CS188
Scalable Internet Services

John Rothfels, 12/1/20

Reminder of lectures

- Today: review scaling toolset, AWS deployment
- Next 2 lectures: bonus topics (NoSQL, CDNs, beyond JavaScript in the browser)
  - House reveal?
  - Extremely limited edition CS188 plant baby
    - Enter for a chance to win: email me how many plants + musical instruments you think I have. Closest to the correct answer wins!

Final paper

Final paper due **Wednesday next week @ 11:59pm**.

- Email PDF to me and the TAs (one submission per team)
- Sample papers from the past that were exceptional
  - InstaWaves
  - PaperSphere
  - Foodies
  - NOITCUA

Final paper

Elements of a good final paper submission:

- Clean organization
- Clear, precise (non-wordy) writing
- Details! Pictures!
  - How does your application work? How is it architected? How do your load tests work? Etc.
  - Collect data during your load tests. Present the most relevant data and summarize it (don't just paste a bunch of graphs or charts with no explanation).
  - What kinds of notable things happened during your project? Feel free to include them.
- Focus on scaling
  - Many different scaling tools/methods attempted (more = better)
  - Thoughtful/correct analysis of results (even if your response is "we can't explain why this is happening")

Final presentation

Your team will present your project to the class during the last lecture & lab (12/10 & 12/11).

- Each team will have 12.5 minutes to present
- Team presentation order will be randomized
- Demo your project and prepare a slide deck with key results from your final paper
    - Each teammate should speak during the presentation!
    - You must be present for your team's final presentation!

Grading

Your grade will be based on 3 primary factors:

- Your final project paper.
- Your final project presentation.
- Your teammates' evaluations of your contributions.

Also considered, to a lesser degree:

- Weekly progress
- Participation (in lecture, assistance to other students on Piazza, etc.)

Grading

I will send a survey to you after the final presentation asking you to evaluate your contributions as well as your teammate's contributions. You must respond.

If it doesn't seem right that all teammates receive the same final letter grade, please let me know.

Grading

Final words:

- The class project was revamped this quarter so I'll be taking that into consideration 😉
- The final project is very open-ended by design. Try not to get stuck in any one place while there are other things you can try/investigate.
- If you do get stuck somewhere, make a note of it in your final paper.
  - What were you trying to do? What was the problem? Try to find workarounds.
- Please remember to clean up all your terraform resources after you provision them.
  - You should only keep resources provisioned while you're actively testing them.
- Push all your code to GitHub!
  - Except your terraform directory & secrets 😅

Scaling toolset

Load testing

- k6
  - Try different scaling scenarios
- Tsung
  - Use graphs find the exact point of scaling failure
  - Use previous class projects for help writing your tsung script
- Hey, Vegeta
  - Try them out! They're easy to use

In your final paper, try using different tools. Explain which work best and what you learn from each.

Scaling toolset

API

- Change your API schema to optimize your fetches
- Add pagination for large result sets
- Use GraphQL subscriptions instead of polling
- Implement a piece of your API without GraphQL (ordinary REST)
- Use a faster GraphQL server

Scaling toolset

Database

- **`EXPLAIN`** / optimize MySQL queries; add table indices for faster lookups
- Experiment with alternative data models (column types, table schemas)
- Use dataloader to batch fetches
- Connection pool tuning
- Don’t use TypeORM! Write some critical SQL statements from scratch

(not for the class project)

- Read replication (splitting reads vs. writes)
- Sharding
- Vertical scaling - (NoSQL!!)

Scaling toolset

Caching

- Use a Redis cache on the server to save & serve data between requests
- Add client side caching (e.g. local storage) to save data between page loads
- Client-side caching with HTTP response headers

Scaling toolset

Servers

- Give your server more RAM or CPU
- Add more servers
- Use serverless functions to decompose small pieces of costly application code
- Add a background process to your application
  - Run the background process on a separate server!
- Try turning off server side rendering!
  - Make sure while you're using server-side rendering, you use **`SchemaLink`** instead of **`HttpLink`**

Scaling toolset

Miscellaneous

- Optimize inefficient code! Beware of O(n) vs O(n^2) algorithms!

```
const myInstruments = ['piano', 'bass', 'guitar']
for (const item of list) {
  if (myInstrument.includes(item)) {
    ...
  }
}
```

