

CS 188

Scalable Internet Services

Andrew Mutz

October 13, 2016



Today's Agenda

Motivation

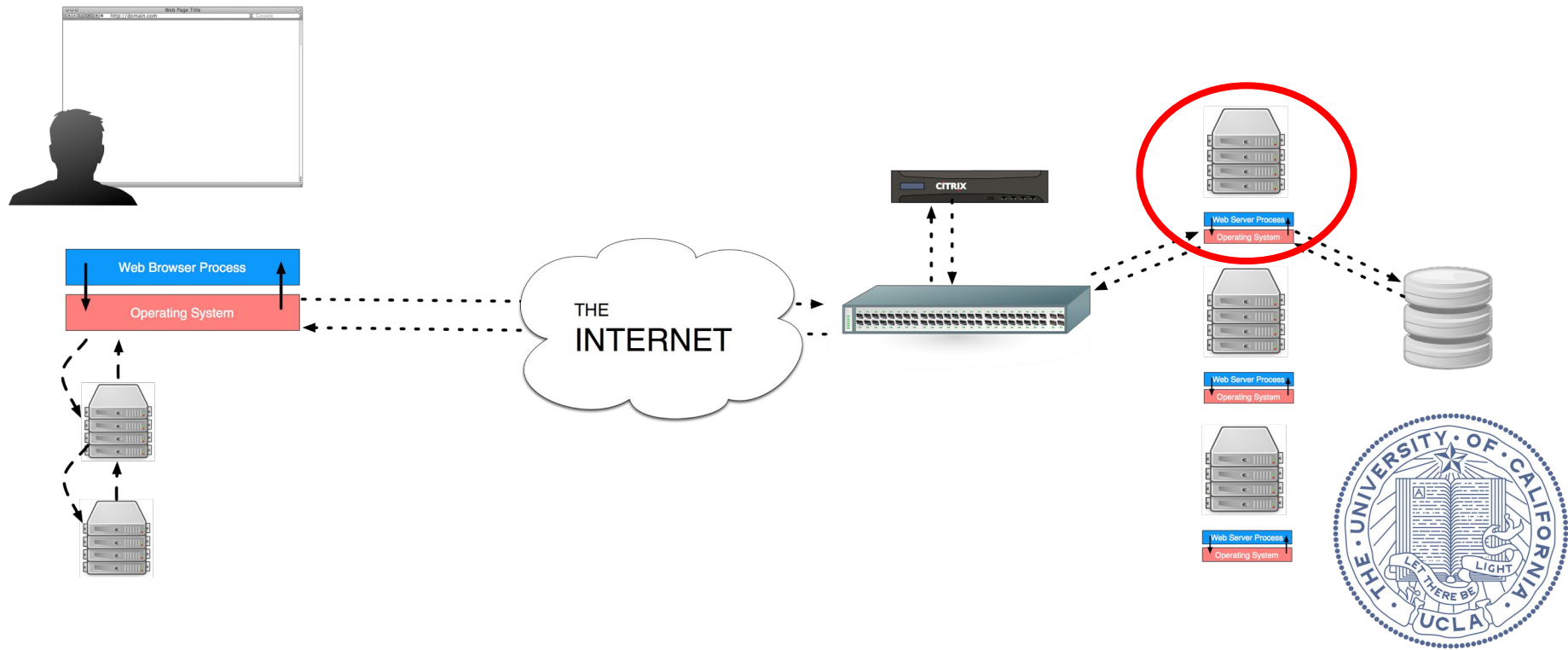
Server-side Caching

Deploying on AWS

For Next Time



Server-side Caching



Motivation

After today you should understand

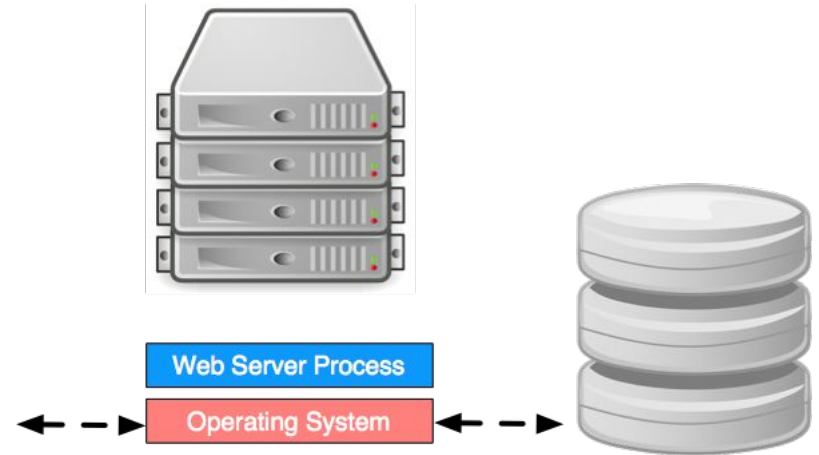
- Why server side caching exists
- What options you have when using server side caching
- How to use this in your projects
- How to deploy on AWS using CloudFormation



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.

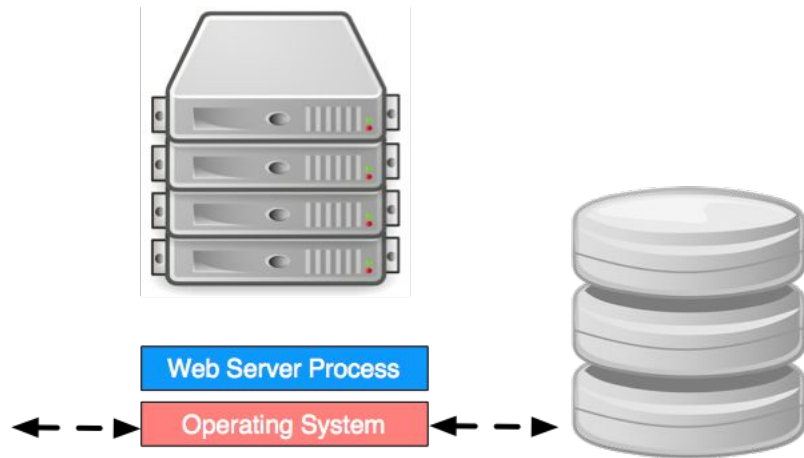


Server-side Caching

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.

