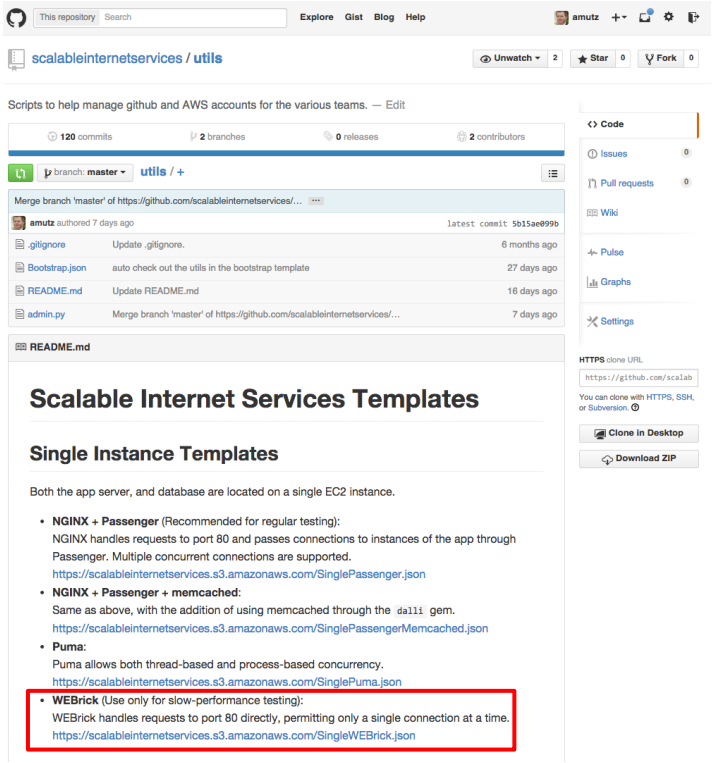
AWS Instructions



The stack name must begin with your team name.

For example: “demo-test”

AWS Instructions



Go to

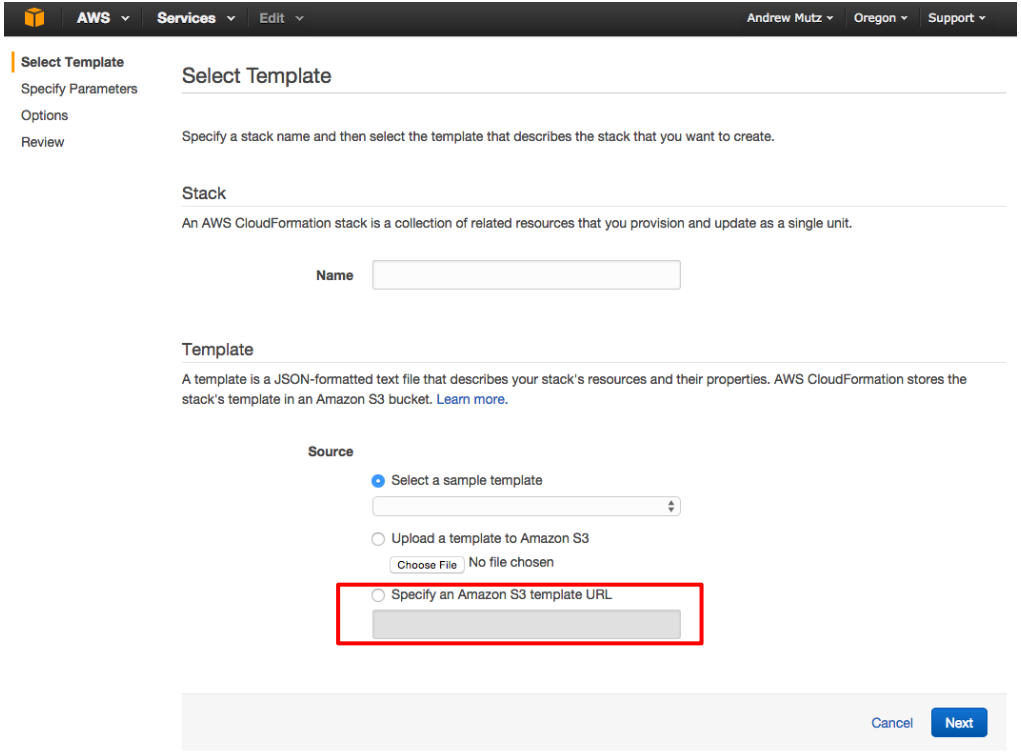
https://github.com/scalableinternetservices/utils

and choose your template.

Today just choose the Webrick template.

Copy the link.