Scaling toolset

Miscellaneous

- Optimize inefficient code! Beware of O(n) vs O(n^2) algorithms!

```
const myInstruments = new Set(['piano', 'bass', 'guitar'])
for (const item of list) {
  if (myInstrument.has(item)) {
    ...
  }
}
```

Scaling toolset

Miscellaneous

- Optimize inefficient fetches! Beware of sequential async operations.

```
const list = ['piano', 'bass', 'guitar']
for (const item of list) {
  await fetch('https://google.com?q=' + item)
}
```

Scaling toolset

Miscellaneous

- Optimize inefficient fetches! Beware of sequential async operations.

```
const list = ['piano', 'bass', 'guitar']
const promises = list.map(item => fetch('https://google.com?q=' + item))
await Promise.all(promises)
```

Scaling toolset

Miscellaneous

- Optimize inefficient fetches! Beware of sequential async operations.
  - *Fetch using batches whenever you can!*

```
const list = ['piano', 'bass', 'guitar']
await fetch('https://google.com?q=' + list))
```

Scaling toolset

Miscellaneous

- Add automated tests!

```
Run | Debug
test('Math.random() returns value between 0 and 1', () => {
  expect(Math.random()).toBeGreaterThanOrEqual(0)
  expect(Math.random()).toBeLessThan(1)
})
```

Terraform (continued)

In this class we're using **terraform** to manage the infrastructure for running our application in production.

Terraform takes a collection of files describe all the infrastructure you want:

- ***resources***: things we want to create (e.g servers, databases, security groups, networking rules)
- ***variables***: data we can pass into our resource definitions (e.g. amount of RAM/CPU per server)
- ***outputs****:* information about resources (e.g. server URLs) printed by terraform for your convenience
- ***modules***: a collection of ^ all of the above, useful for code organization

Terraform

Given a set of *resource* definitions, I can either:

- `terraform plan`
  - Instructs terraform to diff the current state of your provisioned resources against your resource definitions. The plan tells you the sequence of (create/update/destroy) operations terraform would take.
- `terraform apply`
  - Runs the terraform plan.

The result of `terraform apply` is a **terraform state file** (`terrraform.tfstate`) which is a JSON representation of all the resources provisioned during the last apply step.

Terraform

The result of `terraform apply` is a **terraform state file** (`terrraform.tfstate`) which is a JSON representation of all the resources provisioned during the last apply step.

On the next `terraform apply`, terraform uses the prior state to present a diff (so I know exactly what terraform plans to do to get my infrastructure up to date).

Terraform

```
resource "aws_db_instance" "db" {
 name                   = "thecoolestdatabase"
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Terraform

```
resource "aws_db_instance" "db" { // the type of resource we want, see a complete list
 name                   = "thecoolestdatabase"
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Terraform

```
resource "aws_db_instance" "db" { // the name given to our resource, all names for a particular resource type must be unique
 name                   = "thecoolestdatabase"
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Terraform

```
resource "aws_db_instance" "db" {
 name                   = "thecoolestdatabase" // key-value pairs configure the resource
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Terraform

```
resource "aws_db_instance" "db" {
 name                   = "thecoolestdatabase"
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn // values may reference other resources, in this case the "arn" field of the aws_kms_key resource named "db_key"
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Terraform

```
resource "aws_db_instance" "db" {
 name                   = "thecoolestdatabase"
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group // values may also refer to variables, in this case the variable named "subnet_group"
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Packaging and deploying your app

⁉️ **What do we need in order to deploy our application?**

Packaging and deploying your app

⁉️ **What do we need in order to deploy our application?**

- MySQL

`terraform/main.tf`

```
module "mysql" {
  source = "./modules/mysql"
}
```

Packaging and deploying your app

⁉️ **What do we need in order to deploy our application?**

- Redis

`terraform/main.tf`

```
module "redis" {
  source = "./modules/redis"
}
```

Packaging and deploying your app

⁉️ What do we need in order to deploy our application?