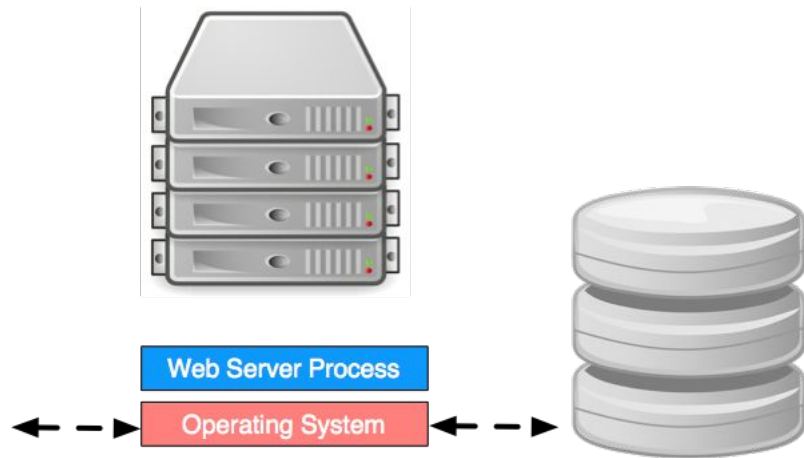


Server-side Caching

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?



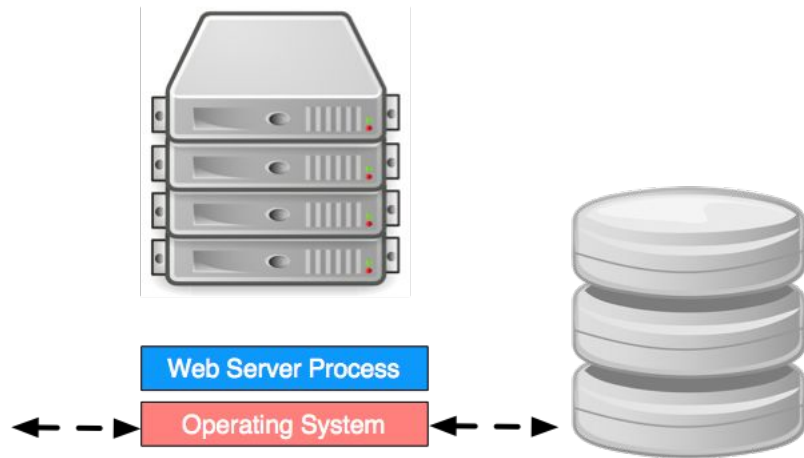
Server-side Caching

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



View Fragments: Similarity between pages

The seal of the University of California, Los Angeles (UCLA). It is a circular emblem with a blue border containing the text "THE UNIVERSITY OF CALIFORNIA" at the top and "UCLA" at the bottom. The center features a shield with a book, a sunburst, and a banner that reads "LET THERE BE LIGHT".

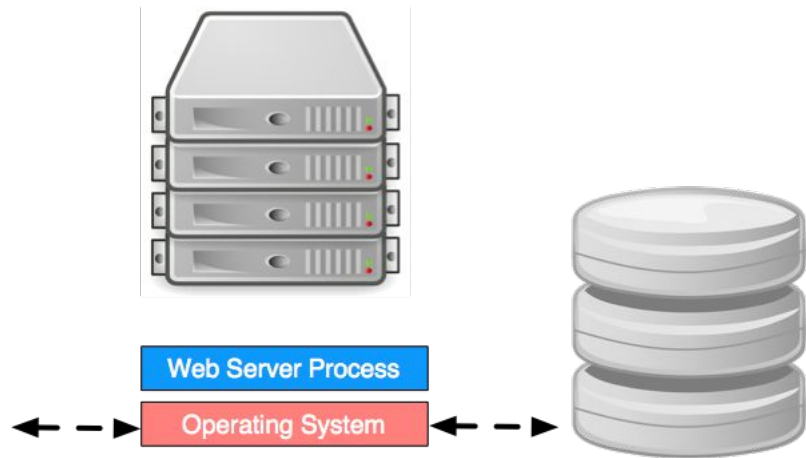
Server-side Caching

Rarely modified ORM objects?

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

Summarized data that is difficult to compute

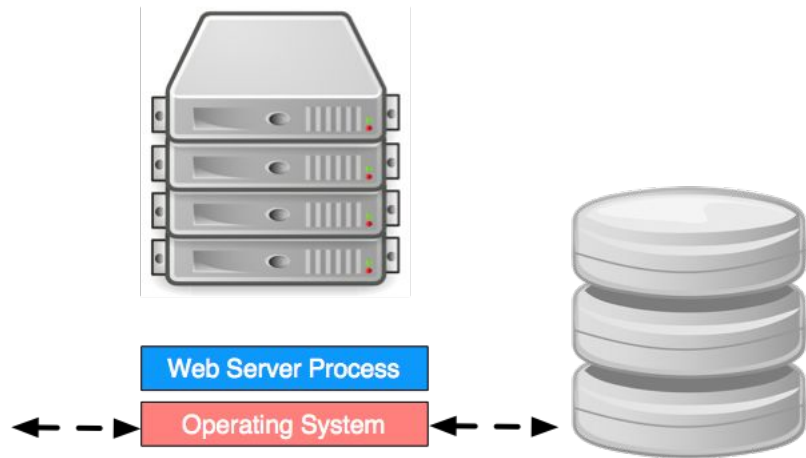
- Any particularly heavyweight SQL query
- Example: Total account balance on Mint.com



Server-side Caching

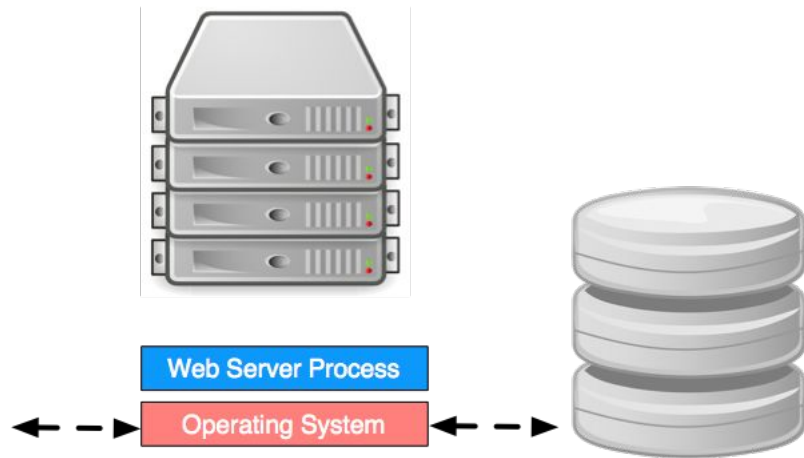
Some of these things are expensive to materialize

- 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?