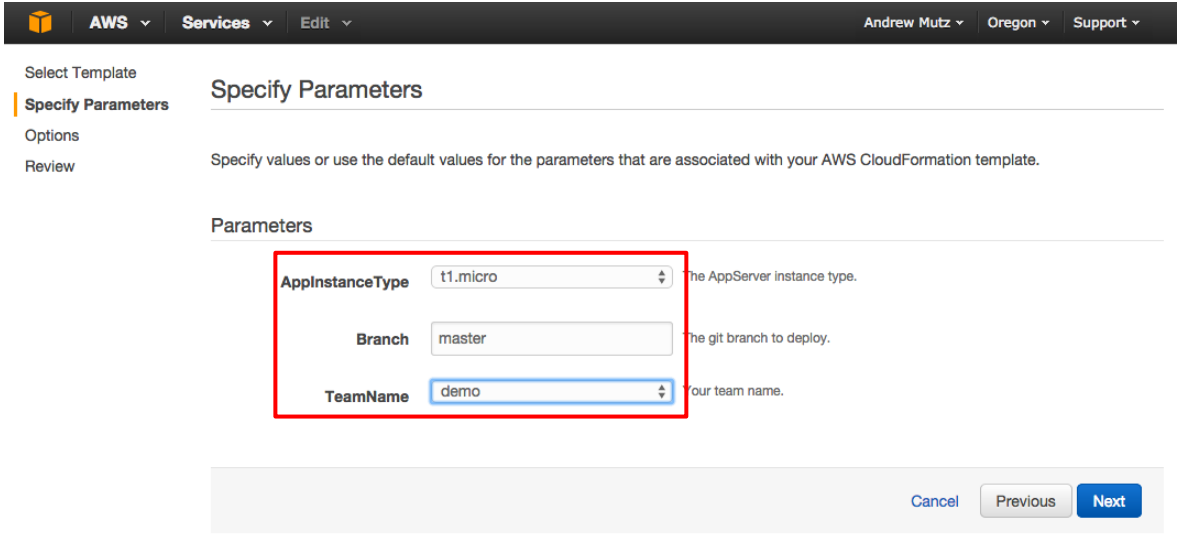
AWS Instructions



Paste that link into the field that says “Specify an Amazon S3 template URL”

Click Next.

AWS Instructions

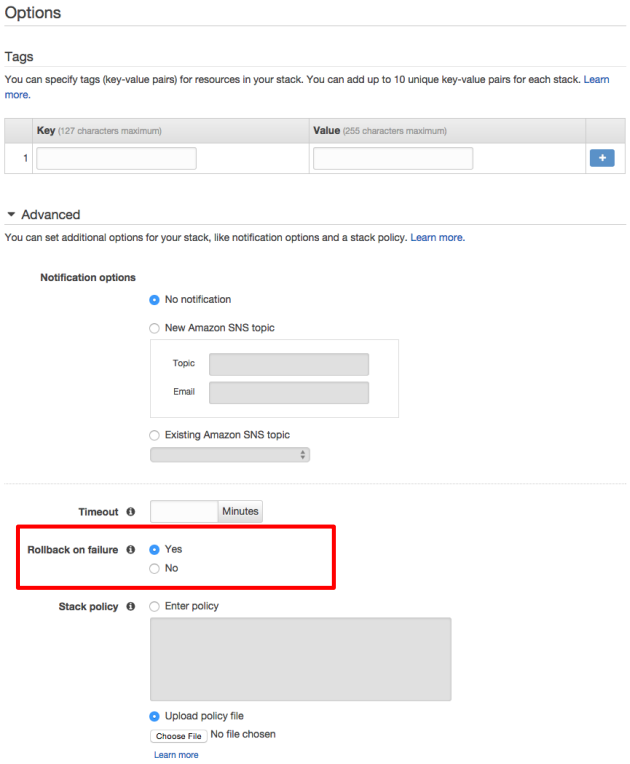


Choose your fields.

Select your TeamName from the dropdown.

The teamname is how the template knows where to get your code.