- Server(s), and networking to reach those servers

`terraform/main.tf`

```
module "webserver" {
  source = "./modules/appserver"

  appserver_tag  = "app-web"
  ecr_repository = aws_ecr_repository.cs188.repository_url
  ecs_cluster    = aws_ecs_cluster.cs188.id

  mysql_host = module.mysql.host
  redis_host = module.redis.host

  honeycomb_key = "d29d5f5ec24178320dae437383480737"

  ws_url = module.websocket_api.url
}
```

Packaging and deploying your app

⁉️ **What do we need in order to deploy our application?**

- Server(s), and networking to reach those servers

`terraform/main.tf`

```
module "lambda" {
  source = "./modules/lambda"

  honeycomb_key = "d29d5f5ec24178320dae437383480737"

  mysql_host = module.mysql.host
  redis_host = module.redis.host
}
```

Packaging and deploying your app

⁉️ **What do we need in order to deploy our application?**

- APIs

`terraform/main.tf`

```
module "rest_api" {
  source         = "./modules/rest_api"
  appserver_host = module.webserver.host
}

module "websocket_api" {
  source         = "./modules/websocket_api"
  appserver_host = module.webserver.host
}
```

Packaging and deploying your app

⁉️ **How do I make my server larger (vertical scaling)?**

`terraform/modules/appserver/variables.tf`

```
variable "cpu" {
  description = "Fargate instance CPU units to provision (1 vCPU = 1024 CPU units)"
  default     = "1024"
}
variable "memory" {
  description = "Fargate instance memory to provision (in MiB)"
  default     = "2048"
}
```

Packaging and deploying your app

⁉️ **How do I add more servers (horizontal scaling)?**

`terraform/modules/appserver/main.tf`

```
resource "aws_ecs_service" "appserver" {
  name            = "${var.app_name}-${var.appserver_tag}-service"
  cluster         = var.ecs_cluster
  task_definition = "${aws_ecs_task_definition.appserver.family}:${aw
  desired_count   = "1"
  launch_type     = "FARGATE"
```

Packaging and deploying your app

⁉️ **How do I run different kinds of servers (service decomposition)?**

`terraform/main.tf`

```
module "backgroundserver" {
  source = "./modules/appserver"

  appserver_tag  = "app-background"
  services = "BACKGROUND"
  ecr_repository = aws_ecr_repository.cs188.repository_url
  ecs_cluster    = aws_ecs_cluster.cs188.id
```

Packaging and deploying your app

⁉️ How do I run different kinds of servers (service decomposition)?

`terraform/main.tf`

```
module "backgroundserver" {
  source = "./modules/appserver"

  appserver_tag  = "app-background"
  services = "BACKGROUND"
  ecr_repository = aws_ecr_repository.cs188.repository_url
  ecs_cluster    = aws_ecs_cluster.cs188.id
```

^ it's almost identical to how we provisioned our other appserver!

Our code lives in a monorepo...

...but we can still deploy on multiple servers. Each of those servers can receive different environment variables that dictate what we want the code running on those servers to do. Our code can conditionally start work/processes depending on the environment.

See `config.ts` for how to read environment variables in your node app.

Packaging and deploying your app

⁉️ **Ok, we've provisioned everything with terraform. How do we actually deploy the code?**

Packaging and deploying your app

⁉️ Ok, we've provisioned everything with terraform. How do we actually deploy the code?

- Our code is a collection of TypeScript files and static resources (e.g. lecture slides)
- We must compile the TypeScript to JavaScript so we can run it with `node`, and put all the static files in the correct folder structure
  - The TypeScript code that runs in the browser must be compiled to a single JavaScript file (bundle), and served as a static resource
  - In another language like Go, we might compile everything down to a binary executable format including the static resources!

(This is done for you by `npm run build`)

Packaging and deploying your app

⁉️ **Ok, we've provisioned everything with terraform. How do we actually deploy the code?**

- To run an appserver on ECS, you build a docker image that runs your code. You push that image to a repository, and tell ECS to (re-)deploy the app using the new image version.
- To downgrade (rollback), you tell ECS to re-deploy the app using an old image version. The repository contains all versions, similar to git.

ℹ️ ECS and AWS load balancer manages the deployment such that there's zero downtime. You may have multiple servers running at once during deployment!

Packaging and deploying your app

⁉️ **Ok, we've provisioned everything with terraform. How do we actually deploy the code?**

- To run code on AWS lambda, you must create a .zip file containing your source/executables.
- Your code must implement a specific interface to run on lambda. See `server/src/lambda/handler.ts`.

Packaging and deploying your app

⁉️ **Ok, we've provisioned everything with terraform. How do we actually deploy the code?**

- To run code on AWS lambda, you must create a .zip file containing your source/executables.
- Your code must implement a specific interface to run on lambda. See `server/src/lambda/handler.ts`.

Update `bundle-server.sh` to create the .zip. Update `deploy-local.sh` to publish the new code to your AWS lambda function.