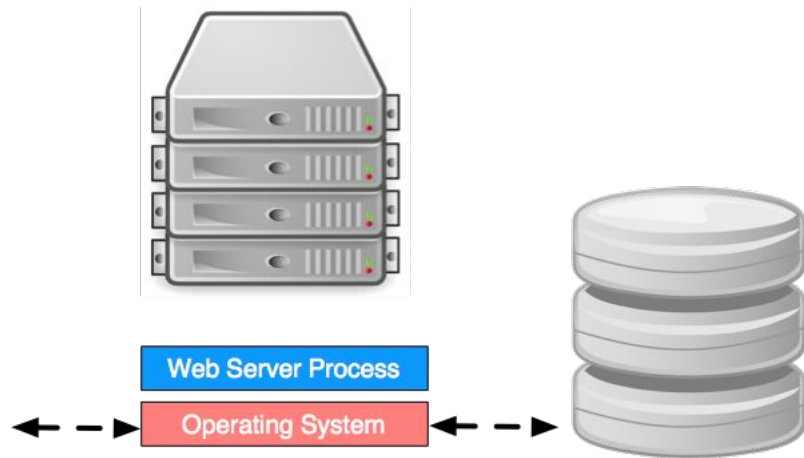


Server-side Caching

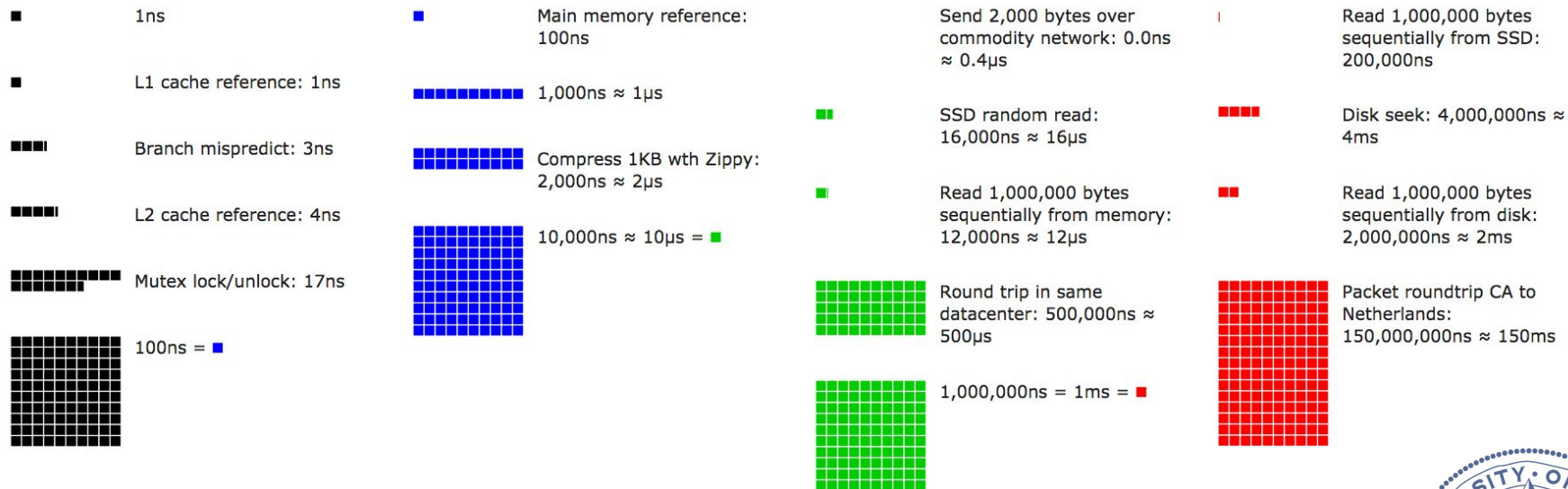
So if we want to keep previously computed results around between requests, how should we do it? Where should we put it?

- 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



Server-side Caching

What can we conclude from these numbers?

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



Server-side Caching

Summary

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

Conclusion

- Always use SSD
- Memory > local 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 local 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

| Demo App | | |
|--|-----------------------------------|--|
| Submissions | | |
| Title | Url | Community |
| A id ea officia. | http://williamson.net/rebekah | Est distinctio qui aut quis doloribus dignissimos sit sed. 20 comments |
| Earum sequi veniam libero quibusdam est corporis omnis voluptatem. | http://dare.biz/adolph_pouros | Est distinctio qui aut quis doloribus dignissimos sit sed. 20 comments |
| Deleniti vel expedita sint voluptate repellat. | http://schamberger.org/raphaelle | Est distinctio qui aut quis doloribus dignissimos sit sed. 20 comments |
| Neque eum sint cum magni. | http://vonruedenborer.name/edythe | Est distinctio qui aut quis doloribus dignissimos sit sed. 20 comments |

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

| Demo App | | | |
|--|-----------------------------------|--|-------------|
| Submissions | | | |
| Title | Url | Community | |
| A id ea officia. | http://williamson.net/rebekah | Est distinctio qui aut quis doloribus dignissimos sit sed. | 20 comments |
| Earum sequi veniam libero quibusdam est corporis omnis voluptatem. | http://dare.biz/adolph_pouros | Est distinctio qui aut quis doloribus dignissimos sit sed. | 20 comments |
| Deleniti vel expedita sint voluptate repellat. | http://schamberger.org/raphaele | Est distinctio qui aut quis doloribus dignissimos sit sed. | 20 comments |
| Neque eum sint cum magni. | http://vonruedenborer.name/edythe | Est distinctio qui aut quis doloribus dignissimos sit sed. | 20 comments |

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

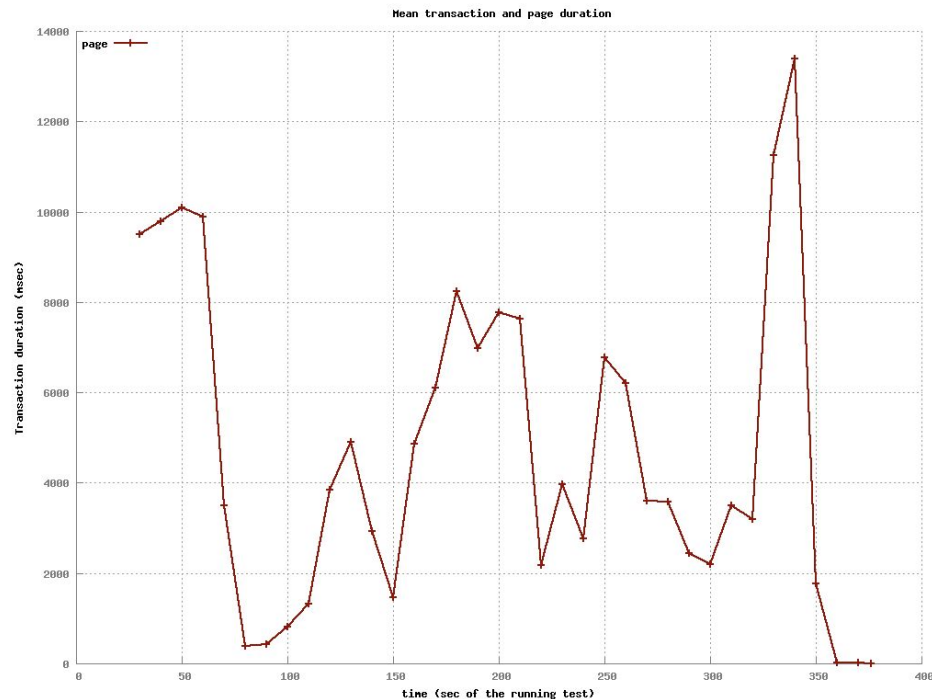
We will use the same testing script from before...