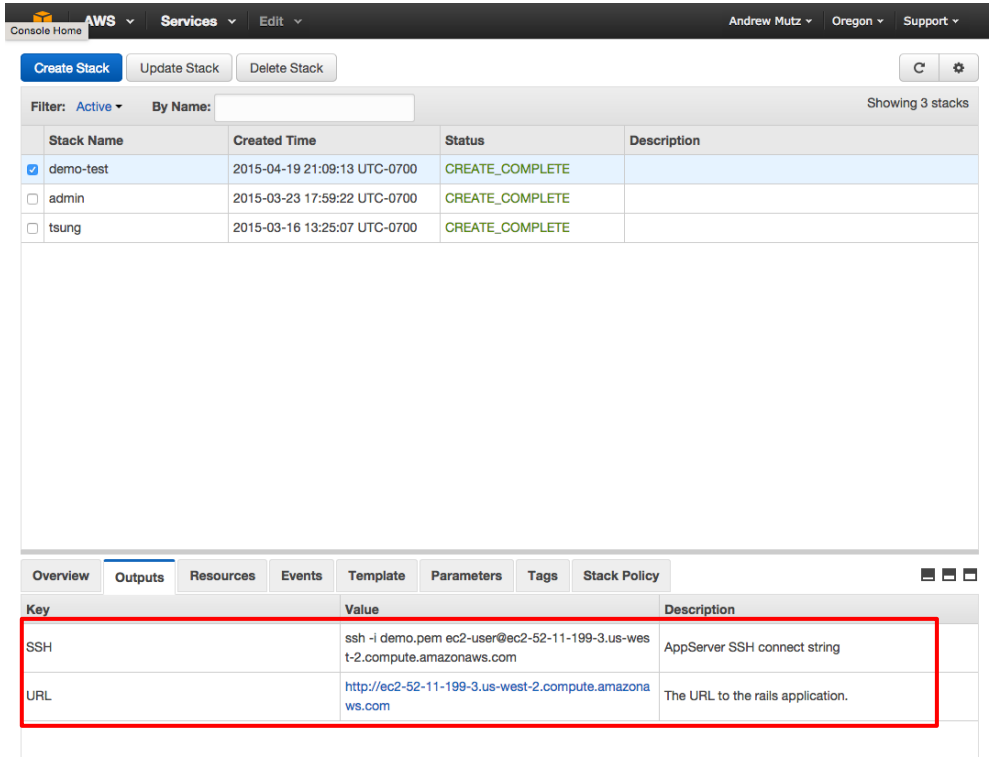
AWS Instructions



On the next screen just choose “Next”

If you are having problems deploying, you can disable rollback.

AWS Instructions

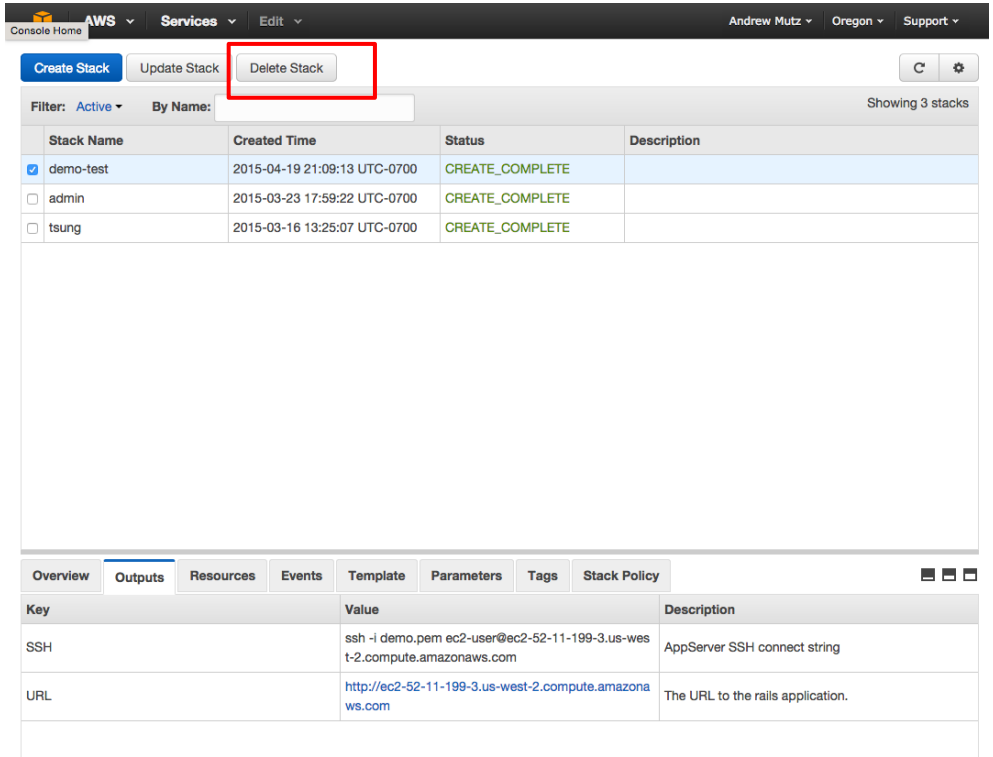


After creation, the outputs tab in the bottom pane will tell you how to reach your server via HTTP and SSH.

The PEM file mentioned in the SSH command was emailed to you.

SCP accepts the same -i FILE.pem argument

AWS Instructions



When you are done with your stack, remember to delete it!

If you have any questions, please post to Piazza!

Motivation

After today you should understand

- Why server side caching exists
- What options you have when using server side caching
- How to use this in your projects
- How to deploy on AWS using CloudFormation

For Next Time...

Try and get your app deployed on AWS. If you run into problems, post on Piazza.

Guest speaker, please be on time:

- Darren Mutz, Founder, Red Aspen Software
- “Technical Challenges in Content Delivery Networks”