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:

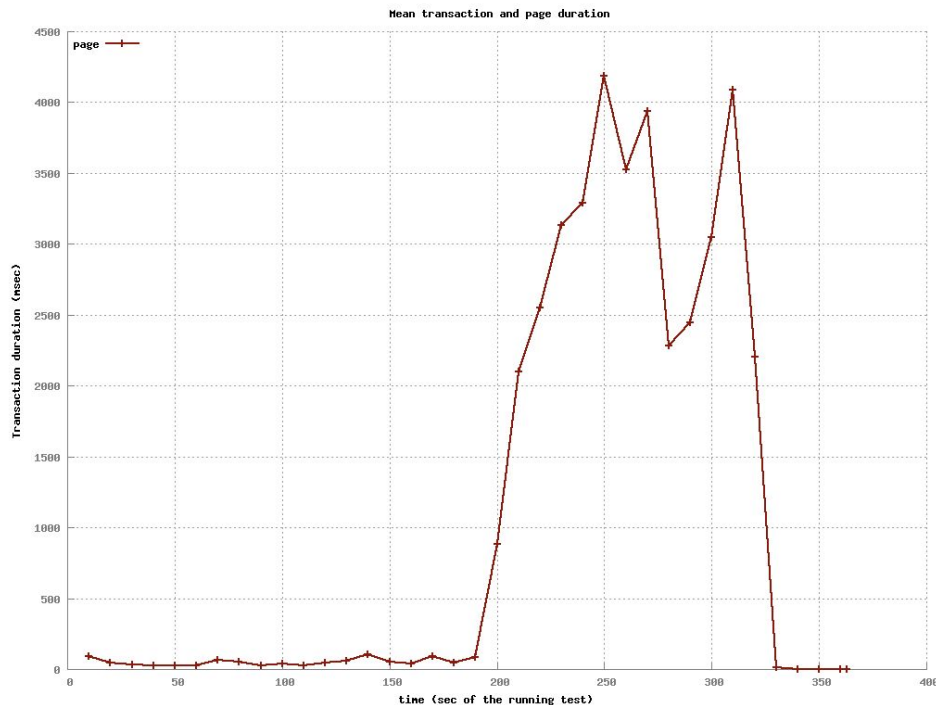


From the start, the application can't handle a single user arriving each second



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

- This is free time on Amazon's infrastructure, and it's not unlimited.
- 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 up longer than 8 hours.



AWS Instructions

Treat these credentials as secrets.

- Do not check into any publicly accessible repository
- There are automated scripts that actively seek out AWS credentials
 - Why?



AWS Instructions

Today you will receive an email with AWS credentials. These include:

- A.txt file
 - Username: what you use to log in to the AWS web interface
 - Password: the password you use to log in to the AWS web interface
- A.pem file
 - Contains a private key that you will use to ssh into the instances that you launch.



AWS Instructions

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



Account:

User Name:

Password:

☐ I have an MFA Token [\(more info\)](#)

Sign In

[Sign-in using root account credentials](#)

English

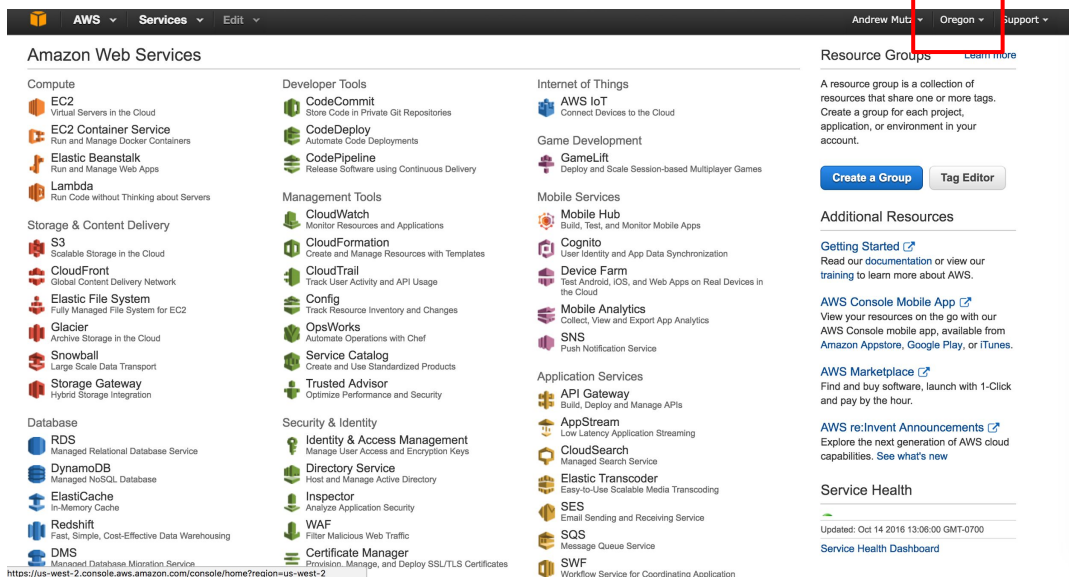
[Terms of Use](#) [Privacy Policy](#) © 1996-2015, Amazon Web Services, Inc. or its affiliates.

Login with Username and Password provided in txt file in email.



AWS Instructions

Make sure your region is set to
“US-West-2 Oregon”



The screenshot shows the AWS Management Console interface. At the top, the navigation bar includes the AWS logo, a dropdown menu for 'Services', an 'Edit' button, and a user profile section for 'Andrew Mub'. The 'Oregon' region is selected in the dropdown menu, which is highlighted with a red rectangle. Below the navigation bar, the 'Amazon Web Services' section displays a grid of service categories: Compute (EC2, EC2 Container Service, Elastic Beanstalk, Lambda), Storage & Content Delivery (S3, CloudFront, Elastic File System, Glacier, Snowball, Storage Gateway), Database (RDS, DynamoDB, ElastiCache, Redshift, DMS), Developer Tools (CodeCommit, CodeDeploy, CodePipeline), Management Tools (CloudWatch, CloudFormation, CloudTrail, Config, OpsWorks, Service Catalog, Trusted Advisor), Security & Identity (Identity & Access Management, Directory Service, Inspector, WAF, Certificate Manager), Internet of Things (AWS IoT), Game Development (GameLift), Mobile Services (Mobile Hub, Cognito, Device Farm, Mobile Analytics, SNS), and Application Services (API Gateway, AppStream, CloudSearch, Elastic Transcoder, SES, SQS, SWF). On the right side, the 'Resource Groups' section is visible, along with 'Additional Resources' and 'Service Health' sections. The URL at the bottom of the browser window is <https://us-west-2.console.aws.amazon.com/console/home?region=us-west-2>.



AWS Instructions

We will be deploying with cloudformation, so click on that

The screenshot shows the AWS Management Console interface. The top navigation bar includes the AWS logo, 'Services', 'Edit', and user information (Andrew Mutz, Oregon, Support). The main content area is divided into three columns. The left column lists various AWS services under categories like Compute, Storage & Content Delivery, Database, etc. The middle column lists services under categories like Developer Tools, Management Tools, Security & Identity, etc. The right column shows 'Resource Groups' and 'Additional Resources'. The 'CloudFormation' service is highlighted with a red rectangular box in the 'Management Tools' category. Below the screenshot, the URL 'https://us-west-2.console.aws.amazon.com/console/home?region=us-west-2' is visible.

Amazon Web Services

Andrew Mutz Oregon Support

Compute

- EC2 Virtual Servers in the Cloud
- EC2 Container Service Run and Manage Docker Containers
- Elastic Beanstalk Run and Manage Web Apps
- Lambda Run Code without Thinking about Servers

Storage & Content Delivery

- S3 Scalable Storage in the Cloud
- CloudFront Global Content Delivery Network
- Elastic File System Fully Managed File System for EC2
- Glacier Archive Storage in the Cloud
- Snowball Large Scale Data Transport
- Storage Gateway Hybrid Storage Integration

Database

- RDS Managed Relational Database Service
- DynamoDB Managed NoSQL Database
- ElastiCache In-Memory Cache
- Redshift Fast, Simple, Cost-Effective Data Warehousing
- DMS Managed Database Migration Service

Developer Tools

- CodeCommit Store Code in Private Git Repositories
- CodeDeploy Automate Code Deployments
- CodePipeline Release Software using Continuous Delivery

Management Tools

- CloudWatch Monitor Resources and Applications
- CloudFormation Create and Manage Resources with Templates**
- CloudTrail CloudTrail Activity and API Calls
- Config Track Resource Inventory and Changes
- OpsWorks Automate Operations with Chef
- Service Catalog Create and Use Standardized Products
- Trusted Advisor Optimize Performance and Security

Security & Identity

- Identity & Access Management Manage User Access and Encryption Keys
- Directory Service Host and Manage Active Directory
- Inspector Analyze Application Security
- WAF Filter Malicious Web Traffic
- Certificate Manager Provision, Manage, and Deploy SSL/TLS Certificates

Internet of Things

- AWS IoT Connect Devices to the Cloud

Game Development

- GameLift Deploy and Scale Session-based Multiplayer Games

Mobile Services

- Mobile Hub Build, Test, and Monitor Mobile Apps
- Cognito User Identity and App Data Synchronization
- Device Farm Test Android, iOS, and Web Apps on Real Devices in the Cloud
- Mobile Analytics Collect, View and Export App Analytics
- SNS Push Notification Service

Application Services

- API Gateway Build, Deploy and Manage APIs
- AppStream Low Latency Application Streaming
- CloudSearch Managed Search Service
- Elastic Transcoder Easy-to-Use Scalable Media Transcoding
- SES Email Sending and Receiving Service
- SQS Message Queue Service
- SWF Workflow Service for Coordinating Application

Resource Groups

A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.

Create a Group Tag Editor

Additional Resources

Getting Started

Read our documentation or view our training to learn more about AWS.

AWS Console Mobile App

View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.

AWS Marketplace

Find and buy software, launch with 1-Click and pay by the hour.

AWS re:Invent Announcements

Explore the next generation of AWS cloud capabilities. See what's new

Service Health

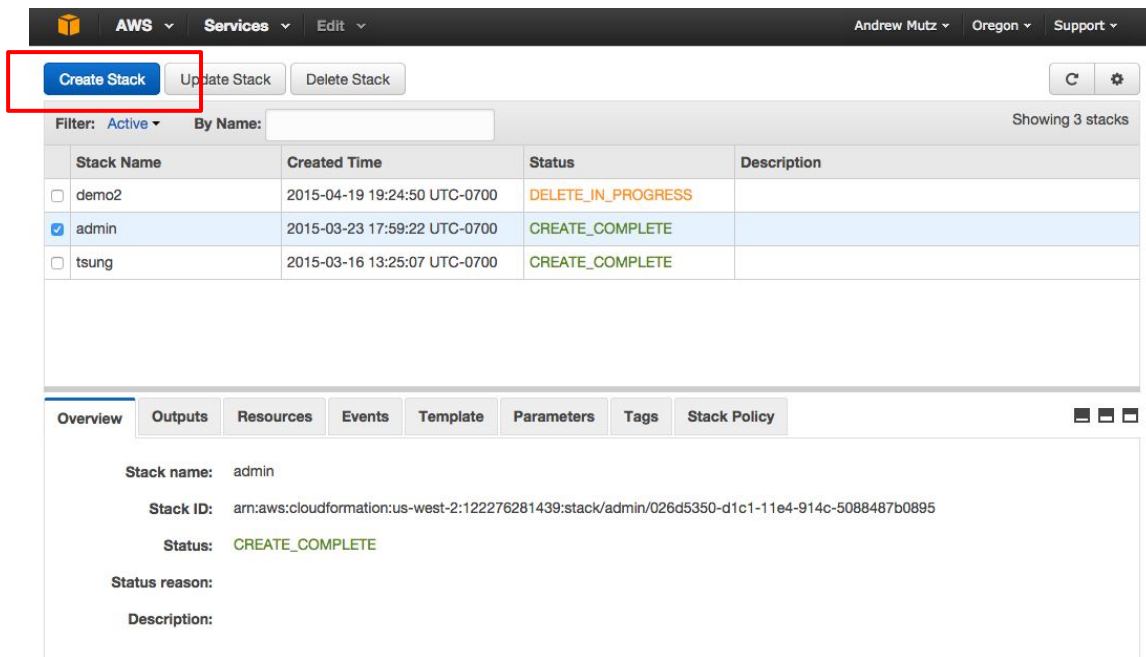
Updated: Oct 14 2016 13:06:00 GMT-0700

Service Health Dashboard

https://us-west-2.console.aws.amazon.com/console/home?region=us-west-2



AWS Instructions



The screenshot shows the AWS CloudFormation console interface. At the top, there's a navigation bar with 'AWS', 'Services', and 'Edit' dropdowns, along with user information 'Andrew Mutz', region 'Oregon', and 'Support'. Below this, there are three buttons: 'Create Stack' (highlighted with a red box), 'Update Stack', and 'Delete Stack'. To the right of these buttons are icons for refresh and settings. Below the buttons, there's a filter section with 'Filter: Active' and a search box 'By Name:'. A table lists three stacks:

| Stack Name | Created Time | Status | Description |
|---|------------------------------|--------------------|-------------|
| <input type="checkbox"/> demo2 | 2015-04-19 19:24:50 UTC-0700 | DELETE_IN_PROGRESS | |
| <input checked="" type="checkbox"/> admin | 2015-03-23 17:59:22 UTC-0700 | CREATE_COMPLETE | |
| <input type="checkbox"/> tsung | 2015-03-16 13:25:07 UTC-0700 | CREATE_COMPLETE | |


Below the table, there's a section for the selected stack 'admin'. It includes tabs for 'Overview', 'Outputs', 'Resources', 'Events', 'Template', 'Parameters', 'Tags', and 'Stack Policy'. The 'Overview' tab is active, showing details for the stack 'admin':

- Stack name: admin
- Stack ID: arn:aws:cloudformation:us-west-2:122276281439:stack/admin/026d5350-d1c1-11e4-914c-5088487b0895
- Status: CREATE_COMPLETE
- Status reason:
- Description:

Click on “Create Stack”



AWS Instructions

 AWS

Services

Edit

demo @ scalable-Internet-servi... Oregon Support

Select Template

Specify Details

Options

Review

Select Template

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

Design a template

Use AWS CloudFormation Designer to create or modify an existing template. [Learn more.](#)

Design template

Choose a template

A template is a JSON-formatted text file that describes your stack's resources and their properties. [Learn more.](#)

☒ Select a sample template

☐ Upload a template to Amazon S3

☐ Specify an Amazon S3 template URL

Choose Fileno file selected








Cancel

Next

We want to specify an Amazon S3 template URL.



AWS Instructions

| | | |
|--|---|-----------------------------------|
|  bboe | Fix issue with constant import. | Latest commit a3e0f1d 27 days ago |
|  scalable_admin | Fix issue with constant import. | 27 days ago |
|  .gitignore | Complete module to package transition. | a year ago |
|  Bootstrap.json | fixing bootstrap.json | a year ago |
|  MANIFEST.in | Extract INIT segments into their own files. | a year ago |
|  README.md | Fix minor formatting issues. | 27 days ago |
|  setup.py | Fix minor formatting issues. | 27 days ago |

 README.md

Scalable Internet Services Templates

Single Instance Templates

Both the app server, and database are located on a single EC2 instance.

- **NGINX + Passenger** (Recommended for regular testing):
NGINX handles requests to port 80 and passes connections to instances of the app through Passenger. Multiple concurrent connections are supported.
 - (UCLA) <https://scalableinternetservices.s3.amazonaws.com/SinglePassenger.json>
 - (UCSB) <https://cs290b.s3.amazonaws.com/SinglePassenger.json>
- **NGINX + Passenger + memcached**:
Same as above, with the addition of using memcached through the `dalli` gem.
 - (UCLA) <https://scalableinternetservices.s3.amazonaws.com/SinglePassengerMemcached.json>
 - (UCSB) <https://cs290b.s3.amazonaws.com/SinglePassengerMemcached.json>

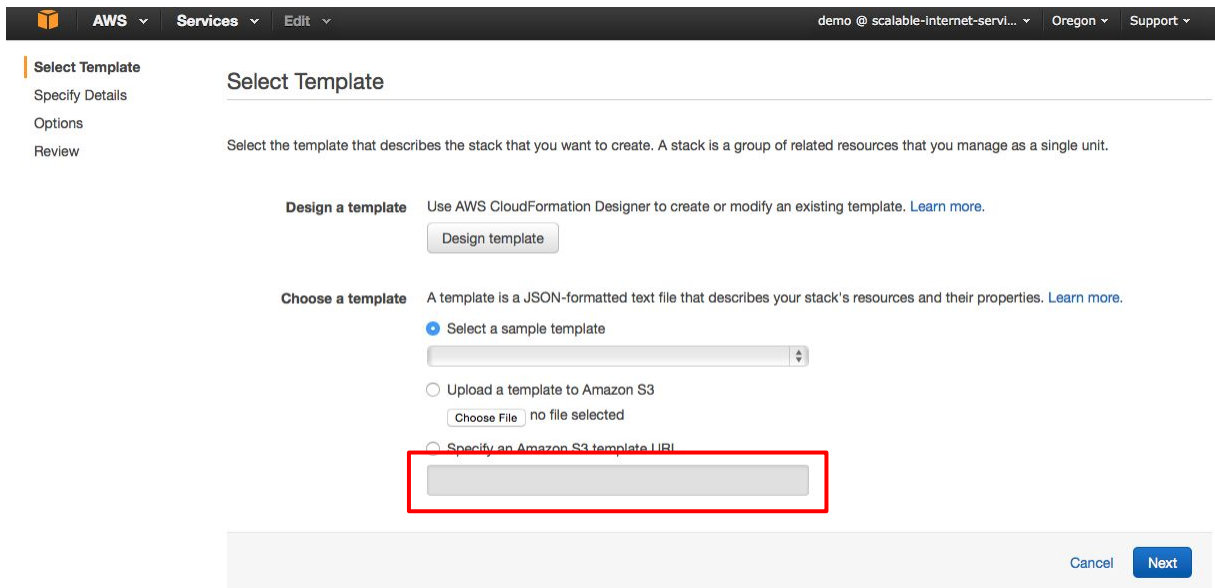
Go to
<https://github.com/scalableinternetservices/utls>
and choose your template.

Today just choose the UCLA passenger template.

Copy the link.



AWS Instructions



AWS ▾ **Services** ▾ **Edit** ▾ demo @ scalable-Internet-servi... ▾ Oregon ▾ Support ▾

Select Template

Specify Details
Options
Review

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

Design a template Use AWS CloudFormation Designer to create or modify an existing template. [Learn more.](#)

Design template

Choose a template A template is a JSON-formatted text file that describes your stack's resources and their properties. [Learn more.](#)

☒ Select a sample template

☐ Upload a template to Amazon S3

Choose File no file selected

☐ Specify an Amazon S3 template URL

Cancel Next

Paste the link to the template in the field labeled “Specify an Amazon S3 Template URL”



AWS Instructions

AWS Services Edit demo @ scalable-internet-servi... Oregon Support

Select Template
Specify Details
Options
Review

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

AppInstanceType The AppServer instance type.

Branch The git branch to deploy.

TeamName Your team name.

▲ Parameters with AllowedValues must not be empty

▲ TeamName has an invalid value.

Cancel Previous Next

The stack name must begin with your team name.

For example: “demo-test”



AWS Instructions

AWS Services Edit demo @ scalable-internet-servi... Oregon Support

Select Template
Specify Details
Options
Review

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

AppInstanceType The AppServer instance type.

Branch The git branch to deploy.

TeamName Your team name.

▲ Parameters with AllowedValues must not be empty

▲ TeamName has an invalid value.

Cancel Previous Next

Choose your fields.

Select your TeamName from the dropdown.

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



AWS Instructions

▼ Advanced

You can set additional options for your stack, like notification options and a stack policy. [Learn more.](#)

Notification options

☒ No notification

☐ New Amazon SNS topic

Topic

Email

☐ Existing Amazon SNS topic

☐ Existing topic ARN

Timeout ⓘ

Minutes

Rollback on failure ⓘ

☐ Yes

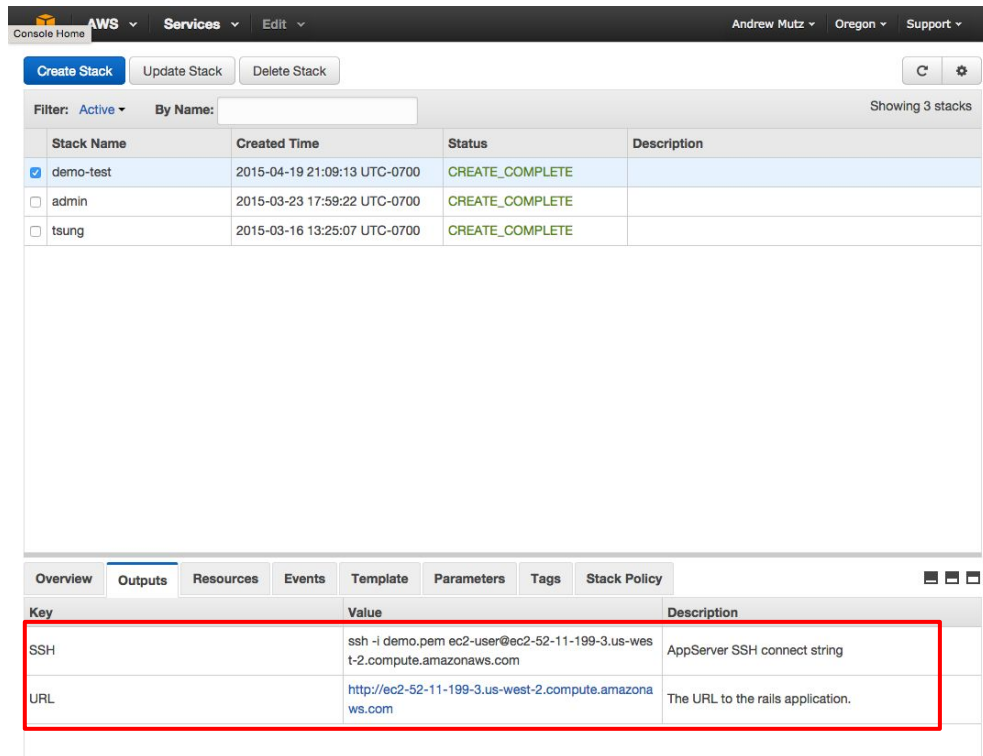
☒ No

On the next screen just choose “Next”

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



AWS Instructions



The screenshot shows the AWS Management Console interface. At the top, there's a navigation bar with 'AWS', 'Services', and 'Edit' dropdowns. Below that, a header bar shows 'Console Home', 'Andrew Mutz', 'Oregon', and 'Support'. The main content area has buttons for 'Create Stack', 'Update Stack', and 'Delete Stack'. A filter section shows 'Filter: Active' and 'By Name:'. Below this is a table of stacks:

| Stack Name | Created Time | Status | Description |
|---|------------------------------|-----------------|-------------|
| <input checked="" type="checkbox"/> demo-test | 2015-04-19 21:09:13 UTC-0700 | CREATE_COMPLETE | |
| <input type="checkbox"/> admin | 2015-03-23 17:59:22 UTC-0700 | CREATE_COMPLETE | |
| <input type="checkbox"/> tsung | 2015-03-16 13:25:07 UTC-0700 | CREATE_COMPLETE | |

Below the stack table, there are tabs for 'Overview', 'Outputs', 'Resources', 'Events', 'Template', 'Parameters', 'Tags', and 'Stack Policy'. The 'Outputs' tab is selected, showing a table of outputs:

| Key | Value | Description |
|-----|--|-----------------------------------|
| SSH | ssh -i demo.pem ec2-user@ec2-52-11-199-3.us-west-2.compute.amazonaws.com | AppServer SSH connect string |
| URL | http://ec2-52-11-199-3.us-west-2.compute.amazonaws.com | The URL to the rails application. |

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

How do I SSH into my instance?

- `ssh -i [your pemfile here] ec2-user@ec2-something.us-west-2.compute.amazonaws.com`

How do I copy files to/from my instance?

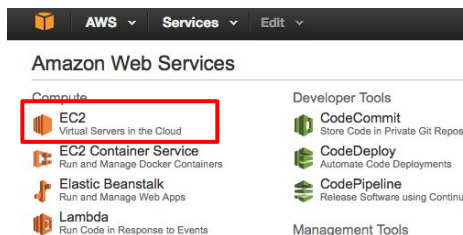
- `scp -i [your pemfile here] ec2-user@ec2-something.us-west-2.compute.amazonaws.com:fromfile tofile`



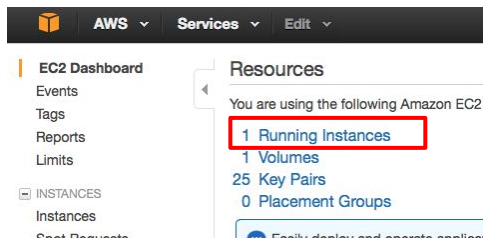
AWS Instructions

My cloudformation stack failed to deploy. How do I debug this?

1. Go to the AWS dashboard and select EC2



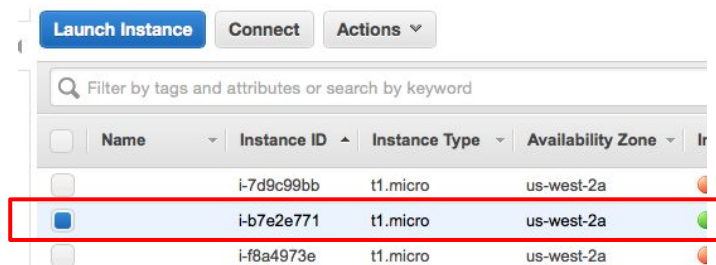
2. Click on “X Running Instances”



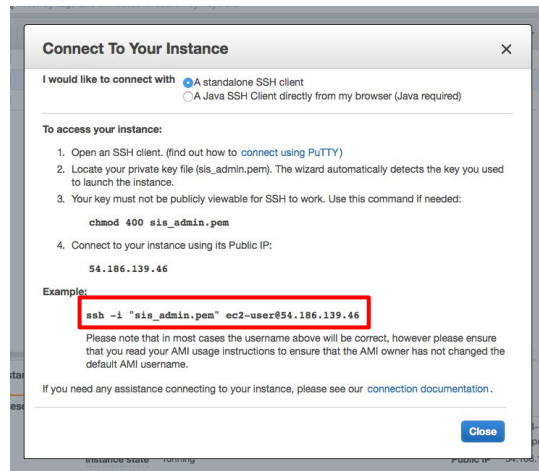
AWS Instructions

My cloudformation stack failed to deploy. How do I debug this?

3. Select the instance corresponding to your team



4. Click the connect button and copy and paste the ssh command.



AWS Instructions

My cloudformation stack failed to deploy. How do I debug this?

Once you have SSHed into your instance, you can find details of problems at:

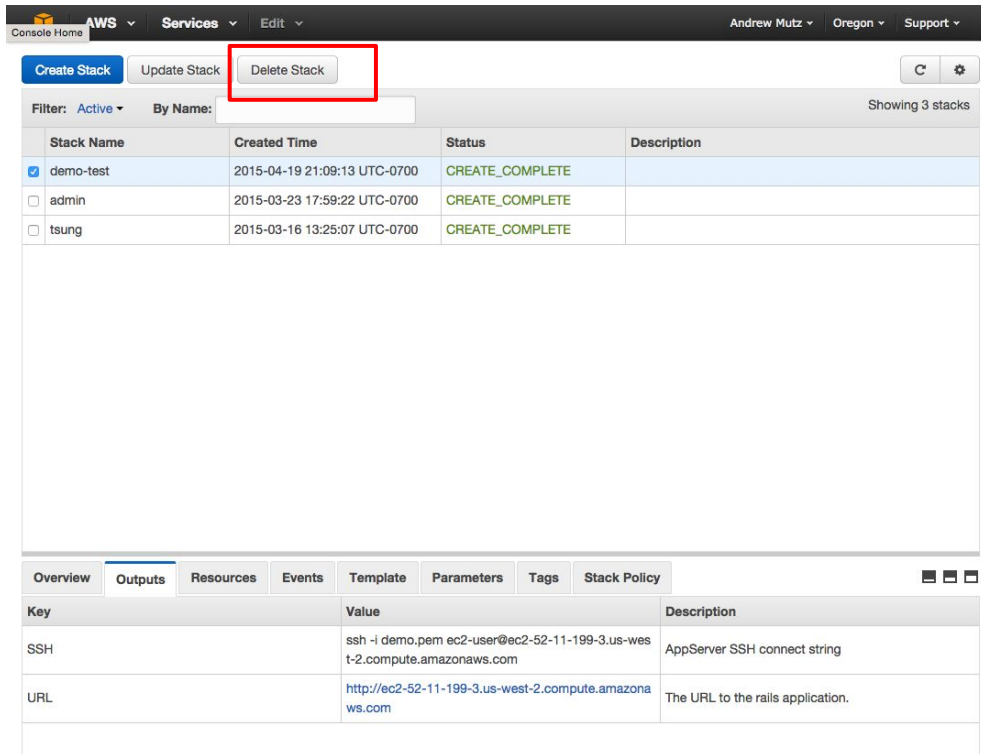
```
/var/log/cloud-init-output.log
```

You can also read the script that was executed on CF startup here:

```
/var/lib/cloud/instance/scripts/part-001
```



AWS Instructions



The screenshot shows the AWS Management Console interface. At the top, there's a navigation bar with 'AWS', 'Services', and 'Edit' dropdowns, along with user information 'Andrew Mutz', region 'Oregon', and 'Support'. Below this, there are three buttons: 'Create Stack', 'Update Stack', and 'Delete Stack'. The 'Delete Stack' button is highlighted with a red rectangular box. Below the buttons, there's a filter section with 'Filter: Active' and 'By Name:' followed by a search input field. A table lists three stacks: 'demo-test', 'admin', and 'tsung'. The 'demo-test' stack is selected with a blue checkbox. The table columns are 'Stack Name', 'Created Time', 'Status', and 'Description'. Below the table, there's a section for 'Outputs' with tabs for 'Overview', 'Outputs', 'Resources', 'Events', 'Template', 'Parameters', 'Tags', and 'Stack Policy'. The 'Outputs' tab is active, showing a table with columns 'Key', 'Value', and 'Description'. The table lists two outputs: 'SSH' and 'URL'.

| Stack Name | Created Time | Status | Description |
|---|------------------------------|-----------------|-------------|
| <input checked="" type="checkbox"/> demo-test | 2015-04-19 21:09:13 UTC-0700 | CREATE_COMPLETE | |
| <input type="checkbox"/> admin | 2015-03-23 17:59:22 UTC-0700 | CREATE_COMPLETE | |
| <input type="checkbox"/> tsung | 2015-03-16 13:25:07 UTC-0700 | CREATE_COMPLETE | |

| Key | Value | Description |
|-----|--|-----------------------------------|
| SSH | ssh -i demo.pem ec2-user@ec2-52-11-199-3.us-west-2.compute.amazonaws.com | AppServer SSH connect string |
| URL | http://ec2-52-11-199-3.us-west-2.compute.amazonaws.com | The URL to the rails application. |

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...

Continue to work on sprint 1 stories. We will demo your progress at tomorrow's lab.

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

